der Bundeswehr
Universität München

Hierarchisches Clustering von Kinect-Bewegungsdaten mittels Dynamic Time Warping

Bachelorarbeit von

Laurenz Fuchs

1196307

Draft vom 29. März 2022

Erstprüfer:	Prof. Dr. Michael Koch
Zweitprüfer:	Prof. Dr. Gunnar Teege
Betreuer:	M. Sc. Julian Fietkau
Abgabetermin:	31. Mai 2022

Universität der Bundeswehr München

Fakultät für Informatik

Abstract

Hier steht eine kurze Zusammenfassung der Arbeit. Sie darf nicht länger als eine Seite sein, sollte aber mindestens eine halbe Seite umfassen.

Inhaltsverzeichnis

1 Einleitung und Motivation

1.1 Einleitung

1.2 Motivation

Von September 2021 bis August 2024 forschen die Hochschule für angewandte Wissenschaften in Hamburg und die Universität der Bundeswehr München im Rahmen des *HoPE-Projekts* zusammen an Effekten rund um die Steigerung der Aufmerksamkeit bei der Nutzung von großen, interaktiven Wandbildschirmen, sogenannten *Ambient Displays*. Dieser Bereich weist noch einen grundsätzlichen Forschungsbedarf auf. Im Projekt wird unter anderem der *Honeypot-Effekt* erforscht (UniBw, 2021). Dieser beschreibt in der Mensch-Computer-Interaktion (HCI) wie Menschen, die mit einem System interagieren, weitere Passanten anregen die Interaktion zu beobachten oder sogar an ihr teilzuhaben (Wouters et al., 2016). Der *Honeypot-Effekt* soll bei der Nutzung von *Ambient Displays* im öffentlichen und halböffentlichen Raum in Langzeit-Feldstudien analysiert werden. Dabei soll auch der Aspekt der Datenerhebung und -analyse weiter ausgebaut werden. Hierzu muss ein methodisches Rahmenwerk entwickelt werden, welches eine auf Sensordaten-basierende, automatische und zeitlich uneingeschränkte Evaluation von *Ambient Displays* ermöglicht (UniBw, 2021). Die Sensordaten werden von Body-Tracking-Kameras bereitgestellt. Konkret wurden die Wandbildschirme im vorliegenden Datensatz mit Microsoft Kinect v2 Kameras ausgestattet. Diese zeichnen die Interaktion von Nutzern mit den Displays auf, wodurch das Nutzerverhalten zu einem späteren Zeitpunkt ausgewertet werden kann. Die Daten liegen als Time-Series Data (TSD) vor. Bei diesem Datentyp handelt es sich um geordnete Sequenzen von Datenpunkten, die über eine gewisse Zeit hinweg aufgenommen werden (Ali et al., 2019). TSD enthalten oft wichtige Informationen die durch Mustererkennung entdeckt werden können. Eine gängige Methode zur Analyse sind hierbei *Clustering*-Verfahren (Aghabozorgi et al., 2015). Lässt sich eine Menge von Kategorien identifizieren, die etwas über das Verhalten von Menschen vor Wandbildschirmen aussagt? Ist die Einteilung in diese Kategorien automati-

sierbar? Aufgrund der Größe des vorliegenden Datensatzes ist eine manuelle Evaluation zur Beantwortung der Fragen nicht möglich. In dieser Bachelorarbeit wird daher versucht, das Problem mithilfe von deterministischen Algorithmen zu lösen. Wesentliches Ziel ist die Implementierung eines Systems zur Kategorisierung der vorliegenden Kinect-Bewegungsdaten. Dieses System wird detailiert beschrieben und anschließend evaluiert. Dabei soll *hierarchisches Clustering* mithilfe des *Dynamic Time Warping (DTW)*-Algorithmus eingesetzt werden. Außerdem wird die Frage beantwortet, welches Vorwissen benötigt wird, um das Verfahren einsetzen zu können und welche Attribute zur Analyse interessant sind (z.B. Laufpfade oder Engaged-Werte).

2 Grundlagen und Problembeschreibung

Bevor näher auf die verwendeten Algorithmen eingegangen wird, werden grundlegende Aspekte erläutert. Dabei werden neben einigen Interaktionsmodellen vor Wandbildschirmen auch die eingesetzte Sensorik und die Struktur des Datensatzes beschrieben. Zudem erfolgt abschließend eine ausführliche Problembeschreibung.

2.1 Kategorien der Interaktion mit Wandbildschirmen

Interaktive digitale Medien sind in der Öffentlichkeit immer präsenter. Deshalb wird es für Wandbildschirme immer schwieriger die Aufmerksamkeit von Passanten zu erregen und sie zur Interaktion zu animieren. Diese Herausforderungen können nicht einfach durch verbesserte Hardware oder attraktivere Displays gelöst werden. Stattdessen muss ein besseres Verständnis von Menschen und deren Technologienutzung geschaffen werden (Wouters et al., 2016). *Ambient Displays* sind große, interaktive Bildschirme im (halb-) öffentlichen Raum, mit denen Nutzer interagieren können. Es handelt sich meist um ästhetisch ansprechende Displays die Personen mit Informationen versorgen (Mankoff et al., 2003). Eine Kategorisierung der Interaktion von Personen mit solchen Wandbildschirmen kann das Verständnis des Verhaltens verbessern. Dazu existieren verschiedenste *Audience Behaviour-Interaktionsmodelle*, wovon im Folgenden zwei näher beschrieben werden.

Das *Audience Funnel Modell* beschreibt, wie Menschen sich um ein großes öffentliches Display versammeln und von Beobachtern zu Interagierenden mit dem System, und anschließend wieder zu Beobachtern werden. Menschen neigen dazu verschiedene Phasen der Interaktivität zu durchlaufen, bevor sie direkt mit dem System interagieren (Mai & Hußmann, 2018; Wouters et al., 2016). Die einzelnen Phasen des *Audience Funnel* werden in Abbildung 2.1 gezeigt. Eine der Aufgaben eines Wandbildschirms ist es also Aufmerksamkeit auf sich zu ziehen und den Nutzer zu motivieren mit dem System zu interagieren (Mai & Hußmann, 2018). Mai und Hußmann (2018) verweisen darauf, dass *Ambient Displays* in der Öffentlichkeit nicht undbedingt der zentrale Punkt der Aufmerksamkeit sind, da vorbei-

gehende Personen eigene intrinsische Ziele verfolgen. Die Herausforderung für Entwickler ist es die Systeme so zu gestalten, dass sie Aufmerksamkeit erregen, sich aber gleichzeitig nicht gezwungen in den Mittelpunkt stellen.

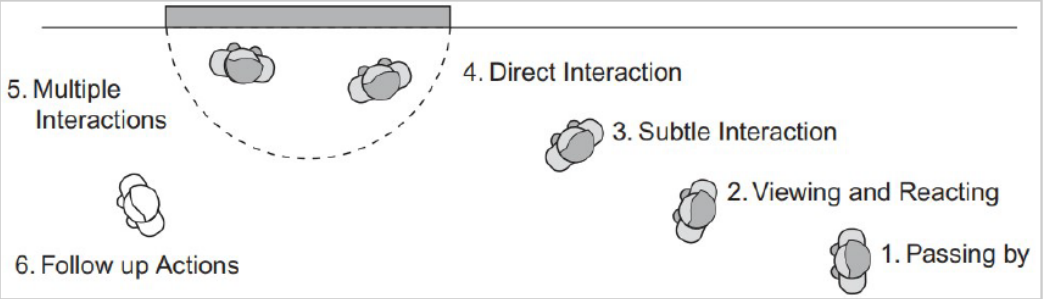


Abbildung 2.1: Audience Funnel Framework. Abbildung aus Mai und Hußmann (2018).

Ein zweites Modell wird durch den bereits erwähnten *Honeypot-Effekt* beschrieben. Er zeigt, dass Individuen undabhängig von Belohnungen, Bestrafungen oder sozialem Wettbewerb von der reinen Präsenz oder den Aktivitäten anderer beeinflusst werden. In der HCI wird dies meist erkennbar, indem Passanten sich einem System nähern und überlegen, ob sie mit ihm interagieren sollen, nachdem sie anderen Menschen dabei zugesehen haben (Wouters et al., 2016).

Eine Einordnung der Bewegungsdaten in die Kategorien solcher Modelle und eine weiterführende Analyse der Bewegungen kann Aufschluss über das Verhalten von Menschen vor interaktiven Bildschirmen geben. Wie bereits erwähnt ist eine Kategorisierung des vorliegenden Kinect-Datensatzes aufgrund der großen Datenmenge nicht manuell realisierbar. Im Folgenden werden die Struktur des Datensatzes und die Grundlagen der verwendeten Sensorik beschrieben. Aufbauend darauf werden Überlegungen angestellt, wie eine Implementierung zur Automatisierung der Kategorisierung aussehen kann.

2.2 Spezifikation der Kinect

(Tölgyessy et al., 2021) verweisen darauf, dass der Xbox 360 Kinect-Sensor eine „Revolution“ im Bereich der erschwinglichen 3D-Erkennungssensorik war. Ursprünglich war er für die Videospiel-Industrie gedacht. Schon bald wurde er aber auch für wissenschaftliche Experimente genutzt. In späteren Jahren folgten weitere Iterationen der Kinect (Tölgyessy et al., 2021). Im vorliegenden Datensatz des *HoPE-Projekts* kam die Kinect v2 für Xbox One

zum Einsatz. Diese Sensorik stellt Farbbilder einer Rot-Grün-Blau (RGB) Kamera, Tiefenbilder einer Tiefenkamera und Audiodateien verschiedener Mikrofone zur Verfügung (Microsoft, 2014). Besonders die Tiefenkamera hilft zuverlässige Ergebnisse bei der Erkennung von Menschen vor *Ambient Displays* zu erzielen. Li et al. (2014) fassen es wie folgt zusammen. Die kompakte Größe, die Benutzungsfreundlichkeit, die stark vereinfachte Hintergrund-Subtraktion im Vergleich zu anderer Sensorik, sowie die hohe Genauigkeit und die hohe Bildrate machen Tiefenkameras zu einer attraktiven Lösung für ein breites Spektrum an Anwendungen. Die Kinect v2 verwendet dabei den Ansatz der kontinuierlichen Wellenintensitätsmodulation, der häufig bei Time-of-Flight (ToF)-Tiefenkameras zum Einsatz kommt. Dabei wird das Licht einer Lichtquelle von Objekten im Sichfeld der Kamera zurückgestreut und die Phasenverzögerung zwischen dem emittierten und dem reflektierten Licht gemessen. Diese Phasendifferenz wird für jedes Pixel im Bildfeld in einen Entfernungswert umgerechnet (Tölgyessy et al., 2021). Der Sensor kann Tiefenbilder mit einer Auflösung von 512 x 424 Pixeln und gewöhnliche Farbbilder mit 1920 x 1080 Pixeln aufnehmen (Marin et al., 2019). Bei der Kinect v2 können bis zu sechs Personen erfasst werden. Dabei wird die Lage von 25 Skelettpunkten, sowie verschiedene Gesichtsattribute erfasst (Microsoft, 2014). Abbildung 2.2 zeigt eine Übersicht dieser Punkte.

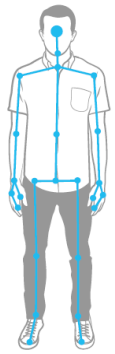


Abbildung 2.2: Skelettpunkte der Kinect v2. Abbildung aus Microsoft (2014).

2.3 Struktur des vorliegenden Datensatzes

Zur Evaluation des implementierten System dienen verschiedene Teilmengen des Kinect-Datensatzes des *HoPE-Projekts*. Hierfür wurden im Jahr 2017 für 18 Wochen *Ambient Displays* mit Kinect v2 Systemen ausgestattet. Zu erwähnen ist zudem, dass das entwickelte

Tool zur Auswertung auch mit anderen Datensätzen funktionieren soll.

Eine Auswertung des Datensatzes ergab, dass dieser 97.626 Records beinhaltet (Temiz, 2022). Dabei enthält jeder mehrere sogenannte Frames. Dabei handelt es sich um Momentaufnahmen der Kinect-Sensorik. Falls sich mehrere Menschen gleichzeitig im Sensorbereich befinden, erhält jede Person eine eindeutige Identifikationsnummer und der Record wird in mehrere Frames unterteilt. Dabei entspricht ein Frame der Momentaufnahme einer Person. Zu jedem Record liegen verschiedene Dateien vor, die jeweils den Zeitstempel in Kombination mit einem geeigneten Postfix als Dateinamen tragen. Die Datei *timestamp.txt* enthält folgende Attributwerte:

Attributname	Beschreibung
Zeitstempel	Zeitpunkt der Aufnahme
KinectId	Skelett-Identifikationsnummer des Framework
RecordId	Identifikationsnummer des Records
BodyIndex	Identifikationsnummer der Person
BodyCount	Anzahl der erfassten Personen
Happy	Person ist glücklich
Engaged	Person zeigt Interesse
WearingGlasses	Person trägt eine Brille
LeftEyeClosed	Person hat das linke Auge geschlossen
RightEyeClosed	Person hat das rechte Auge geschlossen
MouthOpen	Person hat den Mund geöffnet
MouthMoved	Person bewegt den Mund
LookingAway	Person schaut nicht zur Kinect
Body.HandLeftState	Zustand der linken Hand
Body.HandRightState	Zusand der rechten Hand
X	x-Koordinate des Skelettpunkts SpineShoulder
Y	y-Koordinate des Skelettpunkts SpineShoulder
Z	z-Koordinate des Skelettpunkts SpineShoulder
Distance	Distanz zwischen Kinect und Skelettpunkt SpineShoulder

Diese Attribute werden in der Textdatei durch drei Rauten ('###') voneinander abgegrenzt. Ein konkreter Frame aus dem Datensatz kann Abbildung 2.3 entnommen werden.

Da die Anordnung und das Vorhandensein dieser Attribute je nach Version des Datensatzes abweichen kann, sollen diese in der Implementierung manuell konfigurierbar sein.

2017-04-10 07:41:53.943 +02:00 ### 72057594038063128 ### 1341053376 ### ### 1 ### 1 ### No ### Maybe ### Unknown ### No ### No ### No ### Yes ### Maybe ### Unknown ### Closed ### 0,1074784 ### 0,2451882 ### 4,18441 ### 4,19296454679429

Abbildung 2.3: Frame aus dem Datensatz.

Die Datei *timestamp_bodies.txt* enthält für jeden Frame die Position aller 25 Skelettpunkte. *timestamp_summary.txt* zeigt eine kurze Zusammenfassung des Records, welche gut menschenlesbar ist. *timestamp.xef* kann genutzt werden, um die Aufnahme in der Anwendung Kinect Studio zu visualisieren. Letztlich sind die Einträge sogenannte *TSD*. Bei diesem Datentyp handelt es sich um geordnete Sequenzen von Datenpunkten, die über eine gewisse Zeit hinweg aufgenommen wurden. Oft in regelmäßigen Abständen (Ali et al., 2019). Insgesamt befinden sich im Datensatz 34.687.630 Frames (Temiz, 2022). Diese Anzahl bestätigt erneut die Notwendigkeit eines Software-Tools zur Auswertung. Um aussagekräftige Ergebnisse zu erhalten lohnt es sich gegebenenfalls mit Teilmengen des Datensatzes zu arbeiten. So liefern beispielsweise Records mit weniger als zwei Sekunden zu wenig Information für Erkenntnisse über das Interaktionsverhalten. Je nach Fragestellung kann es ebenfalls lohnenswert sein Records mit einer, zwei oder mehreren Personen gesondert zu betrachten.

2.4 Problembeschreibung

Es gibt verschiedene Möglichkeiten sich der Kategorisierung von Bewegungsdaten zu nähern. So kann beispielsweise *Maschine Learning* eingesetzt werden. Dieser Ansatz wird derzeit ebenfalls im Rahmen des HoPE-Projekts erforscht (Plischke, 2022). In dieser Bachelorarbeit werden hingegen *deterministische Algorithmen* genutzt. Durch sie kann die Implementierung eines Kategorisierungssystems verhältnismäßig einfach gehalten werden. Aghabozorgi et al. (2015) verweisen zudem darauf, dass die Verwendung von Methoden wie Maschine Learning (*supervised*), im Falle von großen Datensätzen zu Problemen führen kann. Deterministische Cluster-Algorithmen sollten bei großen Datenmengen weniger Probleme bereiten (*unsupervised*).

Beim vorliegenden Kinect-Datensatz handelt es sich ebenfalls um eine große Datenmenge. Die einzelnen Records müssen daher automatisiert bearbeitet werden. Ziel der Verarbeitung ist es möglichst sinnvolle *Cluster* zu erhalten. Um die Gemeinsamkeiten zweier Aufnahmen

zu berechnen, wird eine geeignete Vergleichsfunktion benötigt. Die Wahl dieser Funktion ist wichtig für den Erfolg des *Clusterings* (Warren Liao, 2005). Zu beachten ist zudem, dass die Bewegungsaufnahmen durch *TSD* beschrieben werden. Werden hier herkömmliche Distanzmetriken, wie die Euklidische-Distanz verwendet kann dies zu Problemen führen und die Aussagekraft des Vergleichs verringern. Es kann vorkommen, dass verschiedene Personen die gleiche Bewegung unterschiedlich schnell, oder zeitlich versetzt ausführen. Gegebenenfalls weisen die Datenreihen bei der gleichen Bewegung daher sogar eine unterschiedliche Anzahl an Frames auf. Abbildung 2.4 zeigt ein exemplarisches Szenario. Die Kurven weisen ähnliche Segmente auf. Sie sind allerdings unterschiedlich lang und die Ähnlichkeiten treten zu unterschiedlichen Zeitpunkten auf. Mit der bekannten euklidischen Distanz ist das Ergebnis hier nicht aussagekräftig und der letzte Punkt der Reihe kann gar nicht zugordnet werden. Um dieses Problem zu lösen ist eine elastische Metrik nötig, die besser mit zeitlichen Verschiebungen umgehen kann (Aghabozorgi et al., 2015).

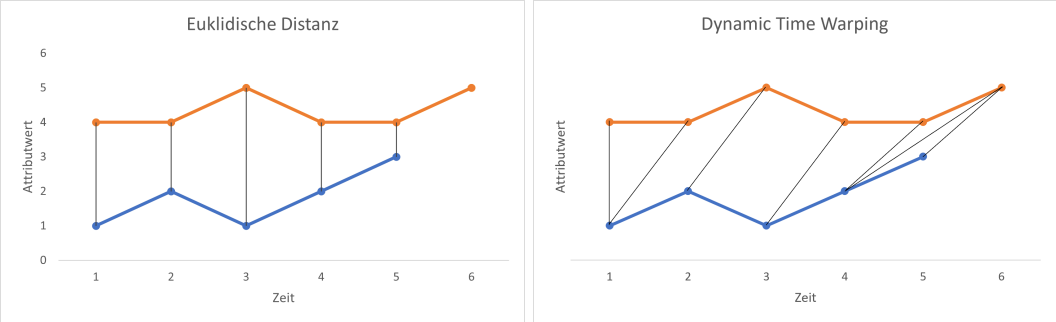


Abbildung 2.4: Zuorndung der Messpunkte zweier Datenreihen.

Das Clustering soll mithilfe von *hierarchischem Clustering* durchgeführt werden, welches in Abschnitt 3.1 beschrieben wird. Das Verfahren ist für die vorliegenden Daten gut geeignet, da es erlaubt TSD unterschiedlicher Länge zu clustern. Zudem muss die Anzahl der zu bildenden Cluster nicht im Voraus definiert werden, wie dies etwa bei *K-Means* der Fall ist (Aghabozorgi et al., 2015). Es ist in diesem Kontext nicht zielführend die Clusteranzahl zuvor zu definieren, da die Arbeit darauf abzielt, mehr Erkenntnis über mögliche sinnvolle Cluster zu gewinnen. Statt die Anzahl vorzugeben, wird ein Threshold definiert der angibt, ab wann zwei Cluster nicht mehr zusammengeführt werden soll, weil die Differenz zwischen ihnen zu groß ist. Als geeignete Distanz-Metrik wird dabei *DTW* verwendet (Abschnitt 3.2). Hier ist eine Mehrfachzuordnung von Punkten möglich. Diese können so verbunden werden, dass die

Kosten minimal sind. Dies erlaubt eine Zuordnung ähnlicher Muster in den Daten auch wenn sie zeitlich verschoben sind und ist daher gut für unsere Problemstellung geeignet.

3 Clustering von Bewegungsdaten mittels Dynamic Time Warping

Nach den einführenden Bemerkungen in Kapitel 1 und Kapitel 2 werden nun die eingesetzten deterministischen Algorithmen näher beschrieben. Die Grundlagen der Verfahren werden erläutert und an einem Beispiel veranschaulicht. Ein Einlick in verwandte Arbeiten (Abschnitt 3.4) vertieft das Verständnis weiter. Abschließend wird auf zu beachtende Besonderheiten des Kinect-Datensatzes eingegangen.

3.1 Hierarchisches Time-Series Clustering

Clustering kann genutzt werden, um große Datensätze zu kategorisieren, wenn keine Informationen zu den Kategorien vorliegen (Aghabozorgi et al., 2015). Ziel ist es, eine Struktur in den vorliegenden Daten zu finden, indem Elemente anhand eines festgelegten Vorgehens in homogene Gruppen einsortiert werden. Dabei sollen die Unterschiede bei Elementen der gleichen Gruppe möglichst gering und bei Elementen verschiedener Gruppen möglichst groß sein (Aghabozorgi et al., 2015; Warren Liao, 2005). Bei n Datenpunkten erzeugt ein *Clustering* k Partitionen des Datensatzes. Jede Partition entspricht dabei einem *Cluster*, dass mindestens ein Objekt enthält. Beim *Clustering* sind besonders die Wahl des Clustering-Algorithmus und der Methode zur Bestimmung der Ähnlichkeit von zwei Datenreihen von Bedeutung (Warren Liao, 2005). Letzteres wird DTW gelöst. Das Verfahren wird in Abschnitt 3.2 beschrieben. Als Clustering-Vorgehen kann das sogenannte *hierarchical Clustering* genutzt werden. Dieses funktioniert, indem Datenobjekte in einen Baum von Clustern eingeteilt werden. Bei agglomerativen Vorgehen wird ein *bottom-up* Ansatz verfolgt. Jedes Objekt ist daher zunächst ein eigenes Cluster. Dann werden schrittweise Cluster zusammengeführt, bis alle Objekte im selben Cluster sind, oder eine Terminierungsbedingung erfüllt ist (Warren Liao, 2005). In jedem Iterationsschritt werden dabei die beiden Cluster zusammengeführt, die die geringste Differenz zueinander aufweisen. Die Terminierungsbedingung ist bei uns das Überschreiten

des vordefinierten Thresholds.

Im Fall des Kinect-Datensatzes liegen die Daten als TSD vor. TSD zu kategorisieren findet in vielen wissenschaftlichen Gebieten Anwendung, um Muster zu finden. Sie erlauben es wichtige Informationen aus großen Datensätzen zu extrahieren. TSD sind dynamische Daten, weil sich die Attributwerte mit der Zeit verändern. Diese Sammlungen von Werten können aber auch als ein einziges Objekt betrachtet werden. Das Kategorisieren von derartigen Objekten kann bei der Detektion relevanter Muster helfen (Aghabozorgi et al., 2015). Die meisten bestehenden Cluster-Algorithmen können auf diesen Datentypen zugeschnitten werden. So auch das hierarchische Clustering. Der größte Unterschied ist wie bereits erwähnt, dass die Distanz-/ Ähnlichkeitsmetrik angepasst werden muss, um sinnvoll bei TSD genutzt werden zu können. Ein Vorteil ist, dass hierarchisches Clustern bei der Verwendung von DTW nicht auf Datenreihen der gleichen Länge beschränkt ist (Warren Liao, 2005).

3.2 Dynamic Time Warping Algorithmus

Der *DTW Algorithmus* kann als Distanz-Metrik beim hierarchischen Clustern genutzt werden. Hier sind *one-to-many* oder *one-to-none* Beziehungen zwischen Datenpunkten möglich. Man bezeichnet das Vorgehen daher auch als eine *elastische Metrik*. Sie kann gut mit zeitlichen Verschiebungen und unterschiedlich langen Datenreihen umgehen (Aghabozorgi et al., 2015).

In der Literatur findet man viele Erläuterungen des DTW-Algorithmus. Beispielsweise in Mohammadzade et al. (2021), Warren Liao (2005), Aghabozorgi et al. (2015) oder Yu und Xiong (2019). Im Folgenden soll der Algorithmus ebenfalls erläutert werden. Das Ziel ist die Zuordnung verschiedener Signale, die im Verlauf der Zeit ähnliche Muster aufweisen. Diese Muster treten zu unterschiedlichen Zeiten oder in unterschiedlichen Raten auf. Um die Zuweisung zueinander zu erreichen muss ein Signal gegebenenfalls lokal gestreckt oder gestaucht werden. Durch *dynamisches Programmieren* kann eine Korrespondenz zwischen den Zeitindizes zweier Signale gefunden werden, sodass die Summe der Distanzen zwischen den beiden Signalen minimal ist (Mohammadzade et al., 2021).

Seien $Q = q_1, q_2, ..., q_i, ..., q_n$ und $R = r_1, r_2, ..., r_j, ..., r_n$ zwei Signale die durch DTW ausgerichtet werden sollen, sodass die Differenz zwischen ihnen minimal ist. Dafür wird eine $n\ x\ m$ Matrix M gebildet, wobei das Element *(i, j)* die Distanz $d(q_i, r_j)$ zwischen zwei Punkten q_i und r_j enthält. Hierfür wird im Normalfall die euklidische Distanz genutzt (Warren Liao, 2005). Allerdings sind auch andere Funktionen denkbar. In der Zelle *(0, 0)* enthält die Matrix

den Wert 0 und für die übrigen Zellen der Spalte und Zeile 0 wird jeweils der Wert unendlich angenommen. Das Verfahren kann auf einfache Weise auf mehrdimensionale TSD angepasst werden. In diesem Fall sind die zu minimierenden Kosten die Summe der Distanzen zwischen den korrespondierenden Werten der beiden Datenreihen (Mohammadzade et al., 2021). Ein *Warping Pfad* $W = w_1, w_2, ..., w_k, ..., w_K$ in dem $max(m, n) \leq K \leq m + n - 1$ gilt, ist eine Reihe von Matrixelementen die drei Bedingungen erfüllen:

- Grenzbedingungen
- Kontinuität
- Monotonität

Die Grenzbedingungen fordern, dass die Startzelle des Warping Pfads in der Matrix eine andere ist als die Zielzelle. Dabei gilt: w_1 = *(m, n)* und w_k = *(0, 0)*. Aufgrund der Kontinuität sind nur Schritte in angrenzende Zellen erlaubt. Die Monotonitätsbedingung zwingt den Warping Pfad dazu, sich monoton in der Zeit zu bewegen.

Mit Hilfe von dynamischer Programmierung können die Matrixeinträge von $M_{(0,0)}$ ausgehend berechnet werden, indem wiederholt die folgende Gleichung ausgewertet wird. $M_{(i,j)} = d(q_i, r_j) + min\ \{M_{(i-1,j-1)}, M_{(i-1,j)}, M_{(i,j-1)}\}$ Sie definiert den kumulativen Abstand als die Summe aus dem Abstand des aktuellen Elementes und dem Minimum der kumulativen Abstände der benachbarten Elemente (Warren Liao, 2005). Der Pfad mit der minimalen Distanz ist von Bedeutung. Er kann mit $d_{DTW} = min(\frac{\sum_{k=1}^{K} w_k}{K})$ berechnet werden.

3.3 Veranschaulichendes Beispiel

Zur Veranschaulichung soll nun das Vorgehen anhand von vier Datenreihen durchgeführt werden. Seien R_1 = *(0, 1, 1, 2, 2, 1, 1)*, R_2 = *(0, 1, 1, 1, 2, 2, 1, 1)*, R_3 = *(1, 1, 0, 1, 2, 2, 1, 1)* und R_4 = *(2, 2, 1, 2, 2, 2)* gegeben. Die Kosten können berechnet werden, indem die Kostenmatrix aufgestellt und der Warping Pfad berechnet wird (Abschnitt 3.2). Für die Berechnung der Kosten zwischen R_1 und R_2 ergibt sich beispielsweise folgende Matrix mit hervorgehobenem Warping Pfad.

0	∞	∞	∞	∞	∞	∞	∞	∞
∞	**0**	1	2	3	5	7	8	9
∞	1	**0**	**0**	0	1	2	2	2
∞	2	0	0	**0**	1	2	2	2
∞	4	1	1	1	**0**	0	1	2
∞	6	2	2	2	0	**0**	1	2
∞	7	2	2	2	1	1	**0**	0
∞	8	2	2	2	2	2	0	**0**

Außerdem ergeben sich folgende gerundete Kosten zwischen den einzelnen Reihen:

Reihen	Kosten
R_1, R_2	**0**
R_1, R_3	1.56
R_1, R_4	2.88
R_2, R_3	2.89
R_3, R_4	2.67

Da R_1 und R_2 zusammen die geringsten Kosten haben bilden sie das erste Cluster C_1. Um dessen Wertreihe zu erhalten werden die Attributwerte paarweise addiert und die Summe anschließend halbiert. Dabei kann anhand des Warping Pfads abgelesen werden, welche Werte zusammengehören. Die Bildung der Paare entspricht also der Zuordnung mit den geringsten Kosten. So ergibt sich C_1 = *(0, 1, 1, 1, 2, 2, 1, 1)*. Falls mehrere verschiedene Attribute berücksichtigt werden kann für sie eine kombinierte Matrix erstellt werden, indem für jedes Attribut eine Matrix erstellt wird und deren Mittelwert in eine neue Matrix geschrieben wird. Da all diese Matrizen das selbe Format haben, ist dies problemlos möglich. Der Warping Path kann dann an dieser kombinierten Matrix abgelesen werden. Die Bildung der Paare erolgt entsprechend. Es sollte ein *Threshold* definiert werden, der angibt, ab wann die Kosten zu groß sind, um die beiden Datenreihen zu einem Cluster zusammenzuführen. Ein sinnvoller Wert hängt dabei vom vorliegenden Datensatz ab. Sei dieser Threshold im Folgenden t = *2*. Nun werden die Schritte wiederholt, wobei C_1 als eine einzelne neue Datenreihe betrachtet wird. Dabei ergeben sich folgende Kosten:

Reihen	Kosten
C_1, R_3	**1.56**
C_1, R_4	2.89
R_3, R_4	2.67

Nun haben C_1 und R_3 die geringsten Kombinationskosten. Da die Kosten geringer sind als der Threshold kann ein neues Cluster gebildet werden. Es ergibt sich C_2 = *(0.5, 1, 0.5, 1, 2, 2, 1, 1)*. Damit können die neuen Kosten berechnet werden:

Reihen	Kosten
C_2, R_4	**2.78**

Da die berechneten Kosten den zuvor definierten Threshold übersteigen wird kein neues Cluster mehr gebildet und wir sind am Ende unserer Berechnungen angekommen. Abbildung 3.1 veranschaulicht die gebildeten Cluster.

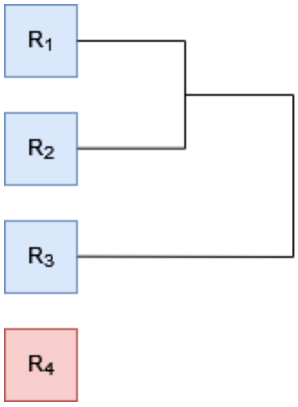


Abbildung 3.1: Graphische Veranschaulichung der gebildeten Cluster.

3.4 Related Work Analyse

DTW kommt in vielen Arbeiten als Distanz-Metrik zum Einsatz. Im Folgenden wird eine Auswahl dieser Veröffentlichungen beschrieben.

Wahyuni et al. (2021) stellen ein Erkennungssystem für Gesten vor. Dabei wird eine Kinect-Kamera genutzt, um die Bewegungen aufzuzeichnen. Es kommt eine Form des DTW-Algorithmus zum Einsatz, um die Gesten zu erkennen. Für den Vergleich werden die x- und y-Koordinaten von sechs Körperpunkten genutzt. Dabei liegen zu jeder bekannten Bewegung jeweils Referenzwerte für den Vergleich mit den aktuellen Werten vor. Für die Entscheidung, ob eine betrachtete Bewegung zu einer Kategorie gehört oder nicht sind die Kosten entscheidend. Es wird ein Threshold definiert. Liegen die aktuellen Kosten über diesem wird die Bewegung verworfen. Falls sie geringer sind wird der Ablauf in die Kategorie einsortiert. Die Definition von geeigneten Threshold-Werten für die Differenz zwischen Datenreihen wird

auch im späteren Verlauf dieser Arbeit relevant. So kann entschieden werden, ob Cluster weiter zusammengeführt werden sollten oder nicht. Yu und Xiong (2019) nuzten DTW, um physische Rehabilitationsübungen mithilfe eines Kinect-Sensors zu bewerten. Dabei wird zunächst die Ähnlichkeit zwischen den TSD des Nutzers und des Trainers berechnet. Als Eingabeattribute werden hier acht sogenannte Knochenvektoren des menschlichen Skeletts und dessen Körperorientierung verwendet. Ein Knochen der zwischen zwei Gelenkpunkten liegt wird als 3D-Vektor im Raum interpretiert. Die Summe der Winkeldifferenzen zwischen allen korrespondierenden Knochenvektoren der beiden Bewegungsreihen dient als Distanz-Metrik. Die Ergebnisse zeigen, dass DTW effektiv eingesetzt werden kann, um automatisiert Bewegungsabläufe mit Vorlagen zu vergleichen und diese somit zu bewerten. Im Kontext der Interaktion mit Wandbildschirmen werden andere Eingabewerte interessant sein als hier. Deshalb müssen geeignete Werte für unsere Szenarien elaboriert werden. Mohammadzade et al. (2021) nutzen TSD-Informationen für eine Überwachung von alltäglichen menschlichen Aktivitäten. Jede Aktion wird durch eine mehrdimensionale Time-Series repräsentiert. Dabei entspricht eine Dimension jeweils der Postion eines Skelettpunkts im Verlauf der Zeit. Im vorliegenden Ansatz wird jeder Bewegungsklasse genau eine Referenzbewegung zugeordnet. Die Daten werden entsprechend mit diesen Referenzen verglichen. Dadurch sind viel weniger Vergleiche nötig als bei anderen DTW-Methoden, wodurch die Rechenkomplexität abnimmt. Allerdings sind dafür vordefinierte Klassen mit teils künstlich erzeugten Beispielbewegungen nötig. Wesentliches Ziel dieser Bachelorarbeit ist es allerdings dazu beizutragen neue Informationen über mögliche Interaktionsverhalten mit interaktiven Wandbildschirmen zu gewinnen. Eine Definition im Voraus ist für dieses Szenario nicht zielführend. Hautamaki et al. (2008) vergleichen experimentell einige Time-Series Clustering-Verfahren mit DTW als Distanzmetrik. Hierarchisches Clustering wird hier agglomerativ implementiert. Das bedeutet, dass zunächst alle Records als eigenes Cluster initialisiert werden. Durch iterierte Anwendung von Merge-Operationen werden schrittweise zwei Cluster zusammengeführt, die eine geringe Distanz in den TSD aufweisen. Hautamaki et al. (2008) prüfen die Zuverlässigkeit anhand mehrerer Experimente. Unter anderem mit Sprachdaten, die ebenfalls zu den TSD gehören. Die Evaluation zeigt, dass hierarchisches Clustering in Kombination mit DTW erfolgreich eingesetzt werden kann.

3.5 Einsatz im Kontext von Kinect-Bewegungsdaten

Wie bereits erwähnt handelt es sich bei den vorliegenden Daten um TSD. Warren Liao (2005) und Aghabozorgi et al. (2015) weisen darauf hin, dass besonders die verwendete Distanz-/ Ähnlichkeitsmetrik entscheidend ist, um diesen Datentyp sinnvoll in Kategorien einteilen zu können. Hierarchisches Clustering in Kombination mit DTW scheint ein vielversprechender Ansatz zu sein. Da die Daten in Textdateien vorliegen ist eine Komponente im System nötig, die diese Informationen ausliest und in geeigneten Objekten der genutzten Programmiersprache abspeichert. Besonders von Bedeutung ist zudem die Wahl geeigneter Attribute. Die Daten enthalten viele Werte die mehr oder weniger interessant für die Kategorisierung sein können. Die Laufwege oder Engagement-Werte sind beispielsweise relevanter als die Tatsache, ob eine Person Brillenträger ist oder nicht. Idealerweise sollte die Wahl der genutzten Attribute vom Nutzer der Anwendung definiert werden können. So kann das Tool effektiv genutzt werden, um nach wiederkehrenden Abläufen in den Daten zu suchen. Des weiteren stellt sich die Frage, ab wann keine weitere Clusterbildung mehr stattfinden soll. Dazu muss ein geeigneter Threshold definiert werden. Zudem ist die große Menge der Daten zu erwähnen. Aghabozorgi et al. (2015) weist darauf hin, dass DTW weniger performant ist als herkömmliche Metriken wie die euklidische Distanz. Gegebenenfalls müssen die Daten vor der Nutzung geeignet gefilter werden. Diese Filterung soll allerdings nicht Bestandteil dieser Bachelorarbeit sein.

4 Konzeption

Vor der eigentlichen Implementierung erfolgt nun eine Konzeption. Damit wird elaboriert, welche Anforderungen an die Software gestellt werden und wie die Implementierung erfolgen soll. Etwaige Ungereimtheiten und Probleme können so frühzeitig erkannt werden.

4.1 Anforderungsanalyse

Es soll ein System implementiert werden, welches Kinect-Bewegungsdaten mithilfe von hierarchischem Clustering und DTW als Distanzmetrik gruppiert. Dieses Tool kann im Rahmen des HoPE-Projekts genutzt werden, um große Datensätze auf wiederkehrende Bewegungsabläufe bei der Interaktion mit Wandbildschirmen zu untersuchen. Es gibt sogenannte *funktionale Anforderungen* und *nicht-funktionale Anforderungen*, die das System erfüllen muss. Letztere beinhalten beispielsweise Anforderungen zu Benutzbarkeit, Effizienz oder Portierbarkeit. Funktionale Anforderungen beschreiben hingegen die primären Features des Systems.

4.1.1 Nicht-funktionale Anforderungen

Bei der Wahl der Programmiersprache und verwendeten Frameworks existieren keine speziellen Vorgaben. Das Tool sollte allerdings ohne großen Installationsaufwand nutzbar sein und daher nach Möglichkeit eine stark verbreitete Programmiersprache nutzen. Bei der Implementierung fällt daher die Wahl auf Java in der Version 14. Dabei sollen auch keine externen Bibliotheken genutzt werden. An die Effizienz gibt es ebenfalls keine besonderen Anforderungen, da die Auswertung der Daten nur einmalig zu einem späteren Zeitpunkt erfolgt. Eine Echtzeitauswertung ist nicht geplant. Da zu lange Wartezeiten, aber die Nutzbarkeit des Tools einschränken, sollen berechnete Kosten in einer geeigneten Datenstruktur zwischengespeichert werden. So können im darauffolgenden Iterationsschritt viele Berechnungsschritte eingespart werden. Das Tool sollte durch wenige Anpassungen auch auf Datensätze anderer Tiefenkameras angewandt werden können. Um dieses Ziel zu erreichen, wird die Software von

Beginn an möglichst generisch entwickelt. Zu allen Bestandteilen des Tools existiert daher ein Interface. Bei Bedarf können so einfach neue Implementierungen, die den Anforderungen anderer Datensätze entsprechen, ergänzt werden.

4.1.2 Funktionale Anforderungen

Die Software kümmert sich um den Input. Ein Datensatz kann aus Textdateien eingelesen und in passenden Datenobjekten abgespeichert werden. Das Tool soll ein Clustering mittels hierarchischem Clustering wie in Abschnitt 3.1 beschrieben anbieten. Dabei ist der Threshold für das Mergen von zwei Clustern vom Nutzer konfigurierbar. Als Distanzmetrik wird DTW (Abschnitt 3.2) genutzt. Dabei ist konfigurierbar, welche Attribute des Datensatzes für den Vergleich genutzt werden. Zur Berechnung der Distanz zwischen zwei Punkten wird die euklidische Distanz verwendet. Allerdings können auch hier einfach weitere Funktionen ergänzt werden. Per Startparameter kann entschieden werden, welche dieser Funktionen für die Berechnung verwendet wird. Zudem besitzt das Output-Funktionalitäten. Die gefundenen Cluster werden in einer Textdatei aufgelistet. Jedes Cluster erhält einen eindeutigen Namen und eine Liste aller Records die zusammengeführt wurden. Für ein erleichtertes Verständnis der Cluster wird auch eine primitive Visualiserung der Cluster angeboten. Hier werden die Laufpfade aus der Vogelperspektive dargestellt. Falls sich mehrere Records im Cluster befinden wird der Mittelwert der Attribute visualisiert. Die starke Konfigurierbarkeit wird durch eine *config-file* erreicht, die der Nutzer beim Start der Anwendung übergeben kann. Sie enthält folgende Parameter:

Attributname	Beschreibung
inputPath	Pfad zum Datensatz.
outputPath	Pfad zum Ablegen der Output-Dateien.
separator	Trennzeichen zwischen den Attributen.
datasetType	Art des vorliegenden Datensatzes.
threshold	Grenzwert für das Zusammenführen von Clustern.
attributes	Vorhandene Attribute.
usedAttributes	Attribute, die zum Vergleich genutzt werden.
distanceFunction	Name der genutzten Distanzfunktion.
flipVisualization	Wahrheitswert, ob die Visualisierung gespiegelt wird.
bodyIdParamName	Bezeichnung des Körperidentifikationsparameters.

4.2 Programmablauf

Im Folgenden wird der Programmablauf beschrieben.

1. Der Nutzer startet den *Clustering-Processor* und gibt als Startparameter den Pfad der *config-Datei* an.

2. Der *Processor* nutzt den *Config-Reader*, um die Werte der *Konfigurationdatei* auszulesen und speichert diese ab.

3. Der *Processor* nutzt den *DataReader*, um den Datensatz einzulesen und legt die Informationen in geeigneten Objekten ab. Beim Kinect-Datensatz sind das die Implementierungen *RecordImpl* und *FrameImpl* der entsprechenden Interfaces.

4. Der *Processor* startet das Clustering durch den Aufruf der *cluster-Methode* der *HierarchicalClustering-Klasse*. Zu beginn entspricht jeder Record einem eigenen Cluster. Die *recordToCluster-Methode* ermöglicht eine Transformation.

5. Das Clustering nutzt die *DTW-Implementierung*, um paarweise die Kosten zweier Cluster zu berechnen. Diese Kosten werden in einer Map abgespeichert, um wiederholte Berechnungen in darauffolgenden Schritten zu vermeiden. Als Key der jeweiligen Map-Einträge dient ein Objekt der Klasse ClusterKey.

6. Die beiden Cluster mit den geringsten Kosten werden zusammengeführt. Dazu wird ein neues *Cluster-Objekt* angelegt, dass durch Kombination der beiden Cluster entsteht. Konkret wird eines der Cluster in das andere integriert. Beim integrierten wird ein boolescher Wert auf *false* gesetzt, damit das Cluster in späteren Berechnungsschritten nicht mehr beachtet wird. Dabei wird jeweils das Mittel der Attributwerte berechnet und abgelegt. Zudem werden die Cluster-Bestandteile abgespeichert. Bei allen Komponenten wird ebenfalls das *consider-flag* auf *false* gesetzt, damit sie bei zukünftigen Vergleichen nicht mehr berücksichtigt werden.

7. Schritte 5 und 6 werden so lange wiederholt, bis die geringsten Kosten den vom Nutzer definierten Threshold übersteigen.

8. Der *Processor* nutzt den *Cluster-Writer*, um alle Cluster der *clusters-Liste* in eine Ausgabedatei zu schreiben.

9. Die gefundenen Cluster werden visualisiert und ebenfalls im Output-Verzeichnis abgelegt.

10. Die Anwendung terminiert.

4.3 Teilsysteme

Abbildung 4.1 kann das Paketdiagramm des Tools entnommen werden. Das Paket *model*

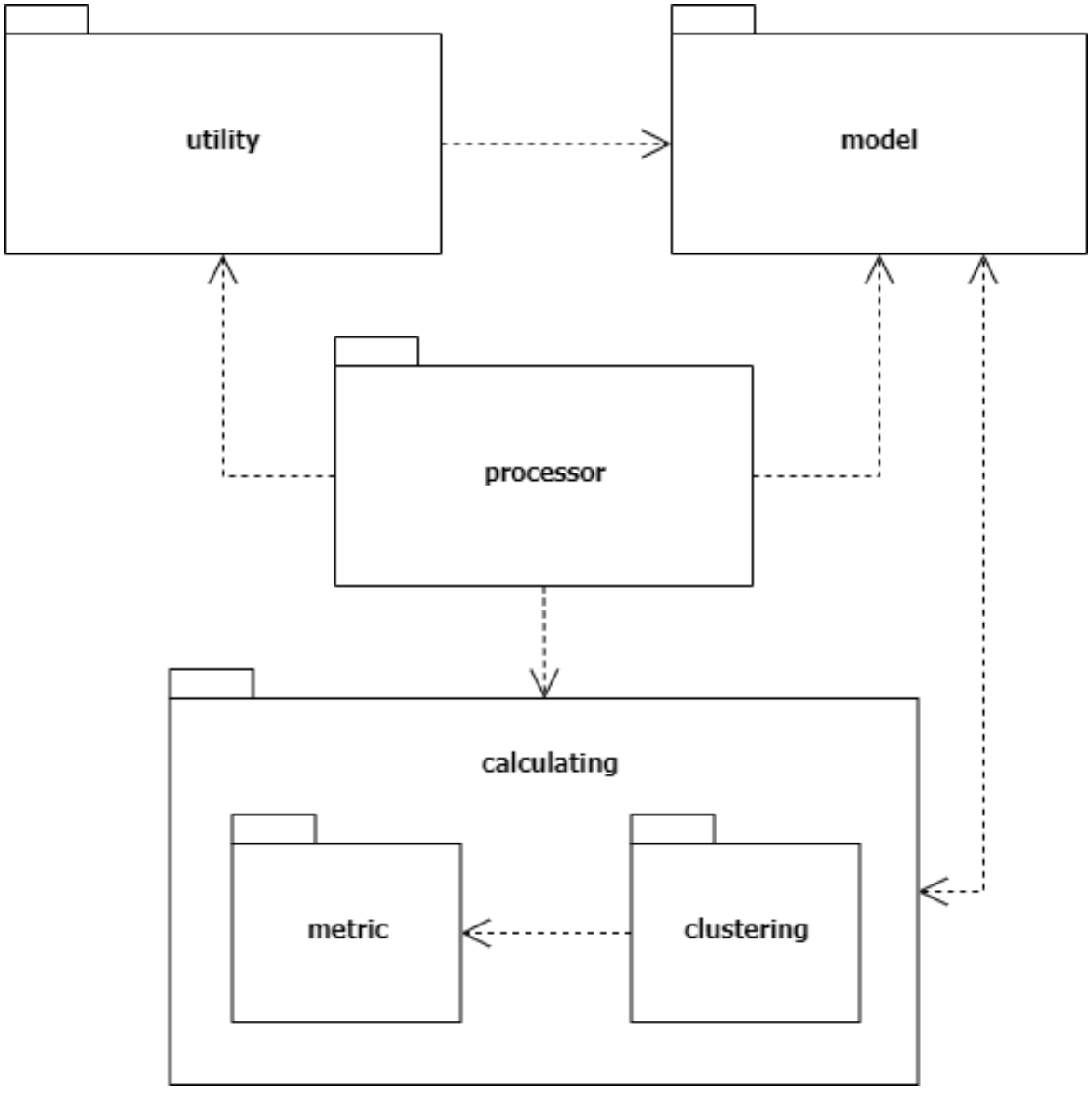


Abbildung 4.1: Paketdiagramm des Tools.

enthält die Interfaces für Cluster, Records und Frames. Außerdem jeweils geeignete Implementierungen dieser Interfaces für den Kinect-Datensatz. Bei Bedarf können hier bei späterer Verwendung der Software weitere Implementierungen für andere Datensätze ergänzt werden. Im *utility*-Paket befinden sich Interfaces für Reader, Writer und Visualizer. Auch hier sind geeignete Implementierungen Bereitsgestellt. Konkret ein Reader für die Config-Datei, ein Reader für die Kinect-Daten, ein Writer für die Ergebnisse und ein primitiver Visualizer, der die Laufwege der gefundenen Cluster visualisiert. Außerdem befindet sich hier die Klasse ClusterKey. Sie implementiert das Comparable-Interface und sorgt dafür, dass zwei Cluster zusammen als Key einer Map verwendet werden können. Im Paket *calculating* befindet sich alles was für das Clustern der Daten benötigt wird. Es ist weiter untergliedert in *clustering* und *metric*. Letzteres liefert die Vergleichsmetrik für unser Clusterverfahren. Wieder liegen geeignete Interfaces vor, sodass die Anwendung bei Bedarf um weitere Metriken oder Clustering-Ansätze erweitert werden kann. Bereitsgestellt werden initial eine Implementierung mit hierarchischem Clustering und DTW als Metrik. Das Paket *processor* ist von zentraler Bedeutung. Hier wird die Funktionalität des Tools zusammengeführt. Zunächst werden die benötigten Reader Instanzen initialisiert. Mit ihnen werden die Daten eingelesen. Anschließend wird das Clustering gestartet und die Ergebnisse mithilfe des Writers geeignet in der Ausgabedatei gespeichert. Zuletzt erfolgt die Visualisierung mithilfe des Visualizers.

5 Implementierung

Wie bereits erwähnt erfolgte die Implementierung in Java. Damit für die Nutzung keine weiteren Installationen nötig sind, wurden nur die Standardbibliotheken zur Umsetzung genutzt. Der gesamte Source Code kann unter github/lzfs/tsprocessor abgerufen werden. In diesem Kapitel wird vertiefend auf die Struktur des Tools eingegangen und wichtige Aspekte der Implementierung betrachtet.

5.1 Aufbau

Der detailierte Aufbau des Tools kann Abbildung 5.1 entnommen werden. Hier wurde aus Gründen der Übersichtlichkeit auf Konstruktoren, sowie Getter- und Setter-Methoden verzichtet. Die Architektur wird im Folgenden detailiert beschrieben.

5.2 Codebeschreibung

```
1    if (datasetType.equals("kinect")) {
2            DataReader dataReader
3                = new DataReader(
4                    "D:",
5                    separator,
6                    attributes);
7            data = dataReader.read(inputPath);
8
9            HierarchicalClustering clustering
10               = new HierarchicalClustering(
11                   data,
12                   threshold,
13                   attributes,
14                   usedAttributes,
15                   distanceFunction);
16           clusterImpls = clustering.cluster();
```

```
17
18          ClusterWriter writer = new ClusterWriter();
19          writer.write(
20              outputPath,
21              clusterImpls);
22
23          VisualizerImpl visualizerImpl
24              = new VisualizerImpl(
25                  flipVisualization,
26                  bodyIdParamName);
27          for (ClusterImpl clusterImpl : clusterImpls) {
28              visualizerImpl.visualize(
29                  clusterImpl.getId(),
30                  outputPath,
31                  clusterImpl.getMedianFrames());
32          }
33      }
```

Listing 5.1: Processor-Workflow.

```
1   public double calculateCost(
2       List<FrameImpl> frames1,
3       List<FrameImpl> frames2) {
4       // add the calculated cost of each attribute to it
5       double cost = 0;
6       // calculate the cost for all attributes
7       for (String attribute : this.usedAttributes) {
8           // build the cost matrix for this attribute
9           double[][] dtwMatrix = buildCostMatrix(
10              frames1,
11              frames2,
12              attribute);
13          // calculate and add the cost of this attribute
14          cost += calculatePathCost(dtwMatrix);
15      }
16      return cost / this.usedAttributes.size();
17  }
```

Listing 5.2: Berechnung der Kosten.

```
1   private double[][] buildCostMatrix(
2       List<FrameImpl> frames1,
3       List<FrameImpl> frames2,
```

```
4         String attribute) {
5         double[] s = new double[frames1.size()];
6         double[] t = new double[frames2.size()];
7
8         int counter = 0;
9         for (FrameImpl frame : frames1) {
10            s[counter] = Double.parseDouble(frame.getValue(attribute));
11            counter += 1;
12        }
13        counter = 0;
14        for (FrameImpl frame : frames2) {
15            t[counter] = Double.parseDouble(frame.getValue(attribute));
16            counter += 1;
17        }
18
19        int n = s.length;
20        int m = t.length;
21        // filled with zeros by default
22        double[][] dtwMatrix = new double[n + 1][m + 1];
23
24        for (int i = 0; i < dtwMatrix.length; i++) {
25            for (int j = 0; j < dtwMatrix[0].length; j++) {
26                // filled with infinity by definition of dtw
27                dtwMatrix[i][j] = Double.POSITIVE_INFINITY;
28            }
29        }
30        // filled with 0 by definition of dtw
31        dtwMatrix[0][0] = 0;
32
33        for (int i = 1; i < dtwMatrix.length; i++) {
34            for (int j = 1; j < dtwMatrix[0].length; j++) {
35                // this is the default distance function
36                double cost = Math.abs(s[i - 1] - t[j - 1]);
37
38                double minTmp = Math.min(
39                    dtwMatrix[i - 1][j],
40                    dtwMatrix[i][j - 1]);
41                double lastMin = Math.min(
42                    minTmp,
43                     dtwMatrix[i - 1][j - 1]);
44                dtwMatrix[i][j] = cost + lastMin;
```

```
45              }
46          }
47          return dtwMatrix;
48      }
```

Listing 5.3: Kostenmatrix.

```
1       public double calculatePathCost(double[][] dtwMatrix) {
2           int n = dtwMatrix.length - 1;
3           int m = dtwMatrix[n - 1].length - 1;
4           int pathCounter = 1;
5           double cost = dtwMatrix[n][m];
6           while (n != 0 && m != 0) {
7               // dynamic time warping algorithm
8               double minTmp = Math.min(dtwMatrix[n - 1][m], dtwMatrix[n][m - 1]);
9               double lastMin = Math.min(minTmp, dtwMatrix[n - 1][m - 1]);
10              cost += lastMin;
11              if (lastMin == dtwMatrix[n - 1][m - 1]) {
12                  n = n - 1;
13                  m = m - 1;
14              }
15              else if (lastMin == dtwMatrix[n - 1][m]) {
16                  n = n - 1;
17              }
18              else if (lastMin == dtwMatrix[n][m - 1]) {
19                  m = m - 1;
20              }
21              pathCounter += 1;
22          }
23          return cost / pathCounter;
24      }
```

Listing 5.4: Warping Path berechnen.

```
1       public List<ClusterImpl> cluster() {
2           List<ClusterImpl> result = new ArrayList<>();
3           // initialize each record as a cluster and add it to the clusters list
4           for (RecordImpl record : this.initialRecords) {
5               this.clusterImpls.add(this.recordToCluster(record));
6           }
7           double currentMinimumCost;
8           double cost;
9           int mergeCandidate1;
```

```
10          int mergeCandidate2;
11
12          do {
13              // reset for next loop iteration
14              currentMinimumCost = this.dtw.calculateCost(
15                  this.clusterImpls.get(0).getMedianFrames(),
16                  this.clusterImpls.get(1).getMedianFrames());
17              cost = this.dtw.calculateCost(
18                  this.clusterImpls.get(0).getMedianFrames(),
19                  this.clusterImpls.get(1).getMedianFrames());
20              mergeCandidate1 = 0;
21              mergeCandidate2 = 1;
22              for (ClusterImpl clusterImpl1 : this.clusterImpls) {
23                  for (ClusterImpl clusterImpl2 : this.clusterImpls) {
24                      if (
25                          clusterImpl1 != clusterImpl2
26                          && clusterImpl1.isConsider()
27                          && clusterImpl2.isConsider()) {
28                          if (this.calculatedCost.containsKey(
29                              new ClusterKey(
30                                  clusterImpl1,
31                                  clusterImpl2))) {
32                              cost = this.calculatedCost.get(
33                                  new ClusterKey(
34                                      clusterImpl1,
35                                      clusterImpl2));
36                          }
37                          else {
38                              cost = this.dtw.calculateCost(
39                                  clusterImpl1.getMedianFrames(),
40                                  clusterImpl2.getMedianFrames());
41                              this.calculatedCost.put(
42                                  new ClusterKey(
43                                      clusterImpl1,
44                                      clusterImpl2),
45                                  cost);
46                          }
47                          if (cost <= currentMinimumCost) {
48                              /* if a cost smaller than the current minimum cost
49                              is found this will be the new cost */
50                              currentMinimumCost = cost;
```

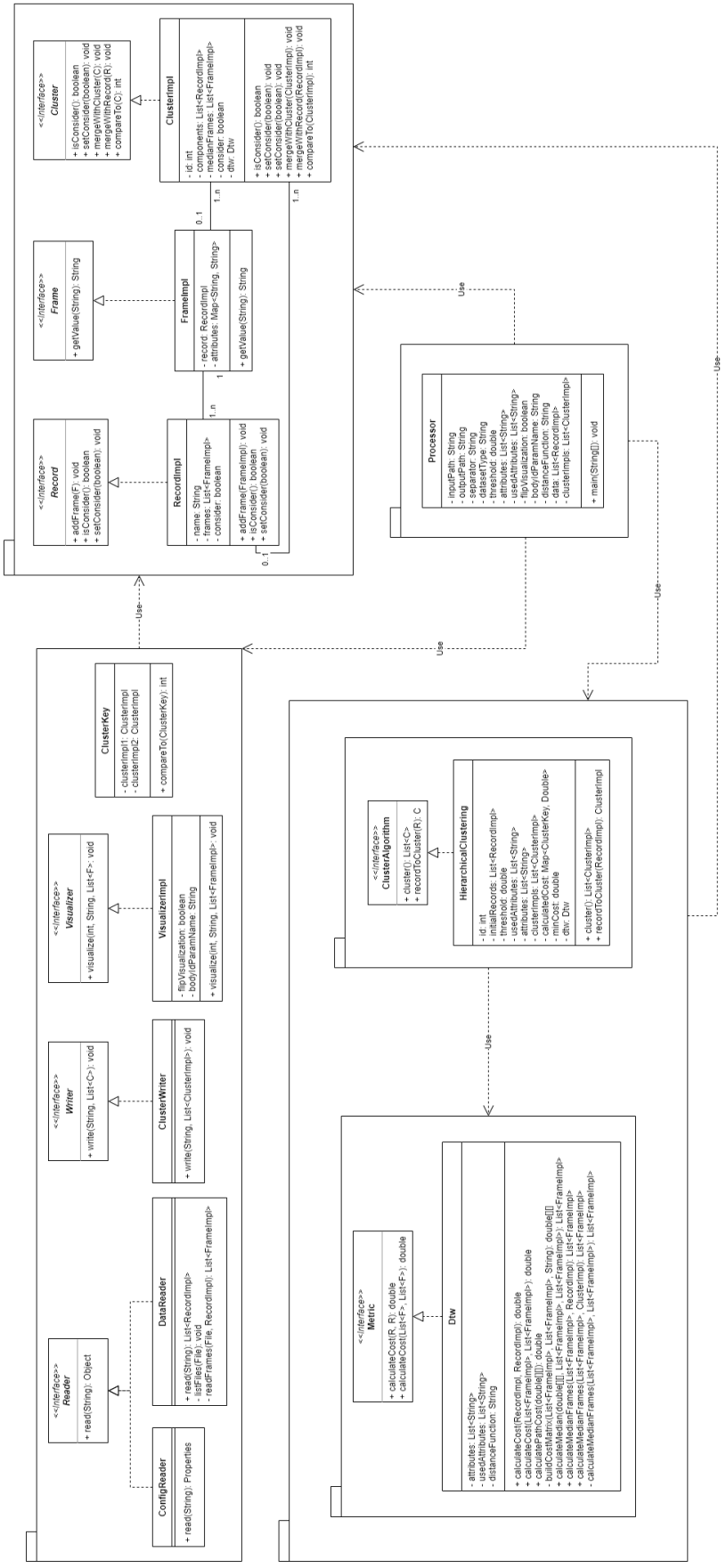
```
                            mergeCandidate1 =
                                this.clusterImpls.indexOf(clusterImpl1);
                            mergeCandidate2 =
                                this.clusterImpls.indexOf(clusterImpl2);
                        }
                    }
                }
            }
            // if the minimum cost is still below the threshold we can continue
            if (
                currentMinimumCost < threshold
                && currentMinimumCost > this.minConst) {
                /* Combining mergeCandidate1 and mergeCandidate2
                has the lowest found cost.
                These two should therefore be merged together.
                Merge cluster2 into cluster1 and update the cluster list.
                Consider of cluster2 is set to false. */
                this.clusterImpls.get(mergeCandidate1).mergeWithCluster(
                    this.clusterImpls.get(mergeCandidate2));

                // create a copy of the map to avoid an exception
                Map<ClusterKey, Double> calculatedCostCopy = new HashMap<>();
                calculatedCostCopy.putAll(calculatedCost);

                /* remove all calculated costs from the map that contain
                cluster1 because it changed and therefore all cost values
                with this cluster have to be calculated again */
                for (Map.Entry<ClusterKey, Double> entry : calculatedCostCopy.
entrySet()) {
                    /* ignore mergeCandidate2 because the consider
                    value of cluster2 is set to false */
                    if (entry.getKey().getCluster1().getId() == mergeCandidate1)
{
                        for (ClusterImpl clusterImpl : clusterImpls) {
                            calculatedCost.remove(
                                new ClusterKey(
                                    this.clusterImpls.get(mergeCandidate1),
                                    clusterImpl));
                        }
                    }
                }
```

```
90              }
91          } while (
92              currentMinimumCost < threshold
93              && currentMinimumCost > this.minConst);
94          /* clean up the list of found clusters by removing all clusters
95          that shouldn't be considered anymore */
96          for (ClusterImpl clusterImpl : this.clusterImpls) {
97              if (clusterImpl.isConsider()) {
98                  result.add(clusterImpl);
99              }
100         }
101         return result;
102     }
```

Listing 5.5: Hierarchisches Clustering.

5.3 Abweichungen zur Konzeption

5.4 Verbesserung der Performanz



29

Abbildung 5.1: Klassendiagramm des Tools.

6 Evaluation

6.1 Daten mit einer Person

6.2 Daten mit mehreren Personen

7 Fazit

Abkürzungsverzeichnis

HCI	Mensch-Computer-Interaktion
RGB	Rot-Grün-Blau
ToF	Time-of-Flight
DTW	Dynamic Time Warping
TSD	Time-Series Data

Abbildungsverzeichnis

Literatur

Aghabozorgi, S., Seyed Shirkhorshidi, A., & Ying Wah, T. (2015). Time-series clustering – A decade review. *Information Systems*, 16–38. https://doi.org/10.1016/j.is.2015.04.007

Ali, M., Alqahtani, A., Jones, M. W., & Xie, X. (2019). Clustering and Classification for Time Series Data in Visual Analytics: A Survey. *IEEE Access*, 181314–181338. https://doi.org/10.1109/ACCESS.2019.2958551

Hautamaki, V., Nykanen, P., & Franti, P. (2008). Time-series clustering by approximate prototypes. *2008 19th International Conference on Pattern Recognition*, 1–4. https://doi.org/10.1109/ICPR.2008.4761105

Li, L., et al. (2014). Time-of-flight camera–an introduction [SLOA190B]. *Technical white paper*. Verfügbar 10. März 2022 unter https://www.ti.com/lit/wp/sloa190b/sloa190b.pdf

Mai, C., & Hußmann, H. (2018). The Audience Funnel for Head-Mounted Displays in Public Environments. *2018 IEEE 4th Workshop on Everyday Virtual Reality (WEVR)*, 5.

Mankoff, J., Dey, A. K., Hsieh, G., Kientz, J., Lederer, S., & Ames, M. (2003). Heuristic Evaluation of Ambient Displays. *NEW HORIZONS*, 169–176. https://doi.org/10.1145/642611.642642

Marin, G., Agresti, G., Minto, L., & Zanuttigh, P. (2019). A multi-camera dataset for depth estimation in an indoor scenario. *Data in Brief*, 104619. https://doi.org/10.1016/j.dib.2019.104619

Microsoft. (2014). Human Interface Guidelines v2.0 [Windows-Developer-Center Microsoft Corporation]. Verfügbar 1. März 2022 unter https://download.microsoft.com/download/6/7/6/676611B4-1982-47A4-A42E-4CF84E1095A8/KinectHIG.2.0.pdf

Mohammadzade, H., Hosseini, S., Rezaei-Dastjerdehei, M. R., & Tabejamaat, M. (2021). Dynamic Time Warping-Based Features With Class-Specific Joint Importance Maps for Action Recognition Using Kinect Depth Sensor. *IEEE Sensors Journal*, 9300–9313. https://doi.org/10.1109/JSEN.2021.3051497

Plischke, T. (2022). *Noch nicht abgeschlossen* (Masterarbeit). Universität der Bundeswehr München.

Temiz, J. (2022). *Konzeption und Implementierung eines Datenanalyse-Werkzeugs für Body-Tracking-Kameras* (Bachelorarbeit). Universität der Bundeswehr München.

Tölgyessy, M., Dekan, M., Chovanec, L., & Hubinský, P. (2021). Evaluation of the Azure Kinect and Its Comparison to Kinect V1 and Kinect V2. *Sensors*, 413. https://doi.org/10.3390/s21020413

UniBw. (2021). Honeypot-Effekt an interaktiven Ambient Displays (HoPE) — Inf2. Verfügbar 28. Februar 2022 unter https://www.unibw.de/inf2/forschung/projekte/hope

Wahyuni, S., Rismayani, Intan, I., Ahyuna, A., & Suryani, S. (2021). Motion Recognition System with Dynamic Time Warping Method using Kinect Camera Sensor. *2021 3rd International Conference on Cybernetics and Intelligent System (ICORIS)*, 1–4. https://doi.org/10.1109/ICORIS52787.2021.9649520

Warren Liao, T. (2005). Clustering of time series data—a survey. *Pattern Recognition*, 1857–1874. https://doi.org/10.1016/j.patcog.2005.01.025

Wouters, N., Downs, J., Harrop, M., Cox, T., Oliveira, E., Webber, S., Vetere, F., & Vande Moere, A. (2016). Uncovering the Honeypot Effect: How Audiences Engage with Public Interactive Systems. *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, 5–16. https://doi.org/10.1145/2901790.2901796

Yu, X., & Xiong, S. (2019). A Dynamic Time Warping Based Algorithm to Evaluate Kinect-Enabled Home-Based Physical Rehabilitation Exercises for Older People. *Sensors*, 2882. https://doi.org/10.3390/s19132882

Hiermit versichere ich, die vorliegende Arbeit selbständig und ohne fremde Hilfe verfasst, die Zitate ordnungsgemäß gekennzeichnet und keine anderen, als die im Literatur/Schriftenverzeichnis angegebenen Quellen und Hilfsmittel benutzt zu haben.

Ferner habe ich vom Merkblatt über die Verwendung von studentischen Abschlussarbeiten Kenntnis genommen und räume das einfache Nutzungsrecht an meiner Bachelorarbeit der Universität der Bundeswehr München ein.

München, den 31. Mai 2022

.....

(Unterschrift)