Effiziente Algorithmen zur inhaltsbasierten Suche in Bewegungsdaten

Diplomarbeit

Andreas Baak

Rheinische Friedrich-Wilhelms-Universität Bonn
Institut für Informatik III

31.03.2008

Inhaltsverzeichnis

Kapitel 1

Einleitung

Aufgrund der Komplexität menschlicher Bewegungen ist das manuelle Erstellen von Bewegungsabläufen eine langwierige und schwierige Aufgabe, aber mit Hilfe von digitalen Repräsentationen natürlicher Bewegungen können diese Abläufe vereinfacht werden. Die Technik des *Motion-Capturing* kann menschliche Bewegungen in maschinenlesbare Daten transformieren, die dann auf virtuelle Charaktere übertragen werden können. So können mit verhältnismäßig geringem Animationsaufwand realistische Bewegungen virtueller Charaktere erzeugt werden.

Ein Nachteil des Motion-Capturing ist der hohe Preis der Motion-Capture-Systeme (Hochgeschwindigkeitskameras, Software) und der zeitliche Aufwand für die Aufnahme und die Nachverarbeitung der Daten. Es ist daher sinnvoll, bereits existierende Datensätze für neue Anwendungen wiederzuverwenden. Im Hinblick auf eine *Wiederverwendung von Motion-Capture-Daten* werden Methoden benötigt, um in den Daten gezielt nach bestimmten von Anwendungen verwendeten Bewegungen suchen zu können. Für den Menschen ist die Klassifikation von Bewegungen eine leichte Aufgabe. Ohne Schwierigkeiten können wir zwischen Bewegungsklassen wie Gehen, Rennen und Hüpfen unterscheiden. Täglich werden wir beim Spazieren, Autofahren oder Einkaufen mit solchen Bewegungen konfrontiert und können diese spielend auseinander halten. Auch in einer Menge von aufgenommenen Motion-Capture-Daten ist es für den Menschen ein Leichtes, einzelne Bewegungen von anderen zu trennen und ihnen eine Bewegungsklasse zuzuordnen. Unter Zuhilfenahme manuell generierter Annotationen der Daten, also textuellen Beschreibungen der enthaltenen Bewegungen, sind Computer mit bekannten text-basierten Suchalgorithmen in der Lage, stichwortbasiert diese Daten zu durchsuchen. Der entscheidende Nachteil ist dabei, dass das manuelle Erstellen der Annotationen sehr zeitaufwändig und für immer größer werdende Sammlungen von Motion-Capture-Daten nicht durchführbar ist.

Die Suche in Bewegungsdaten und deren Klassifikation ohne Zuhilfenahme von Annotationen ist eine anspruchsvolle Aufgabe. Bei der so genannten *inhaltsbasierten Suche* wird nur auf die Motion-Capture-Daten selbst und nicht auf manuell generierte Metadaten zurückgegriffen. Eine große Herausforderung bei der Suche und dem Vergleich von Bewegungen stellt dabei die Entwicklung von geeigneten Ähnlichkeitsmaßen dar. Bewegungen können in unterschiedlicher Geschwindigkeit (schnelles Gehen, langsames Gehen) und mit verschiedenen Details (Arme schwingen beim Gehen stark oder weniger stark) ausgeführt werden. Trotzdem sind solche Bewegungen semantisch ähnlich. Andererseits können zwei Bewegungen, die semantisch eine unterschiedliche Handlung ausdrücken, sich nur in subtilen Details unterscheiden. Zum Beispiel unterscheidet sich eine Bewegung, die einen Gegenstand in einem Regal ablegt,

nur kaum von einer Bewegung, die einen Gegenstand aus einem Regal greift. Grundlegend für die vorliegende Arbeit ist ein anschaulicher Ähnlichkeitsbegriff, der mit Hilfe von relationalen Merkmalen und Motion Templates motiviert wird. Motion Templates repräsentieren die Gemeinsamkeiten in einer Klasse von Bewegungen in Bezug auf die relationalen Merkmale.

Um Toleranz gegenüber zeitlichen und räumlichen Variationen von Bewegungen erlauben zu können, nehmen Suchalgorithmen hohe Laufzeiten in Kauf. Die Verbesserung der Laufzeit dieser Algorithmen ist ein aktuelles und wichtiges Forschungsgebiet, damit die inhaltsbasierte Suche in der Praxis ohne lange Wartezeiten genutzt werden kann.

Beitrag und Gliederung dieser Arbeit

Diese Arbeit baut auf den in [MR06] und [MRC05] publizierten Verfahren zur inhaltsbasierten Suche von Bewegungsdaten auf. Dort wurde vorgestellt, wie mit Hilfe von robusten Merkmalen und einer gelernten Repräsentation einer Klasse von Bewegungen inhaltsbasierte Suche durchgeführt werden kann. Insbesondere wurden relationale Merkmale eingeführt, die ein hohes Maß an Invarianz unter räumlichen Deformationen aufweisen. Unter Benutzung dieser Merkmale können Repräsentationen von Bewegungsklassen (so genannte Motion Templates, MTs) generiert werden, mit deren Hilfe eine fehlertolerante Suche mit Dynamic Time Warping (DTW) möglich wird. Den guten Suchergebnissen dieses Verfahrens steht die hohe Laufzeitkomplexität des DTW-Algorithmus gegenüber.

In dieser Arbeit wird ein neuer Algorithmus vorgestellt, der mit Hilfe von Schlüsseleigenschaften (so genannten Keyframes) einer Bewegungsklasse effizient nach Repräsentanten der Klasse in einer großen Sammlung von Bewegungsdaten suchen kann. Dabei werden, ähnlich wie bei DTW, zeitliche Deformationen der Schlüsseleigenschaften erlaubt, aber trotzdem kurze Laufzeiten (wenige Sekunden bei Datenbankgrößen von mehreren Stunden) erreicht. Im Anschluss an diese effiziente Suche kann in einem Nachverarbeitungsschritt, wie in [MR06] vorgestellt, eine Rangordnung der gefundenen Treffer hergestellt werden. Da typischerweise die erwartete Treffermenge klein ist, bewegt sich die Laufzeit des DTW-Algorithmus bei der Nachverarbeitung in einem akzeptablen Rahmen von wenigen Sekunden.

Kapitel 2 fasst die Grundlagen über die Erstellung und Modellierung von Motion-Capture-Daten zusammen. Die Grundidee der in [MRC05] vorgestellten relationalen Merkmale wird in einer anschaulichen Darstellung anhand eines Beispiels wiedergegeben.

In Kapitel 3 werden die in [MR06] veröffentlichten Methoden zur inhaltsbasierten Suche mit Hilfe von Motion Templates zusammengefasst. Dabei wird kurz auf das Konzept des DTW-Algorithmus eingegangen. Im Anschluss daran wird das Lernverfahren einer neuen, in dieser Diplomarbeit entwickelten Form von MTs beschrieben. Eine Eigenschaft dieser MTs, die einen viel versprechenden Ansatz für die automatische Gewinnung von Schlüsseleigenschaften bietet, wird modelliert und bewiesen. Schließlich wird die Suche mit Hilfe von MTs beschrieben und es werden experimentelle Resultate vorgestellt.

Die beiden Kernkapitel 4 und 5 dieser Diplomarbeit beschäftigen sich mit dem neu entwickelten Algorithmus zur keyframebasierten Suche nach Schlüsseleigenschaften in einer Datenbank. Dabei werden zuerst theoretische und danach praktische Aspekte untersucht. Kapitel 4 erklärt die Funktionsweise des Algorithmus im Detail. Die Korrektheit des Algorithmus und die lineare Laufzeit in der Gesamtlänge der durch die Schlüsseleigenschaften beschriebenen invertierten Listen werden gezeigt. Auch ein Vergleich mit einem vorher existierenden Ansatz wird gegeben. Die zur Überprüfung der Praxistauglichkeit des Algorithmus durchgeführten Experimente und deren Auswertungen werden in Kapitel 5 beschrieben.

Kapitel 6 beschäftigt sich kurz mit einem ersten Ansatz zur automatischen Generierung von Schlüsseleigenschaften für Bewegungsklassen.

Die Arbeit wird durch Kapitel 7 mit einer Zusammenfassung und einem Ausblick abgeschlossen.

Danksagung

An dieser Stelle möchte ich mich herzlich bei allen bedanken, die mir das erfolgreiche Verfassen dieser Diplomarbeit ermöglicht haben. Ich bedanke mich bei der gesamten Arbeitsgruppe von Prof. Dr. Michael Clausen für die hervorragenden Rahmenbedingungen zur Erstellung dieser Arbeit. Meinem Betreuer Priv.Doz. Dr. Meinard Müller gilt dabei ein besonderer Dank. In einer intensiven Zeit seines Habilitationsverfahrens, Umzugs und einer neuen Forschungsstelle stand er stets für Diskussionen, produktive Vorschläge und konstruktive Kritik zur Verfügung. Die Treffen mit ihm waren immer Ereignisse, die ganz entscheidend zu dem schnellen Fortschreiten der Diplomarbeit beigetragen haben. Ich bedanke mich, dass ich in einem spannenden, interessanten und hochaktuellen Forschungsgebiet mitarbeiten durfte und die Chance bekommen habe, in der Wissenschaft weiterarbeiten zu können.

Ein großer Dank gilt den Freunden und Kommilitonen an der Uni Bonn, die den Studienalltag in allen Belangen bereichert haben. Insbesondere bedanke ich mich für das zuverlässige und kritische Korrekturlesen bei Jens Behley, Marcell Missura, Matthias Thiebes, Shahram Faridani, Thomas Helten und Tom Noll. Ich bedanke mich bei Björn Krüger für die gute Zusammenarbeit.

Ich bedanke mich bei meinen Eltern, dass Sie mich in allen meinen Unternehmungen – nicht nur beim Studium – mit ihrer ganzen Kraft unterstützt haben.

Bei der Hauptperson meines Lebens bedanke ich mich vor allen Dingen. Jana stand mir in einer anstrengenden Zeit mit ihrer ganzen Liebe und Kraft zur Seite. Durch die einzigartige Wärme, mit der sie es verstand, mir Energie, Gelassenheit und Ausdauer zu spenden, konnte ich in dieser intensiven Zeit Entspannung finden.

Kapitel 2

Grundlagen

Dieses Kapitel gibt einen Überblick über die Grundlagen, welche entscheidend für das Verständnis der Beiträge in den folgenden Kapiteln sind. Zuerst wird in Abschnitt 2.1 beschrieben, wie menschliche Bewegungen in ein vom Computer lesbares Format transformiert werden können. Abschnitt 2.2 zeigt, wie die Bewegungsdaten in ein Format umgewandelt werden können, das umfangreiche semantische Analysen erlaubt.

Der Vergleich verschiedener Motion-Capture-Datenströme ist ein zentraler Bestandteil der inhaltsbasierten Suche von Motion-Capture-Daten. Hierbei ist festzustellen, welche Ausschnitte der Datenbank zu einer vorgegebenen Anfrage ähnlich sind. Dabei ist stark anwendungsabhängig, was mit dem Begriff „ähnlich“ gemeint ist. In dieser Arbeit sollen Bewegungen als ähnlich angesehen werden, wenn sie derselben Bewegungsklasse angehören. Dies ist eine semantische Ähnlichkeit der Daten, die keineswegs mit numerischer Ähnlichkeit der Motion-Capture-Daten einhergeht. Abschnitt 2.3 zeigt, wie mit Hilfe von relationalen Merkmalen (veröffentlicht in [MRC05]) der Vergleich verschiedener Bewegungen auf einer semantischen Ebene, die robust unter räumlichen Deformationen ist, durchgeführt werden kann. Durch den in Abschnitt 2.4 vorgestellten Ansatz der adaptiven Segmentierung kann man dann auch zeitliche Deformationen in den Griff bekommen. Relationale Merkmale und adaptive Segmentierung blenden irrelevante Details aus Motion-Capture-Datenströmen aus und fokussieren auf gewünschte semantische Eigenschaften. Zusätzlich reduzieren die Techniken die Datenmenge eines Motion-Capture-Datenstromes erheblich, wodurch effiziente Berechnungen möglich werden.

2.1 Motion-Capture-Daten

Motion-Capture-Daten sind Aufnahmen von Bewegungen in einem Format, das von Computern gelesen und analysiert werden kann. In der Computeranimation werden diese Daten für Videospiele oder Filme genutzt, indem die Bewegungen von Schauspielern mit einem Motion-Capture-System aufgenommen, eventuell verändert und verarbeitet werden und dann auf einen animierten, virtuellen Charakter übertragen werden. Im Gegensatz zur vollständig manuellen Animation von Charakteren erspart diese Methode viel Handarbeit beim Erstellen der Bewegung. Weiterhin sehen die mit Hilfe von Motion-Capture-Daten erzeugten Bewegungen realistischer und menschlicher aus als manuell erzeugte Bewegungen. Gerade in Animationsfilmen ist es erwünscht, die Bewegung der virtuellen Charaktere möglichst menschlich aussehen zu lassen, weshalb hier die Verwendung von Motion-Capture-Daten eine wichtige Rolle spielt.

Es ist aus Kostengründen wünschenswert, die erstellten Motion-Capture-Daten wiederzuverwenden. Da Sammlungen von aufgenommen Bewegungen schnell unübersichtlich groß werden können, ist es erforderlich, logische Zusammenhänge in den Daten zu erkennen und die Sammlung nach bestimmten Bewegungen durchsuchbar und damit handhabbar zu machen. Hierbei spielen die Analyse, der Vergleich, die Klassifikation und das Retrieval auf Motion-Capture-Daten eine elementare Rolle. Es ist möglich, diese Daten manuell zu annotieren und ihnen Informationen hinsichtlich ihres Inhaltes zuzuordnen. Mit Hilfe dieser Annotationen kann leicht in diesen Daten mit bekannten Textretrieval-Methoden gesucht werden. Annotationen haben jedoch den Nachteil, dass die Erstellung für große Datensammlungen aufwendig ist. Es ist ebenfalls schwierig, eine universelle, für jede Anfrage ausreichende Annotation zu erstellen, da im Voraus nicht absehbar ist, welche Annotationskategorien für zukünftige Suchanfragen benötigt werden. Es können beispielsweise Annotationen erstellt werden, welche die Bewegungsklasse (wie z.B. Laufen, Gehen, Springen) enthalten. Wird nun nach einer Bewegung gesucht, die ausgelassenes oder fröhliches Laufen enthält, so helfen die erstellten Annotationen nur bedingt und müssen mit viel Aufwand erweitert werden.

Somit sind Methoden, die nicht auf manuell erstellten Annotationen beruhen, von großer Bedeutung. Solche Methoden werden in [DRME06], [MR06] und [MRC05] behandelt. In der vorliegenden Arbeit wird aufbauend auf den in [MR06] und [MRC05] vorgestellten Methoden ein neuer Algorithmus zum indexbasierten Retrieval von Motion-Capture-Daten vorgestellt.

Für die Experimente werden die Motion-Capture-Daten aus der HDM05-Datenbank (vorgestellt in [MRC$^+$07]) verwendet, die im Jahr 2005 an der Hochschule der Medien in Stuttgart (HDM) erstellt wurden. Bernhard Eberhardt (Hochschule der Medien, Stuttgart) hatte die Aufsicht über die Durchführung des Motion-Capturing. Die Bewegungssequenzen wurden systematisch erstellt: Fünf verschiedene Laien-Schauspieler führten jeweils mehrere Wiederholungen von verschiedenen, vorgeschriebenen Bewegungsabläufen durch. Die Bewegungsabläufe enthielten dabei verschiedene Bewegungsklassen wie Gehen, Laufen, Radschlag, Hampelmann, Springen, Greifen usw. Aus den Aufnahmen wurde eine Bewegungsdatenbank mit einer Gesamtlänge von 210 Minuten aufgebaut, die im Folgenden mit $\mathcal{D}_{210}$ bezeichnet wird. Aus dieser Datenbank wurden dann manuell kurze Bewegungssequenzen, die einer bestimmten Bewegungsklasse zugeordnet werden können, herausgeschnitten. Auf diese Weise entstanden nach Bewegungsklassen geordnete Sammlungen von kurzen Repräsentanten dieser Klasse. So gibt es beispielsweise 21 verschiedene Repräsentationen einer Radschlag-Bewegung, die mit der linken Hand beginnt und 31 verschiedene Gehbewegungen von zwei Schritten, ausgeführt mit dem linken Fuß beginnend. Die meisten der 130 verschiedenen Bewegungsklassen werden mit 10 bis 50 verschiedenen Ausführungen repräsentiert. Die dadurch entstandene, nach Bewegungsklassen sortierte Menge von Bewegungsdaten $\mathcal{D}^{MC}$ kann für die Überprüfung von Retrievaltechniken benutzt werden. Dadurch, dass genau bekannt ist, welche Bewegungsklasse an welcher Stelle der Datenbank auftritt, kann automatisiert identifiziert werden, welche der gefundenen Treffer keine der beabsichtigten Treffer sind (False Positives) und welche der beabsichtigen Treffer nicht gefunden werden (False Negatives). So lassen sich Ergebnisse von verschiedenen Retrievaltechniken objektiv beurteilen. Aus den 130 verschiedenen Bewegungsklassen wurden nun 57 Klassen, die durch eine ausreichende Anzahl von Bewegungen repräsentiert werden, ausgewählt und in der Datenbank D^{57} (Anhang A) zusammengefasst.

Die HDM05-Daten wurden mit dem optischen Motion-Capture-System Vicon MX aufgenommen. Hierzu tragen die Schauspieler einen speziellen Anzug, an dem 40-50 passive Marker

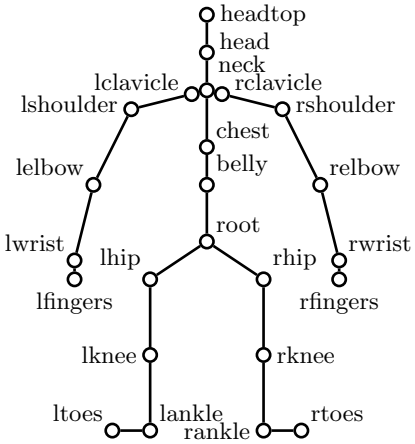


Abbildung 2.1. Hier ist ein Modell eines menschlichen Skelettes, wie es in dieser Arbeit verwendet wird, abgebildet. Das Modell ist eine kinematische Kette, also eine hierarchische Struktur aus starren Segmenten (den Knochen) und beweglichen Gelenken. Die Gelenke sind im Skelett markiert und mit einem Namen versehen, welcher die Position des Gelenkes im Körper beschreibt.

angebracht werden. Mit insgesamt 12 Kameras werden die Positionen der Marker mit einer Abtastrate von 120 Hz verfolgt. Das Motion-Capture-System kann dadurch, dass die Kameras aus verschiedenen Blickwinkeln dieselbe Szene beobachten, mit Hilfe von Triangulierung die 3D-Positionen der Marker berechnen. Obwohl zwei Kameras ausreichen, um die 3D-Positionsberechnung der Marker durchzuführen, ist es sinnvoll, mehr Kameras zu verwenden und damit Verdeckungserscheinungen zu minimieren und die Genauigkeit der Berechnung zu erhöhen. Trotz der 12 verwendeten Kameras kann es zu Verdeckungserscheinungen kommen. Daher werden Lücken in den Daten nach der Aufnahme mit einem semi-automatischen Verfahren geschlossen. Ein schwieriges Problem ist das Verfolgen der Marker, also das Identifizieren identischer Marker in zwei aufeinander folgenden Bildern. Durch Verdeckungen von Markern kann es zu Uneindeutigkeiten in der Zuordnung kommen. Auch diese Probleme werden semi-automatisch korrigiert. Das Ergebnis der Nachverarbeitung sind vollständige 3D-Trajektorien jedes Markers, welche im C3D-Dateiformat (siehe [C3D06]) gespeichert werden.

2.2 Modellierung von Motion-Capture-Daten

Im Folgenden werden die Motion-Capture-Daten mit Hilfe einer kinematischen Kette modelliert. Eine kinematische Kette ist ein hierarchisch angeordnetes System von Starrkörpern (den Knochen), welche ihre Form unter der Bewegung beibehalten. Die Starrkörper sind mit Gelenken verbunden, die unterschiedliche Freiheitsgrade haben können. So kann ein Kugelgelenk, wie es in der Hüfte oder Schulter vorhanden ist, mit drei Freiheitsgraden modelliert werden, während ein Ellenbogengelenk oder Kniegelenk als Scharniergelenk mit nur einem Freiheisgrad ausgestattet ist. Ein spezielles Gelenk wird als die Wurzel der kinematischen Kette ausgezeichnet. Die hierarchische Struktur ist ein Baum: Es dürfen keine Zyklen in der Kette auftreten. Dieser Anforderung genügt das menschliche Skelett: In Abbildung 2.1 ist eine mögliche kinematische Kette abgebildet, die das menschliche Skelett repräsentiert. Dabei kann das Gelenk „root“ als die Wurzel der Kette gewählt werden. Es bezeichne J die Menge der Gelenke. Jedes

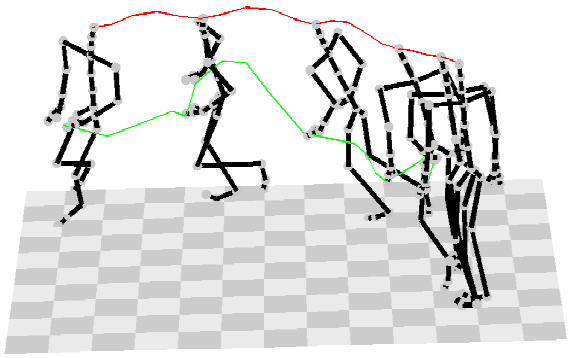


Abbildung 2.2. Hier sind zwei Trajektorien veranschaulicht. Die Bahnkurve der Skelettposition „headtop“ (rot) und die Trajektorie des Markers „lfingers“ (grün) sind mit einer durchgehenden Linie gezeichnet.

Gelenk wird mit einem intuitiven Namen bezeichnet, der sich auf seine Position im menschlichen Körper bezieht: „lankle“ für den linken Knöchel, „rwrist“ für das rechte Handgelenk und „lclavicle“ für das linke Schlüsselbein. Die Endeffektoren („{l|r}fingers“, „{l|r}toes“, „headtop“) werden der Einfachheit halber ebenfalls als Gelenke modelliert. Das „root“-Gelenk ist die Wurzel der Baumstruktur und befindet sich bei einer Grundstellung, wie sie in 2.1 abgebildet ist, ungefähr im Schwerpunkt des menschlichen Körpers. Ein Motion-Capture-Datenstrom ist eine Folge von Frames. Jeder einzelne Frame stellt dann die Information bereit, wo sich zum gegebenen Zeitpunkt jedes Gelenk im Raum befindet. Da im Folgenden ein Frame einer geometrischen Konstellation von Körpersegmenten entspricht, kann man anstatt von einem Frame auch von einer Pose P sprechen. Eine Pose P wird dabei durch $P \in \mathbb{R}^{3\times|J|}$ modelliert. Die j-te Spalte sei mit P^j bezeichnet. Sie enthält die 3D-Koordinaten des Gelenkes $j \in J$. Der Raum aller Posen sei mit $\mathcal{P}$ bezeichnet. Damit kann ein Motion-Capture-Datenstrom wie folgt als Funktion definiert werden:

$$D : [1 : T] \to \mathcal{P} \subset \mathbb{R}^{3\times|J|}.$$

T bezeichne die Länge des Datenstromes, also die Anzahl der enthaltenen Frames oder Posen. Dabei ist die Zeitachse mit einer festen Samplingrate unterteilt. Die Trajektorie eines einzelnen Gelenkes bezeichnet die durch das Gelenk beschriebene Bahnkurve: Es ist also die Projektion eines Datenstromes auf die 3D-Koordinaten eines bestimmten Gelenkes. Sie wird mit $\mathcal{D}^j = [1 : T] \to \mathbb{R}^3$ bezeichnet. Zwei Trajektorien sind in Abbildung 2.2 veranschaulicht.

Viele Anwendungen benötigen die Datenströme als Gelenkwinkeldaten eines Skelettes. Die vorhergehenden Definitionen gehen von einer Repräsentation eines Motion-Capture-Datenstromes aus, in der die Positionen der Gelenke und nicht die Position der Marker bekannt sind. Das Modell einer kinematischen Kette eignet sich gut für so eine Repräsentation des menschlichen Skelettes. Um die aufgenommenen Trajektorien der Marker in Gelenkwinkeldaten umzuwandeln, bedarf es eines Skelett-Einpassungs-Algorithmus. Er bildet die Daten auf eine allgemeine Repräsentation eines menschlichen Skelettes (Abbildung 2.1) ab. Eine Bewegung wird dabei durch die Angabe von zwei Informationen repräsentiert: Erstens muss das verwendete Skelett als kinematische Kette mit Längenangaben der Verbindungssegmente zwischen den Gelenken (also den Knochen) angegeben werden. Zweitens sind für jeden Frame die Gelenkwinkel anzugeben. Sind die C3D-Daten einmal in eine solche Repräsentation

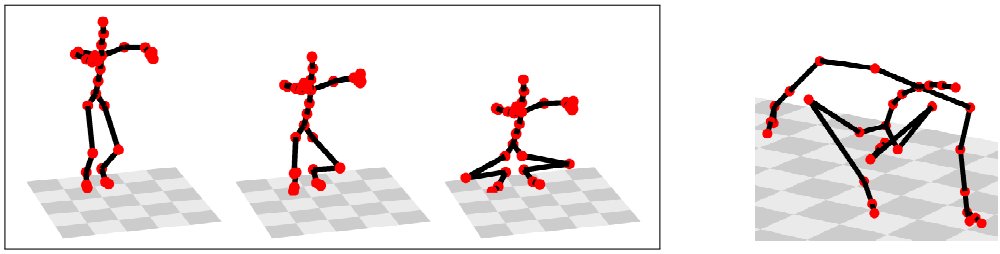
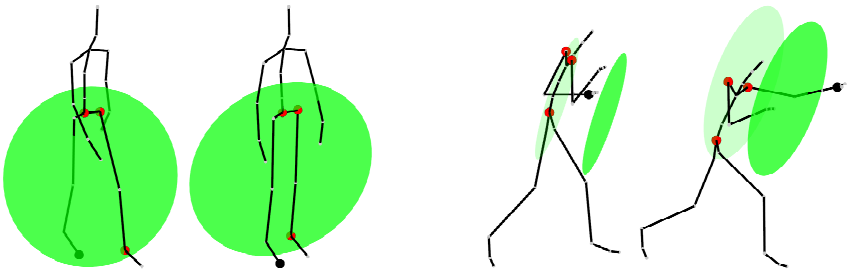


Abbildung 2.3. Links: Das Abspreizen der Knie nach außen in der dritten Pose kommt dadurch zustande, dass die Skelett-Einpassung Nebenbedingungen wie konstante Knochenlängen einhalten muss. Rechts: Eine Pose ist zu sehen, in der sich die Person auf den Boden legt. Durch die Skelett-Einpassung entsteht hier eine Pose, in der das Bein nach hinten gedreht wird.

überführt (z.B. im ASF/AMC-Format, siehe [ASF99]), so haben sie den Vorteil, dass dieses allgemeine Modell nicht von der Anzahl und Positionierung der verwendeten Marker abhängt. So lassen sich mit verschiedenen Markern aufgenommene Bewegungen gut vergleichen. Die Skelett-Einpassung ist jedoch ein fehlerbehafteter Prozess. Es wird bei der Repräsentation der Daten als kinematische Kette davon ausgegangen, dass die Knochenlängen über eine Bewegungssequenz konstant bleiben. Dadurch, dass die Marker an der Kleidung der Schauspieler angebracht werden und sich somit natürlicherweise ihre Position relativ zum Knochen bei der Bewegung verändert, können die Abstände zwischen zwei Markern in den C3D-Daten variieren, obwohl sie beide den selben Knochen repräsentieren sollen. Weitere Probleme treten durch Haut- und Masseverschiebung auf. Auch die Annahme, dass das menschliche Skelett eine kinematische Kette ist, ist falsch. In der Praxis sind Gelenke keine starren Konstrukte, die Segmente verbinden, sondern elastische Gebilde. Als Beispiel kann hier das Schultergelenk dienen, das in der Praxis nicht nur ein Kugelgelenk ist, sondern in sich viel Spiel für Verschiebungen des Gelenkes bietet. Damit ist klar, dass die Repräsentation als Kugelgelenk eine Annahme ist, die zu Artefakten in den Daten führen kann. Diese Artefakte können Unstetigkeiten in den ASF/AMC-Daten oder unrealistische Körperhaltungen sein, wie in Abbildung 2.3 links gezeigt. Hier klappen die Knie bei einer Kniebeuge auseinander, da der Einpassungs-Algorithmus diese Haltung vorgibt, um Nebenbedingungen wie die Unveränderlichkeit der Länge aller Knochen einzuhalten. Abbildung 2.3 rechts zeigt, wie das linke Bein eine völlig unnatürliche Haltung einnimmt.

Die Skelett-Einpassung ist, wie in [Ewe07] beschrieben, ein komplexer Prozess, der je nach Vorwissen unterschiedliche Arbeitsschritte enthält. Bei der Erstellung der HDM05-Daten waren die Marker durch ihre Kennung eindeutig einem Körperteil zugeordnet. Mit diesem Wissen kann der Skelett-Einpassungs-Prozess automatisiert ein Skelett erstellen, oder die Skelett-Topologie wird als Vorwissen in den Algorithmus eingegeben. Danach werden die Gelenkpositionen des Skelettes berechnet. In der Arbeit [Ewe07] wurde dies durch einen heuristischen Ansatz, der Vorwissens über die Markerpositionen am Skelett ausnutzt, durchgeführt.

Die HDM05-Motion-Capture-Daten enthalten alle Bewegungssequenzen sowohl im C3D, als auch im ASF/AMC-Format.



(a) Ist der rechte Fuß vorne? (b) Ist die linke Hand vorne?

Abbildung 2.4. Veranschaulichung zweier relationaler Features. In (a) wird gezeigt, wie geprüft werden kann, ob sich der rechte Fuß vor dem Körper befindet. Abbildung (b) zeigt, ein Merkmal, das bestimmen kann, ob die linke Hand einen bestimmten Abstand vom Körper hat. Die Abbildungen sind entnommen aus [Mül07].

2.3 Relationale Merkmale

Werden Motion-Capture-Daten mit Hilfe von Skelett-Einpassungs-Algorithmen in eine kinematische Kette umgewandelt, so haben die Daten eine implizite, semantische Annotation. Die Datenströme werden mit Positionen im menschlichen Skelett verbunden, so dass die Position jedes Gelenkes zu jedem Zeitpunkt bestimmt werden kann. Relationale Merkmale nutzen diese Informationen, um geometrische Relationen aus einer Pose oder einer kurzen Abfolge von Posen zu extrahieren. Diese Relationen können in Bewegungsdatenströmen verglichen werden.

Die Notation sei wie folgt festgelegt: Ein boolsches Feature F ist definiert als Funktion $F : \mathcal{P} \to \{0, 1\}$. Ein Vektor von f verschiedenen Featurefunktionen ($f \geq 1$) führt zu der Featurefunktion F als

$$F : \mathcal{P} \to \{0, 1\}^f.$$

Im Folgenden wird F die Featurefunktion genannt und $F(P)$ ist der Featurevektor einer Pose $P \in \mathcal{P}$. Wenn die Featurefunktion für jede Pose eines Motion-Capture-Datenstromes $D : [1 : T] \to \mathcal{P}$ ausgewertet wird, so entsteht eine F-Featurefolge $F(D)$.

Das anschauliche Prinzip der relationalen Merkmale sei an einem Beispiel erklärt. Für technische Details sei auf [MRC05] verwiesen. Betrachte ein boolsches Merkmal, welches angibt, ob der rechte Fuß sich vor (ausgedrückt mit dem Featurewert 1) oder hinter dem Körper befindet (Featurewert 0). Um diese Information zu erhalten, kann man prüfen, ob sich der rechte Fuß vor einer Ebene, die durch die Punkte „root“, „lhip“ und „lankle“ aufgespannt wird, befindet. Dies wird in Abbildung 2.4(a) veranschaulicht. Dieses Konzept lässt sich wie folgt verallgemeinern: Seien $p_i \in \mathbb{R}^3$ mit $1 \leq i \leq 4$ vier 3D-Punkte. Die ersten drei davon seien in allgemeiner Lage gegeben. Mit $\langle p_1, p_2, p_3 \rangle$ sei die orientierte Ebene, die durch die ersten drei Punkte aufgespannt wird, gegeben. Die Orientierung sei durch die Reihenfolge der Punkte gegeben. Definierte nun

$$B(p_1, p_2, p_3, p_4) := \begin{cases} 1 & , \text{ wenn } p_4 \text{ vor oder auf } \langle p_1, p_2, p_3 \rangle \text{ liegt,} \\ 0 & , \text{ wenn } p_4 \text{ hinter } \langle p_1, p_2, p_3 \rangle \text{ liegt.} \end{cases}$$

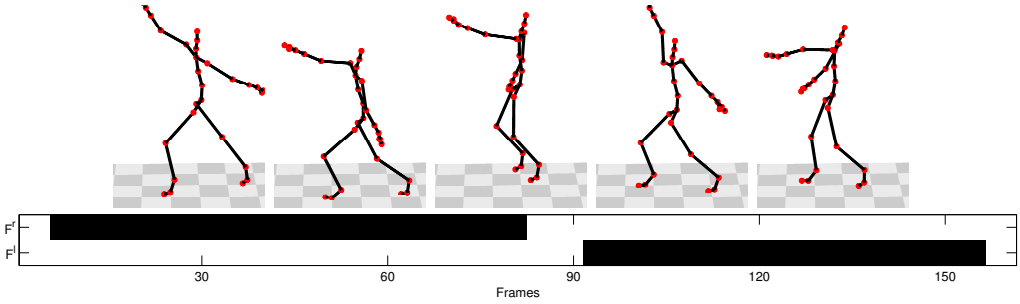


Abbildung 2.5. Man sieht die Werte der Featurefunktion F^2 zusammen mit ausgewählten Standposen an den angegebenen Framepositionen 30, 60, 90, 120 und 150. Die Featurematrix beschreibt die Posen, die in den Standbildern zu sehen sind.

Damit kann die Featurefunktion $F_{\text{plane}}^{(j_1,j_2,j_3;j_4)} : \mathcal{P} \to \{0,1\}$ für vier verschiedene Gelenke $j_i \in J$, $1 \leq i \leq 4$ definiert werden als

$$F_{\text{plane}}^{(j_1,j_2,j_3;j_4)}(P) := B(P^{j_1}, P^{j_2}, P^{j_3}; P^{j_4}).$$

Mit diesem Konzept lassen sich nun verschiedene Features bilden. Sei j_1 = „root“, j_2 = „lankle“, j_3 = „lhip“ und j_4 = „rtoes“. Dann ist $F^r := F_{\text{plane}}^{(j_1,j_2,j_3;j_4)}$ genau die Featurefunktion, die anzeigt, ob sich der rechte Fuß vor dem Körper befindet oder nicht. Setzt man nun für j_1 = „root“, j_2 = „rhip“, j_3 = „rankle“ und j_4 = „ltoes“, so erhält man die Featurefunktion $F^l := F_{\text{plane}}^{(j_1,j_2,j_3;j_4)}$, welche angibt, ob sich der linke Fuß vor dem Körper befindet oder nicht. Fügt man der Featurefunktion eine mögliche Verschiebung der Ebene hinzu, so lässt sich wie in Abbildung 2.4(b) prüfen, ob sich die linke Hand ausgestreckt vor dem Körper befindet oder nicht.

Man beachte, dass die so definierten Featurefunktionen invariant unter der Position, Orientierung und Größe des Skelettes sind. Ebenfalls spielt es für die Featurewerte von F^r und F^l keine Rolle, welche Bewegung im Oberkörperbereich ausgeführt wird. Es spielt auch keine Rolle, ob die Füße weit auseinander oder nahe zusammen bewegt werden. Das Konzept der relationalen Features ist also ein robustes Mittel zum extrahieren semantischer Informationen aus Bewegungsdaten. Unter Einbeziehung von Winkeldaten der Gelenke, Geschwindigkeits- und Bewegungsrichtungsinformationen ist es möglich, weitere semantische Aspekte von Motion-Capture-Daten zu boolschen Merkmalen umzuwandeln.

Abbildung 2.5 (oben) zeigt fünf Standbilder aus dem Dokument D_{skier}, das die Ausführung einer sportlichen Übung beinhaltet, die eine Skilanglauf-Bewegung nachahmt. Die Bewegung ist 161 Frames bei 120 Hz, also ca. 1.3 Sekunden lang, die abgebildeten Standbilder treten an den Framepositionen 30, 60, 90, 120 und 150 auf. Abbildung 2.5 (unten) zeigt eine Matrix, welche die Werte der Featurefunktion $F^2 = (F^r, F^l) : \mathcal{P} \to \{0,1\}^2$ veranschaulicht. Die Zeit wächst entlang der horizontalen Achse und eine Zeile visualisiert das Ergebnis einer Posenweise ausgewerteten Featurefunktion. Dabei wird der Featurewert 1 mit einem weißen Block, der Featurewert 0 mit eine schwarzen Block markiert. So ist zu erkennen, dass das Feature F^r währen der ersten ca. sieben Frames der Bewegung auf 1 ist, was bedeutet, dass die Bewegung mit dem rechten Fuß vor dem Körper beginnt. Gleichzeitig befindet sich auch der linke Fuß vor dem Körper, wie das Feature F^l angibt. In der Tat beginnt die Bewegung mit einer Standpose, in welcher beide Füße parallel stehen und daher die Featurewerte beide

1 sind. Nun beginnt eine Phase, in der sich das rechte Bein nicht mehr vor dem Körper befindet und das linke sich zuerst nach vorne und dann wieder zur Ausgangsposition zurück bewegt. Dies ist an den Standbildern zu den Frames 30 und 60 klar erkennbar. Dann stehen die Füße wieder parallel und beide Features nehmen den Wert 1 an. Schließlich sind die Featurewerte in Bezug auf Frame 30 umgekehrt: F^r nimmt den Wert 1 und F^l den Wert 0 an. Die hier vorgestellte Darstellung einer Featurefunktion wird Featurematrix genannt. Es ist eine anschauliche Darstellung der Features von Bewegungsdaten, an der Eigenschaften einer Bewegung in Bezug auf die benutzten Features schnell erfasst werden können.

2.4 Adaptive Segmentierung

Semantisch ähnliche Bewegungen können räumliche und zeitliche Deformationen aufweisen. Der vorhergehende Abschnitt hat gezeigt, wie mit Hilfe von relationalen Merkmalen Robustheit unter räumlichen Deformationen eingeführt werden kann. In diesem Abschnitt wird kurz gezeigt, wie auch zeitliche Variationen umgangen werden können (siehe auch [MRC05]).

Es sei $F : \mathcal{P} \to \{0,1\}^f$ eine Featurefunktion. Zwei Posen P_1, P_2 seien F-äquivalent, wenn die Features der Posen übereinstimmen, also wenn $F(P_1) = F(P_2)$ ist. Dann ist ein F-Lauf definiert als eine Teilfolge des Dokumentes D, die aus aufeinander folgenden F-äquivalenten Posen besteht. Nun sind die F-Segmente von D definiert als die nicht erweiterbaren F-Läufe von D. Für das Beispiel der Featurefolge der Skier-Bewegung (Abbildung 2.5) mit der Featurefunktion F^2 ergibt sich die F^2-Segmentierung des Dokumentes als

$$F^2[D_{\text{skier}}] = \left(\binom{1}{1}, \binom{0}{1}, \binom{1}{1}, \binom{1}{0}, \binom{1}{1}\right).$$

Es ist klar, dass verschiedene Ausführungen der Ski-Bewegung, egal ob sie schnell, langsam oder mit sich verändernder Geschwindigkeit ausgeführt werden, zu derselben F^2-Segmentierung führen. Daher können Vergleiche der Bewegungen auf deren F-Segmentierungen, anstatt auf den F-Featurefolgen durchgeführt werden.

In dieser Arbeit wird die Suche von Bewegungen mit Hilfe eines Indexes untersucht (siehe Kapitel 4). Der benutzte Index enthält dabei die F-Segmentierung $F[D]$ und nicht die F-Featurefolge $F(D)$. Somit wird nur ein Bruchteil der in den Bewegungen vorhandenen Frames in dem Index gespeichert und ein zusätzlicher Effizienzgewinn erreicht.

Kapitel 3

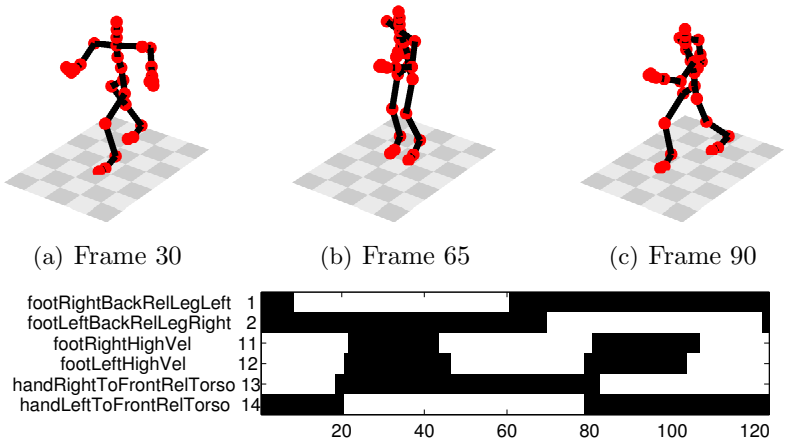
Motion Templates (MTs)

Motion Templates (MTs) sind eine Verallgemeinerung von boolschen Featurematrizen relationaler Features. Der Artikel [MR06] stellt das Prinzip von MTs und ihre Anwendungen vor. Sie sind dafür geeignet, die Gemeinsamkeiten einer Klasse von Bewegungen kompakt zu repräsentieren. Man kann sich MTs als eine verwaschene Form einer Featurematrix vorstellen, in welcher Gemeinsamkeiten in den Featurematrizen einer Klasse von Bewegungen durch klare, scharfe Werte repräsentiert werden. Eigenschaften, die innerhalb der Bewegungsklasse unterschiedlich sind und somit nicht zur semantischen Abgrenzung dieser Klasse beitragen, werden in einem MT durch unscharfe Werte repräsentiert.

Dieses Kapitel beginnt in Abschnitt 3.1 mit einer Motivation für MTs. Grundlagen werden anhand eines Beispiels erklärt. Ein Überblick über den Dynamic Time Warping-Algorithmus, der im Lernverfahren der MTs benutzt wird, wird in Abschnitt 3.2 gegeben. Dann wird ein Lernverfahren für eine neue Form eines Motion Templates in Abschnitte 3.3 vorgestellt. Diese Form eines MTs wird „hartes MT“ genannt. Aufgrund einer in Abschnitt 3.3.2 bewiesenen Eigenschaft bieten die harten MTs eine Grundlage für die Auswahl charakteristischer Schlüsseleigenschaften einer Klasse von Bewegungen. Den Namen „weiche MTs“ erhalten die in [MR06] vorgestellten MTs. Die Vorstellung experimenteller Retrievalergebnisse mit weichen MTs in Abschnitt 3.5 schließt dieses Kapitel ab.

3.1 Motivation

In Abschnitt 2.3 wurde gezeigt, wie geometrische Konstellationen und semantische Eigenschaften einer Bewegungssequenz mit Hilfe von relationalen Merkmalen erfasst werden können. Dabei wird bei einer Anzahl von f Features mit einer Featurefunktion $F : \mathcal{P} \to \{0,1\}^f$ die Bewegung in eine Featurematrix umgewandelt: Auf jede Pose wird die Featurefunktion F angewandt und das Ergebnis spaltenweise in die binäre Matrix $X \in \{0,1\}^{f \times T}$ eingetragen, so dass für $t \in [1:T]$ die t-te Spalte $X(t)$ den t-ten Featurevektor $F(D(t))$ enthält. Dabei werden die ursprünglichen Daten der 3D-Trajektorien stark vergröbert, wobei semantische Eigenschaften klar herausgearbeitet werden. Die Featurematrix X ist dann eine anschauliche Repräsentation der aufeinanderfolgenden Posen der Bewegung. Dies wird in Abbildung 3.1 deutlich. Hier ist die Bewegung einer sportlichen Übung dargestellt, in der die Person eine Skilanglauf-Bewegung imitiert. Startend mit einem Ausfallschritt, bei dem sich der rechte Fuß hinten befindet, werden die Positionen der Füße durch einen Sprung, in dem gleichzeitig der rechte und linke Fuß bewegt wird, vertauscht. Die Hände werden entsprechend mit bewegt. Der erste Ausfallschritt ist in Abbildung 3.1(a) dargestellt. In dieser Pose befindet sich der



(a) Frame 30 (b) Frame 65 (c) Frame 90

(d) Featurematrix

Abbildung 3.1. Die Korrelation der 3 dargestellten Positionen in (a), (b) und (c) zu der Featurematrix in (d) ist klar zu erkennen. Die Nummerierung der Features ist konsistent mit der Auflistung in Tabelle 5.1 auf Seite 65, die Featurematrix wird mit einer Framerate von 120 Hz dargestellt.

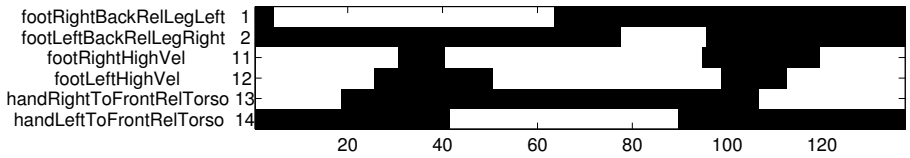


Abbildung 3.2. Die Featurematrix einer weiteren Ski-Bewegung zeigt große Gemeinsamkeiten zu Abbildung 3.1(d), jedoch auch Unterschiede.

rechte Fuß hinter dem linken, wie auch in der Featurematrix 3.1 an dem Feature 1 (Frame 30) zu sehen ist. Das Feature 14 zeigt an, dass sich in dem Moment die linke Hand nach vorne bewegt. Die Features 11 und 12 beschreiben, dass die Füße in dem Moment keine hohe Geschwindigkeit haben. Tatsächlich steht die Person in der angegebenen Pose kurz in Ruhe, bevor der Sprung in den anderen Ausfallschritt gestartet wird. Die rechte Hand bewegt sich in dem Moment nicht nach vorne, wie Feature 13 angibt. Abbildung 3.1(b) zeigt die Pose, in welcher sich die Füße beim Sprung in der Luft befinden und aneinander vorbei bewegt werden. In der Featurematrix ist klar zu erkennen, dass die Füße mit hoher Geschwindigkeit bewegt werden. Auch, dass die Füße sich nebeneinander befinden, kann an den Features 1 und 2 in Frame 65 erkannt werden. Schließlich bewegt sich die linke Hand (genau wie in Frame 30) nach vorne. Der Schluss des Sprungs aus einem in den anderen Ausfallschritt ist in Abbildung 3.1(c) dargestellt. Hier erreicht die Person wieder eine Pose, in der die Füße keine Bewegung ausführen. Es ist klar zu erkennen, dass die Features der rechten und linken Körperseite im Gegensatz zu Frame 30 vertauscht sind.

Nun ist zu erwarten, dass nicht jede Ski-Bewegung zu identischen Featurematrizen führt. Vielmehr werden die Featurematrizen einige strukturelle Gemeinsamkeiten haben, jedoch im Detail auch Unterschiede zeigen. Abbildung 3.2 zeigt die Featurematrix einer weiteren Bewegung derselben Klasse, wobei diese Featurematrix unter Anderem in Frame 100 einen strukturellen Unterschied zeigt: Hier ist eine Pose vorhanden, in welcher keines der Features auf 1 ist. Eine

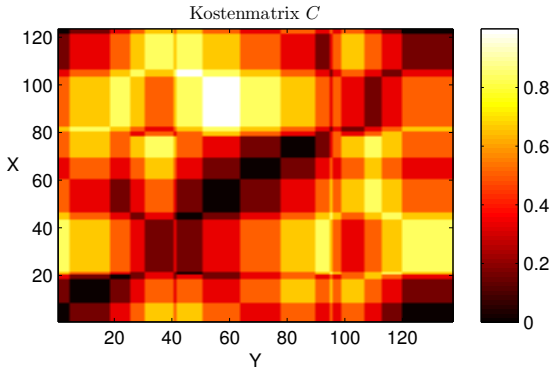


Abbildung 3.3. Kostenmatrix zwischen den Featurematrizen aus Abbildung 3.1 als X und 3.2 als Y.

solche Pose ist in der vorhergehenden Bewegung nicht vorhanden. Das im folgenden vorgestellte Konzept der *Motion Templates* erlaubt es, solche Unterschiede und Gemeinsamkeiten in einer kompakten Form zu repräsentieren.

Ein Motion Templates (MT) ist eine Art Featurematrix, welche die Gemeinsamkeiten und Unterschiede der Features innerhalb einer Klasse von Bewegungen darlegt. Es kann mit Hilfe von Trainingsbewegungen gelernt werden und codiert dann die Essenz einer Bewegungsklasse C. Eine wichtige Feststellung ist, dass Unterschiede innerhalb der Trainingsbewegungen verschiedene Variationsmöglichkeiten derselben Klasse anzeigen. Bei der Suche in einer Datenbank mit Hilfe von einem MT der Bewegungsklasse C wird genau diese Feststellung berücksichtigt, indem an den Positionen, an denen Inkonsistenzen in den Trainingsbewegungen auftreten, beliebige Variationen der Datenbank erlaubt werden.

Bevor das Verfahren zum Lernen der Motion Templates erklärt wird, soll die zugrundeliegende Technik des Dynamic Time Warping erklärt werden.

3.2 Dynamic Time Warping (DTW)

Dynamic Time Warping (DTW) ist ein Verfahren, welches die optimale zeitliche Ausrichtung zweier Featurefolgen unter gewissen Nebenbedingungen herstellt. Das Verfahren wird beim Lernen der MTs dazu benutzt, verschiedene Featurematrizen semantisch sinnvoll auf dieselbe Länge zu verzerren. Dabei ist die Verzerrung optimal gemäß einer zu spezifizierenden Kostenfunktion c, welche die Kosten zwischen zwei Featurevektoren definiert. Sei $\mathcal{F}$ die Menge aller möglichen Featurevektoren. Dann ist c definiert als Funktion

$$c : \mathcal{F} \times \mathcal{F} \to \mathbb{R}_{\geq 0}.$$

Diese Funktion soll zwei identische Featurevektoren auf den Kostenwert 0 abbilden. Der Grad der Verschiedenheit soll durch entsprechend hohe Kosten ausgedrückt werden. Um eine zeitliche Korrespondenz zwischen zwei Featurefolgen $X \in \mathcal{F}^N$ und $Y \in \mathcal{F}^M$ der Länge N und M herzustellen, wird zuerst eine Kostenmatrix $C \in \mathbb{R}^{N \times M}$ gebildet. Die Matrix entsteht, indem die Kostenfunktion für alle Paare von Features ausgewertet wird: $C(n, m) := c(X(n), Y(m))$ Als Beispiel dient die Featurematrix aus Abbildung 3.1 als X und diejenige aus Abbildung

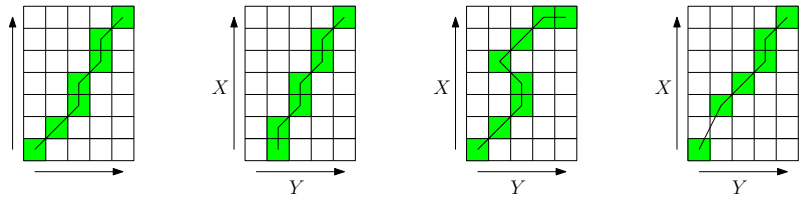


Abbildung 3.4. In (a) ist ein korrekter Warping-Pfad zu sehen. Die anderen Abbildungen zeigen Verletzungen der Bedingungen (i), (ii) und (iii) aus Definition 3.1.

3.2 als Y. Die Kostenfunktion c sei für $v, w \in \mathcal{F} = \{0,1\}^\ell$, wobei die Anzahl der Einträge in einem Featurevektor als ℓ definiert ist, gegeben als

$$c(v,w) = \sum_{i=1}^{\ell} |v(i) - w(i)|/\ell.$$

Die Kostenmatrix ist in Abbildung 3.3 dargestellt. Die Einträge der Matrix sind dabei farblich codiert: Niedrige Kosten entsprechen einer dunklen Farbe, hohen Kosten sind helle Farben zugeordnet. Es ist gut zu erkennen, dass entlang der Diagonalen die Kosten am Geringsten sind. Der Eintrag $(15, 8)$ in der Matrix (vertikale Achse: 15, horizontale Achse: 8) hat beispielsweise die Kosten 0. Das bedeutet mit dem gewählten Kostenmaß c, dass die Featurevektoren $X(15)$ und $Y(8)$ identisch sind: Es gilt

$$X(15) = Y(8) = (1,0,1,1,1,0).$$

Eine Zuordnung zwischen den beiden Featurefolgen wird durch das Konzept eines Warping-Pfades hergestellt.

Definition 3.1. *Ein Warping-Pfad (WP) ist eine Folge $p = (p_1, \ldots, p_L)$ von Tupeln $p_\ell = (j_\ell, k_\ell) \in [1:N] \times [1:M]$ für $\ell \in [1:L]$, so dass die folgenden Bedingungen eingehalten werden:*

(i) Randbedingung: $p_1 = (1,1)$ und $p_L = (N, M)$.

(ii) Monotoniebedingung: $1 \leq j_1 \leq j_2 \leq \ldots \leq j_L = N$ und $1 \leq k_1 \leq k_2 \leq \ldots \leq k_L = M$.

(iii) Schrittweitenbedingung: $p_{\ell+1} - p_\ell \in \{(1,0), (0,1), (1,1)\}$.

Abbildung 3.4 veranschaulicht die vorangehende Definition anhand zweier Beispielfolgen X mit Länge $N = 7$ und Y mit Länge $M = 5$. Ein nach Definition 3.1 korrekter Warping-Pfad ist in Abbildung 3.4(a) eingezeichnet. Hier werden beispielsweise das 5. und 6. Element von X dem 4. Element von Y zugeordnet. Der Warping-Pfad p, der hier aufgezeichnet ist, hat die Form

$$p = ((1,1),\ (2,2),\ (3,3),\ (4,3),\ (5,4),\ (6,4),\ (7,5)).$$

Abbildung 3.4 (a)-(d) zeigen ungültige Warping-Pfade: In (b) ist die Randbedingung verletzt. Bedingung (i) fordert, dass die Anfänge und Enden der beiden Featurefolgen miteinander assoziiert werden. In Abbildung (c) ist die Monotoniebedingung verletzt: Sie verbietet, dass der Pfad zeitlich rückwärts läuft. Schließlich zeigt (d), dass die Schrittweitenbedingung verletzt

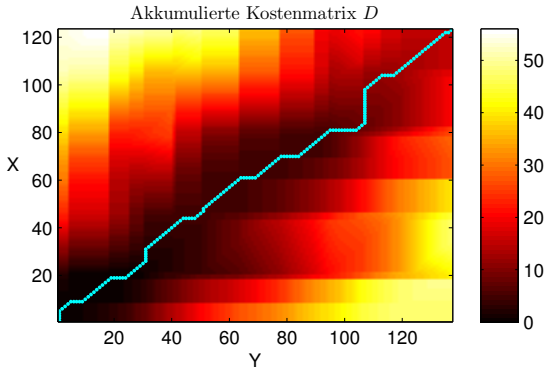


Abbildung 3.5. Die akkumulierte Kostenmatrix zusammen mit dem optimalen Warping-Pfad der beiden Featurematrizen aus Abbildung 3.1 als X und 3.2 als Y.

ist. Sie fordert, dass kein Element aus X oder Y bei der Zuordnung übersprungen werden darf.

Um verschiedene Warping-Pfade miteinander vergleichen zu können, werden mit der folgenden Definition die Kosten eines WPs festgelegt.

Definition 3.2. *Die Kosten eines Warping-Pfades p bezüglich einer Kostenmatrix C mit $\forall n \in [1:N], m \in [1:M] : C(n,m) = c(X(n), Y(m))$ sind definiert als*

$$c(p) := \sum_{\ell=1}^{L} C(j_\ell, k_\ell).$$

Die DTW-Kosten zwischen zwei Folgen X und Y werden wie folgt definiert:

Definition 3.3. *Es seien $X \in \mathcal{F}^N$ und $Y \in \mathcal{F}^M$. Es sei p^* ein WP zwischen X und Y mit minimalen Kosten unter allen möglichen WPs. Dann ist der DTW-Abstand* $\mathrm{DTW}(X,Y)$ *zwischen den Featurefolgen X und Y definiert als $c(p^*)$.*

Das Suchen eines kostenminimalen Warping-Pfades geschieht mittels dynamischer Programmierung und liegt sowohl in Bezug auf Laufzeit, als auch auf Speicherkomplexität in $O(N \cdot M)$. Der Algorithmus wird detailliert und in einem allgemeineren Kontext vorgestellt in [Mül07]. Aus der Kostenmatrix C wird mit Algorithmus 3.1 eine akkumulierte Kostenmatrix D gebildet. Für die beiden Beispielbewegungen ist diese in Abbildung 3.5 aufgezeichnet. Anhand der Matrix D lassen sich die Kosten des minimalen Warping-Pfades p^* zwischen X und Y ablesen: Der Eintrag $D(N,M)$ sind genau die Kosten $\mathrm{DTW}(X,Y)$. Der WP selbst lässt sich nun mit der in Algorithmus 3.2 angedeuteten Prozedur berechnen. Die Anzahl der benötigten Operationen ist linear in der Länge des Warping-Pfades. Bei der gewählten Schrittweitenbedingung beträgt die Länge des längstmöglichen Warping-Pfades genau $L = N + M - 1$, somit ist die Anzahl der benötigten Operationen von Algorithmus 3.2 linear in der Eingabelänge.

Der WP kann dafür benutzt werden, eine Featurematrix Y auf die Länge der anderen Matrix X nichtlinear zu verzerren. Dabei werden Teile von Y gestaucht oder gestreckt, um die beste Zuordnung zwischen X und Y zu erreichen. Das Ergebnis der Verzerrung von Y werde Y'

Algorithmus 3.1 : Akkumulierte Kostenmatrix

input : Kostenmatrix $C \in \mathbb{R}^{N \times M}$
output : akkumulierte Kostenmatrix $D \in \mathbb{R}^{N \times M}$

begin
  $D(1,m) \leftarrow \sum_{i=1}^{m} C(1,i)$ für $m \in [1:M]$
  $D(n,1) \leftarrow \sum_{i=1}^{n} C(i,1)$ für $n \in [2:N]$
  für alle noch nicht gesetzten $D(n,m)$ setze rekursiv:
  begin
    $D(n,m) \leftarrow \min\{D(n,m), D(n,m-1), D(n-1,m)\} + C(n,m)$
  end
end

Algorithmus 3.2 : Optimaler Warping-Pfad

input : akkumulierte Kostenmatrix $D \in \mathbb{R}^{N \times M}$
output : Optimaler Warping-Pfad p^*

begin
  Der optimale Warping-Pfad $p^* = (p_1, \ldots, p_L)$ wird beginnend mit $p_L = (N, M)$ berechnet.
  Angenommen, $p_\ell = (n, m)$ sei bereits berechnet.

$$p_{\ell-1} \leftarrow \begin{cases} (1, m-1) & \text{falls } n = 1 \\ (n-1, 1) & \text{falls } m = 1 \\ \operatorname{argmin}\{D(n-1,m-1), D(n,m-1), D(n-1,m)\} & \text{sonst} \end{cases}$$

end

genannt. Das Verfahren der Verzerrung soll hier nur kurz anhand eines Beispiels angeschnitten werden: In Abbildung 3.5 ist der optimale Warping-Pfad mit einer grünen Linie zwischen zwei Featurefolgen von Motion-Capture-Bewegungen dargestellt. Man sieht, dass die Frames $95 - 105$ von Y mit dem Frame 81 von X assoziiert werden. Dies bedeutet semantisch, dass in diesem Bereich die Bewegung Y langsamer als die Bewegung X ausgeführt wird. Um Y auf die Länge von X zu verzerren, kann der Framebereich $95 - 105$ von Y gestaucht werden, indem Frames übersprungen oder verschmolzen werden. Der Frame 107 von Y wird den Frames $84 - 98$ von X zugeordnet. Hier wird die Bewegung Y verlängert, indem der Frame 107 mehrfach wiederholt wird. Die so verzerrte Bewegung Y' hat die gleiche Länge wie X. Punktweise verglichen entsprechen sich nun diejenigen Features in X und Y', die in dem optimalen Warping-Pfad miteinander assoziiert wurden.

Für die Suche nach Bewegungen einer Bewegungsklasse wird der Algorithmus der Teilfolgen-DTW, eine Variante des DTW, angewandt. Hier wird eine typischerweise lange Featurefolge (die Datenbank) mit einer kurzen Featurefolge (der Anfrage) verglichen. Dabei ist das Ziel, aus der Datenbank diejenigen Ausschnitte zu extrahieren, die mit möglichst geringen Kosten zu der Anfrage passen. Dazu wird der DTW-Algorithmus leicht modifiziert. Für Details zur Teilfolgen-DTW sei auf [Mül07] verwiesen. Hier wird lediglich ein anschauliches Beispiel in Abbildung 3.6 gegeben. Eine Bewegungssequenz, die viele Sportbewegungen enthält, wird nach Ski-Bewegungen durchsucht. Dabei wird auf ähnliche Weise wie bei dem DTW-Algorithmus eine Kostenmatrix C und eine akkumulierte Kostenmatrix D berechnet. Abbildung 3.6(b) zeigt die akkumulierte Kostenmatrix. Die oberste Zeile von D sei die Trefferkurve Δ. Ihre Werte sind in Abbildung 3.6(a) gezeigt. Δ zeigt nun für jede Position zum Zeitpunkt t die minimal möglichen Kosten für einen Warping-Pfad, der an t endet. An den Positionen, an

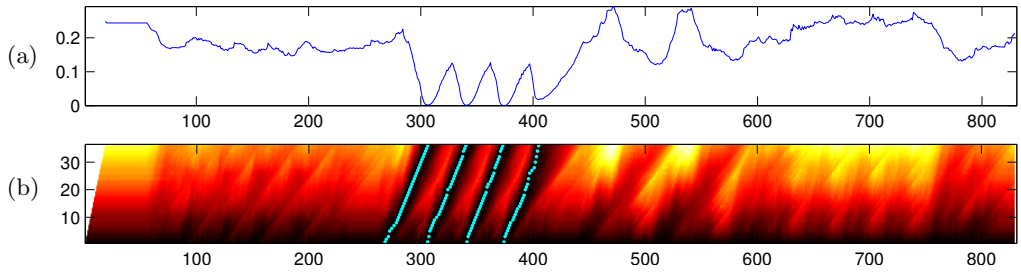


Abbildung 3.6. Hier ist das Ergebnis einer Teilfolgen-DTW illustriert. Die Eingabefolgen sind die Featurematrix einer längeren Bewegungsabfolge von Sportbewegungen und ein quantisiertes, weiches MT (siehe Abschnitt 3.3.3) der Ski-Bewegungsklasse. Das benutzte MT ist in Abbildung 3.12, Seite 30 abgebildet. Es wurde eine 30 Hz-Abtastrate benutzt. Die Trefferkurve Δ ist in (a) gezeigt. Die akkumulierte Kostenmatrix D mit eingezeichneten optimalen Warping-Pfaden der 4 Treffer ist in (b) dargestellt.

welchen Δ einen Wert nahe 0 hat, endet ein Treffer. Um den Anfang des Treffers zu finden, wird nun ähnlich dem Algorithmus 3.2 in der Matrix D der Weg bis zur untersten Zeile zurückverfolgt. Mit diesem Verfahren können also in einer Sequenz von Bewegungen mehrere Treffer gefunden werden.

3.3 Lernverfahren für MTs

Mit Hilfe einer Menge von N Beispielbewegungen für eine Bewegungsklasse kann ein MT gelernt werden, das die Gemeinsamkeiten und Unterschiede der Klasse in Bezug auf die extrahierten Features anschaulich beschreibt. Das Lernverfahren ist ein Iterationsverfahren, das in der Praxis mit wenigen Iterationsschritten konvergiert. Zuerst wird in Abschnitt 3.3.1 das Lernverfahren für so genannte „harte“ MTs beschrieben. Im Gegensatz zu den in [MR06] vorgestellten „weichen“ MTs ist für diese MTs der DTW-Abstand zu jeder der Trainingsbewegungen 0. Diese Eigenschaft zeigt, dass jede Spalte des harten MTs einen Featurevektor codiert, der in jeder der Trainingsbewegungen vorkommt. Dies ist eine nützliche Eigenschaft für die automatische Auswahl von Keyframes, was in Kapitel 6 noch näher erläutert wird. Die Eigenschaft wird in Abschnitt 3.3.2 bewiesen. Die harten MTs sind strikt in Bezug auf Inkonsistenzen innerhalb der Klasse. Unterschiede innerhalb der Features der Trainingsmenge werden unabhängig von der Häufigkeit ihres Auftretens codiert. Im Gegensatz dazu kann bei „weichen“ MTs abgelesen werden, wie stark die einzelnen Regionen des MTs konsistent oder inkonsistent innerhalb der Klasse aufgetreten sind. Die Unterschiede im Verfahren zu harten und weichen MTs werden kurz in Abschnitt 3.3.3 erläutert. Für weitere Details zu dem Lernverfahren für weiche MTs sei an [MR06] verwiesen.

3.3.1 Harte MTs

In diesem Abschnitt wird beschrieben, wie aus einer Menge von N Beispielbewegungen $\mathcal{X}^0 = \{(X_1^0, \alpha_1^0), \ldots, (X_N^0, \alpha_N^0)\}$ einer Bewegungsklasse ein hartes MT gelernt werden kann. Dabei ist X_n^0 die Featurematrix einer Bewegung bezüglich einer Featurefunktion F. Jeder Featurematrix wird dabei ein Gewichtsvektor α_n^0 zugeordnet. Während des Lernens werden

die Featurematrizen mit Hilfe von DTW (Abschnitt 3.2) verzerrt. Die Gewichtsvektoren α_n protokollieren dabei die zeitlichen Verzerrungen jeder Featurematrix: Wird eine Spalte verdoppelt, so wird das Gewicht auf die beiden Spalten gleichmäßig verteilt. Wird eine Spalte mit einer weiteren Spalte zusammengefasst, so werden die Gewichte addiert und der neuen Spalte zugewiesen. So können über die Gewichtsvektoren die Verzerrung der Featurematrizen wieder rückgängig gemacht werden, wobei durch das Zusammenfassen von Spalten verloren gegangene Informationen nicht rekonstruiert werden können. Die Gewichte α_n der Trainingsbewegungen werden für alle $n \in [1:N]$ mit $\alpha_n^0 \equiv 1$ initialisiert. Im Folgenden werden die in der Mittelung veränderten Featurematrizen Templates genannt.

Das Lernen der harten MTs geschieht wie folgt: Ein iteratives Verfahren bildet die Menge $\mathcal{X}^m = \{(X_1^m, \alpha_1^m), \ldots, (X_N^m, \alpha_N^m)\}$ aus der Menge $\mathcal{X}^{m-1}$ durch eine Mittelung. Die Mittelung ist referenzbasiert, da die einzelnen Bewegungen mit DTW auf die gleiche Länge - die Länge der Referenzbewegung - verzerrt werden. Um eine Beeinflussung des Ergebnisses durch die Wahl der Referenzbewegung zu eliminieren, wird in einem Iterationsschritt für jede Trainingsbewegung als Referenz eine Mittelung durchgeführt. Dieser Vorgang wird so lange wiederholt, bis sich alle Templates ähnlich genug sind.

Die Iteration wird mit der Menge $\mathcal{X}^0$ gestartet. Es wird ein Operator $\Omega_{(X,\alpha)}^{A\&}$ definiert, welcher das Mittel einer Menge $\mathcal{X}$ von Templates in Bezug auf ein Referenztemplate (X, α) bildet. Zuerst wird eine Definition eines weiteren benutzten Operators gegeben:

Definition 3.4. *Es seien $v, w \in \{0, 0.5, 1\}^L$. Dann ist $\oplus$ definiert durch*

$$\oplus : \{0, 0.5, 1\}^L \times \{0, 0.5, 1\}^L \to \{0, 0.5, 1\}^L$$

$$\forall \ell \in [1:L] : (v \oplus w)(\ell) := \begin{cases} v(\ell), & \textit{wenn } v(\ell) = w(\ell) \\ 0.5, & \textit{sonst.} \end{cases} \tag{3.1}$$

Der Operator $\oplus$ ist als eine Art harte Summe definiert. Dies wird am folgenden Beispiel deutlich: Sei $v = (0, 0.5, 1)$ und $w = (1, 1, 1)$. Dann ist $v \oplus w = (0.5, 0.5, 1)$. Sobald einer der Einträge den Wert 0.5 hat, ist der Eintrag in dem Ergebnisvektor ebenfalls 0.5. Nur, wenn beide Einträge identisch sind, wird der Wert beibehalten.

Nun wird beschrieben, wie eine Mittelung geschieht: Sei (X, α) das Referenztemplate und sei K die Länge von X. Sei $\mathcal{Y} = \{(Y_1, \beta_1), ..., (Y_N, \beta_N)\}$ die Menge von Templates, die mit der Referenz gemittelt werden soll. Da die Längen der Templates aus $\mathcal{Y}$ im Allgemeinen nicht mit der Länge K von X übereinstimmen, wird jedes Template der Menge $\mathcal{Y}$ zuerst auf die Länge K verzerrt. Dazu wird der DTW-Algorithmus verwendet. Die zeitlichen Deformationen werden dabei in den Gewichten protokolliert, in dem beim Stauchen von Spalten das Gewicht der Spalte erhöht wird und beim Strecken von Spalten das Gewicht auf die gedehnten Spalten gleichmäßig verteilt wird. Die für DTW benötigte Kostenfunktion $c_{\&}$ zwischen zwei Vektoren eines Templates $X(k)$ und $Y(\ell)$ der Länge f sei wie folgt definiert:

$$c_{\&}(k, \ell) = \sum_{i=1}^{f} d_{\&}(X(k)_i, Y(\ell)_i) \tag{3.2}$$

$$d_{\&}(x, y) = \begin{cases} 0, & \text{falls } x = 0.5 \text{ oder } y = 0.5 \\ |x - y|, & \text{sonst} \end{cases} \tag{3.3}$$

Hierbei bezeichnen $X(k)_i$ und $Y(\ell)_i$ die einzelnen Einträge in dem Vektor. Die Kostenfunktion wird anhand der folgenden Beispiele verdeutlicht.

Es sei $X(k) = (1,0,1,1)^T$ und $Y(k) = (1,1,0,0)^T$. Dann ist $c_{\&}(k,l) = 3$. Für $X(k) = (0.5,0,1,0.5)^T$ und $Y(k) = (1,1,0.5,0.5)^T$ ist $c(k,l) = 1$: Sobald einer der Einträge in einem Vektor 0.5 ist, trägt dieser nichts zu den Gesamtkosten bei. Semantisch bedeutet ein 0.5-Eintrag in einem Vektor eine Inkonsistenz in diesem Feature an der aktuell betrachteten Position im Template. Mit dieser Kostenfunkion erzeugen Bereiche, die als Inkonsistenzen innerhalb der Trainingsbewegungen markiert sind, keine Kosten. Dies geschieht aus der Motivation, dass Inkonsistenzen innerhalb der Trainingsbewegungen erlaubte Variationen in der Bewegungsklasse darstellen.

Es seien nun

$$(Z_n, \gamma_n) = \Omega^{W\&}_{(X,\alpha)}(Y_n, \beta_n)$$

für $n \in [1:N]$ diejenigen Templates, die auf die Länge K von X verzerrt wurden. Die zeitliche Verzerrung geschieht, indem bei einer Verlängerung einer Spalte diese dupliziert wird und das vorherige Gewicht der Spalte gleichmäßig auf die neuen Spalten verteilt wird. Bei einer Verkürzung des Templates werden adjazente Spalten v und w gemittelt. Dazu wird der Operator $\oplus$ verwendet, so dass in der gemittelten Spalte die Unterschiede zwischen den beiden Vektoren als Inkonsistenz markiert werden. Das Gewicht der neuen Spalte ergibt sich als Summe der Gewichte der bisherigen Spalten. Nun haben alle Templates in $\mathcal{Z} = \{(Z_1, \gamma_1), \ldots, (Z_N, \gamma_N)\}$ die Länge K. Das gemittelte Template (Z, γ) wird nun gebildet mit $k \in [1:K]$ durch

$$\begin{aligned} \gamma(k) &:= \frac{1}{N+1}\left(\alpha(k) + \sum_{n=1}^{N} \gamma_n(k)\right) \\ Z(k) &:= X(k) \oplus Z_1(k) \oplus \ldots \oplus Z_N(k). \end{aligned} \tag{3.4}$$

Jeder Eintrag $\gamma(k)$ im Gewichtsvektor ist also der Mittelwert aus den Gewichten des entsprechenden Eintrags in dem Referenztemplate und allen verzerrten Templates. Die Spalten des gemittelten Templates Z werden werden mit dem $\oplus$-Operator berechnet. Dadurch werden Inkonsistenzen innerhalb der zu mittelnden Vektoren mit dem Wert 0.5 als Inkonsistenz markiert.

Nach der Mittelung wird das Template Z mit Hilfe des gemittelten Gewichtes γ entzerrt. Dazu wird ein Expansions- und ein Kontraktions-Operator verwendet. Der Expansions-Operator Ω^E dehnt die Spalten $Z(k)$ eines Templates mit assoziiertem Gewicht $\gamma(k) \geq 1.5$ aus. Sei $[\gamma(k)]$ der ganzzahlig gerundete Wert von $\gamma(k)$. Betrachte nun alle $k \in [1:K]$ mit $\gamma(k) \geq 1.5$. Dann wird $Z(k)$ durch $[\gamma(k)]$ Kopien von $Z(k)$ ersetzt und jeder Kopie das Gewicht $\gamma(k)/[\gamma(k)]$ zugewiesen. Für das neue Gewicht gilt $0.75 \leq \gamma(k)/[\gamma(k)] < 1.5$. Das Ergebnis des Operators sei mit $\Omega^E(Z, \gamma)$ bezeichnet.

Der Kontraktions-Operator $\Omega^{C\&}$ kontrahiert das Template an den Stellen, an denen das Gewicht kleiner als 0.75 ist. Für $k \in [1:K]$ mit $\gamma(k) < 0.75$ und $k' := \operatorname{argmin}\{\gamma(k-1), \gamma(k+1)\}$ werden die beiden Gewichte $\gamma(k)$ und $\gamma(k')$ durch das neue Gewicht $\gamma := \gamma(k) + \gamma(k')$ ersetzt und die adjazenten Spalten durch den neuen Vektor $Z(k) \oplus Z(k')$ ersetzt. Diese Prozedur wird so lange wiederholt, bis alle Gewichte ≥ 0.75 sind. Das Ergebnis wird als $\Omega^{C\&}(Z, \gamma)$ bezeichnet.

Schließlich ist der Mittelungs-Operator $\Omega^{A\&}_{(X,\alpha)}$ definiert durch

$$\Omega^{A\&}_{(X,\alpha)}(\mathcal{Y}) := \Omega^{E}(\Omega^{C\&}(\Omega^{E}(Z,\gamma))). \tag{3.5}$$

Dieser Operator realisiert also eine Mittelung bezüglich einer Referenzbewegung. Man beachte, dass so aus einer Menge von Templates und einem Referenztemplate ein neues, gemitteltes Template gebildet wird. In dem späteren Iterationsverfahren wird in einem Iterationsschritt jedes Template der Trainingsmenge einmal als Referenz genommen und somit aus einer Menge von N Templates wiederum eine Menge von N Templates generiert.

Es fällt auf, dass nach der Expansion und anschließenden Kontraktion zum Schluss noch ein Expansionsschritt angewandt wird. Der Grund hierfür ist, dass nach der Kontraktion nicht garantiert werden kann, dass alle Gewichte zwischen 0.75 und 1.5 liegen: Ist das Gewicht einer Spalte kleiner als 0.75, so kann das Gesamtgewicht der neuen Spalte nach der Anwendung des Kontraktionsoperators größer als 1.5 sein, sofern die Gewichte der Nachbarspalten beide größer als 0.75 sind. Daher werden in einem anschließenden Expansionsschritt diese Spalten wieder geteilt. Es ist möglich, den ersten Expansionsschritt auszulassen, ohne die Bedingung für die Gewichte zu verletzen. Dadurch können jedoch ungewünschte Effekte entstehen, in welchen bei der Kontraktion Spalten mit stark unterschiedlichen Gewichten gemittelt werden. So könnte eine Spalte mit Gewicht 10 mit einer Spalte des Gewichtes 0.5 gemittelt werden. Semantisch sinnvoll ist es jedoch, die Spalte mit dem Gewicht 10 zuerst in 10 einzelne Spalten auszudehnen. Danach kann dann die Mittelung mit einer dieser Spalten geschehen. Um eine referenzfreie Mittelung zu realisieren wird ein Iterationsverfahren verwendet, in dem in einem Iterationsschritt jede der Bewegungen als Referenz genommen wird. Dabei entstehen aus einer Menge von N Templates nach einem Iterationsschritt wiederum N Templates, da jedes Template einmal als Referenz genommen wird. Die Menge der Templates des m-ten Iterationsschrittes $\mathcal{X}^m = \{(X_1^m, \alpha_1^m), \ldots, (X_N^m, \alpha_N^m)\}$ wird aus der Menge $\mathcal{X}^{m-1}$ gebildet durch folgende Iterationsvorschrift:

$$\forall n \in [1:N] : (X_n^m, \alpha_n^m) := \Omega^{A\&}_{(X_n^{m-1}, \alpha_n^{m-1})} \left(\mathcal{X}^{m-1} \setminus \left\{ (X_n^{m-1}, \alpha_n^{m-1}) \right\} \right). \tag{3.6}$$

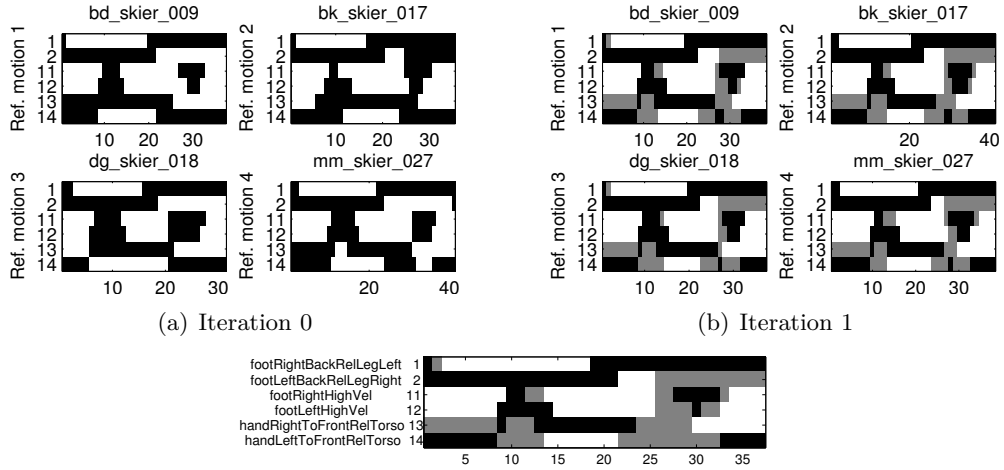
Es zeigt sich, dass die Templates einer Menge $\mathcal{X}^m$ im Verlaufe der Iteration $m = 1, 2, 3, \ldots$ immer ähnlicher zueinander im Sinne einer ℓ^1-Distanz werden. In der Praxis zeigt sich, dass schon nach wenigen Iterationsschritten das Verfahren konvergiert. Das harte MT wird nun als das Template, gegen das die Iteration konvergiert, definiert:

$$\Omega^{A\&}(\mathcal{X}^0) := (X^\infty, \alpha^\infty).$$

Abbildung 3.7 zeigt einen Ausschnitt des Iterationsprozesses für vier Skier-Bewegungen. Auf die Darstellung der Gewichte wird hier verzichtet. Um eine anschauliche Darstellung zu erhalten wird hier ein hartes MT mit 6 Features anstatt der in den Experimenten verwendeten 39 Features berechnet. Die Nummern der Features stimmen mit der Nummerierung in Tabelle 5.1 überein.

3.3.2 Eine Eigenschaft harter MTs

Es stellt sich heraus, dass harte MTs zu der Trainingsdatenbank den DTW-Abstand 0 haben, wenn man das Kostenmaß $c_{\&}$ benutzt. Dieses veranschlagt im Vergleich zweier Featurevektoren bei „Grauwerten“ (Wert 0.5) keine Kosten. Die Eigenschaft bedeutet, dass jede



(a) Iteration 0

(b) Iteration 1

(c) Iteration 4

Abbildung 3.7. Abbildung (a) zeigt die Menge $\mathcal{X}^0$ der Traningstemplates (30 Hz-Abtastrate). Die Werte der Templates sind farblich codiert: Der Wert 0 ist mit Schwarz, der Wert 0.5 mit Grau und 1 mit Weiß markiert. Nach einem Iterationsschritt sind schon viele Inkonsistenzen mit Grau markiert (Abbildung (b)). Nach 4 Iterationen ergeben sich keine Änderungen mehr und das harte MT X^∞ ist in (c) zu sehen.

Spalte des harten MTs in jeder Trainigsbewegung vorhanden ist. Dabei bedeutet ein Eintrag 0.5 in dem Vektor, dass dieses Feature in der Trainingsbewegung den Wert 0 oder 1 annehmen darf. In diesem Sinne können gemeinsame Eigenschaften aller Trainingsbewegungen schnell erkannt werden. Die Spalten des harten MTs können damit als erster Ansatz für die Gewinnung von Featurevektoren, die charakteristisch für eine Klasse von Bewegungen sind, gelten. Wählt man Featurevektoren aus dem harten MT aus, so ist garantiert, dass diese Vektoren in allen Trainingsbewegungen vorkommen.

Um zu zeigen, dass die DTW-Distanz zwischen einer Trainingsbewegung und dem harten MT 0 beträgt, wird ein Hilfssatz benutzt:

Lemma 3.1. *Seien X, Y, $Z \in \mathcal{F}$ Folgen mit* $\mathrm{DTW}(X,Y) = 0$ *und* $\mathrm{DTW}(Y,Z) = 0$ *unter einer Kostenfunktion $c : \mathcal{F} \times \mathcal{F} \to \mathbb{R}_{\geq 0}$ und der Schrittweitenbedingung $d \in \{(0,1),(1,0),(1,1)\}$. Dann ist* $\mathrm{DTW}(X,Z) = 0$.

Beweis 3.1. Das Lemma ist klar, wenn man die anschauliche Bedeutung betrachtet. Zwei Spalten v und $w \in \mathcal{F}$ seien äquivalent, wenn $c(v,w) = 0$ gilt. Die Aussage $\mathrm{DTW}(X,Y) = 0$ bedeutet, dass die äquivalenten Spalten von X und Y in der selben Reihenfolge vorkommen, sofern man die Vielfachheiten ignoriert. X und Y beginnen und enden also mit zueinander äquivalenten Spalten. In der Mitte der Folgen kann eine Spalten aus Y in X oder eine Spalte aus X in Y mehrfach hintereinander vorkommen, sofern sie äquivalent zueinander sind. Sie kommen jedoch in beiden Folgen in derselben Reihenfolge vor.

Somit kommen in X und Y die Spalten in derselben Reihenfolge vor, in Y und Z ebenfalls. Dann kommen die Spalten auch in X und Z in derselben Reihenfolge vor und es gilt $\mathrm{DTW}(X,Z) = 0$. □

Die folgende Behauptung für die harten MTs soll hier bewiesen werden.

Behauptung 3.1. *Die DTW-Distanz aller Trainingsbewegungen zu dem harten MT beträgt 0 bei Kostenfunktion $c_{\&}$ und DTW-Schrittweitenbedingung $d \in \{(0,1),(1,0),(1,1)\}$.*

Beweis 3.2. Sei $X \in \{0, 0.5, 1\}^{f \times K}$ ein Template. Wird X durch eine der folgenden Operationen modifiziert, dann gilt für das resultierende Template X': $\mathrm{DTW}(X, X') = 0$.

Ersetzte einen Eintrag von X durch 0.5. Da der Wert 0.5 in der Kostenfunktion $c_{\&}$ keine Kosten verursacht, hat hier der Warping-Pfad

$$p = ((1,1),\ (2,2),\ \ldots,\ (K,K))$$

die Kosten 0.

Verdopple eine Spalte von X. Angenommen, die k-te Spalte von X wird wie folgt verdoppelt: $\forall i \in [1:k] : X'(i) = X(i)$ und $X'(k+1) = X(k)$ und $\forall i \in [k+1:K] : X'(i+1) = X(i)$. Mit dem Warping-Pfad zwischen X und X'

$$p = ((1,1),\ (2,2),\ \ldots,\ (k,k),\ (k,k+1),\ (k+1,k+2),\ \ldots,\ (K,K+1))$$

gilt $c(p) = 0$, da mit $c(k,k) = 0$ direkt $c(k,k+1) = 0$ folgt.

Verschmelze zwei benachbarte Spalten von X. Angenommen, die zwei benachbarten Spalten k und $k+1$ von X werden wie folgt durch den Operator $\oplus$ verschmolzen: $\forall i \in [1 : k-1] : X'(i) = X(i)$ und $X'(k) = X(k) \oplus X(k+1)$ und $\forall i \in [k+1 : K] : X'(i-1) = X(i)$. Dann hat der Warping-Pfad

$$p = ((1,1),\ (2,2),\ \ldots,\ (k,k),\ (k+1,k),\ (k+2,k+1),\ \ldots,\ (K,K-1))$$

die Kosten 0. Dieser Warping-Pfad assoziiert mit (k,k) und $(k+1,k)$ genau $X(k)$ mit $X'(k)$ und $X(k+1)$ mit $X'(k)$. Da in $X'(k)$ durch die Konstruktion des $\oplus$-Operators alle Positionen, an denen sich $X(k)$ und $X(k+1)$ unterscheiden, den Wert 0.5 haben, sind die Kosten $c(k,k) = c(k+1,k) = 0$.

Es bleibt zu zeigen, dass bei der Iteration zur Berechnung von X^{∞} nur solche Operatoren verwendet werden.

Die Iterationsvorschrift (Gleichung (3.6)) benutzt nur den Operator $\Omega^{A\&}_{X,\alpha}(\mathcal{Z})$ der referenzbasierten Mittelung bezüglich der Referenz (X, α). Dieser wiederum benutzt nach Gleichung (3.5) die Operatoren Ω^{E} und $\Omega^{C\&}$. Diese Operatoren sind so definiert, dass sie Spalten vervielfachen (was als Hintereinanderausführung von Verdopplungen modelliert werden kann) oder adjazente Spalten mit dem $\oplus$-Operator verschmelzen.

Der Mittelungs-Operator $\Omega^{A\&}_{X,\alpha}(\mathcal{Z})$ wird auf die Menge $\mathcal{Z}$ angewandt, welche nach Gleichung (3.4) aus einer Anwendung des $\oplus$-Operators auf die Referenz X und alle Templates aus $\mathcal{Z}$ besteht. Das bedeutet für ein beliebiges Template T aus $X \bigcup \mathcal{Z}$, dass man Z aus T erzeugen kann durch das Ersetzen von bestimmten Einträgen aus T durch den Wert 0.5.

Alle Templates der Menge $\mathcal{Z}$ entstehen durch die zeitliche Verzerrung der Templates aus $\mathcal{Y}$. Dabei beachte man, dass $X \bigcup \mathcal{Y}$ die gesamte Menge aller Templates ist, mit der eine Iteration gestartet wird. Bei der zeitlichen Verzerrung der Templates aus $\mathcal{Y}$ werden ausschließlich Zeilen eines Templates vervielfacht oder adjazente Zeilen mit dem $\oplus$-Operator gemittelt.

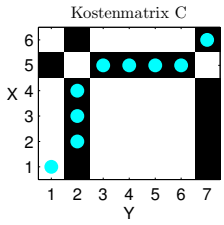


Abbildung 3.8. Kostenmatrix zwischen den Templates X und Y. Schwarze Blöcke bedeuten Kosten 0, mit Weiß wird der Kostenwert 1 ausgedrückt.

Damit ist gezeigt, dass ausgehend von einer beliebigen Menge von Templates $\mathcal{X}$, die aufgeteilt wird in ein Referenztemplate X und eine Menge $\mathcal{Y} := \mathcal{X} \setminus X$, das Ergebnis Z der referenzbasierten Mittelung durch eine Folge der drei beschriebenen Operationen, angewandt auf ein beliebiges Template aus $\mathcal{Y}$, hergestellt werden kann. Damit kann jedes X_ℓ^m für $m \geq 1$ und $\ell \in [1:N]$ aus jedem X_n^0 mit $n \in [1:N]$ durch eine Folge der drei Operationen abgeleitet werden. Damit gilt dies auch für das Template X^∞. Weil die Operationen jeweils einen Warping-Pfad ohne Kosten erzeugen, sind nach Lemma 3.1 auch die Kosten von X^∞ zu jedem beliebigen Template der Trainingsmenge gleich 0. □

Das folgende Beispiel führt einen Iterationsschritt zur Bildung eines harten MTs anhand zweier Beispieltemplates durch und zeigt, dass die DTW-Kosten eines Trainingstemplates zu dem Ergebnis der referenzbasierten Mittelung 0 betragen.

Beispiel 3.1. Die Menge $\mathcal{X}^0$ sei gegeben durch

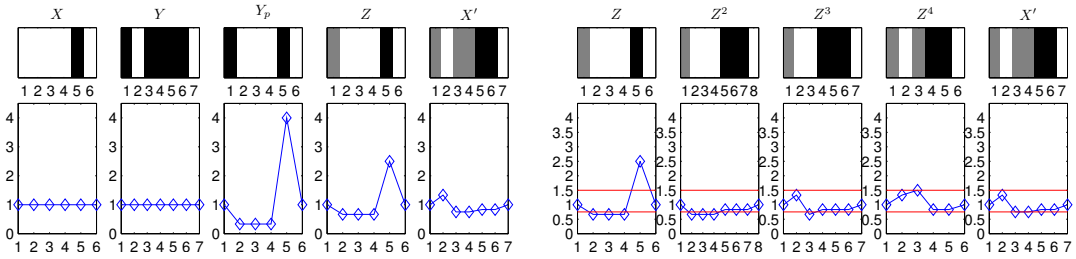
$$\begin{aligned}\mathcal{X}^0 &= \{(X,\alpha),(Y,\beta)\}\\ X &= [1,\ 1,\ 1,\ 1,\ 0,\ 1]\\ Y &= [0,\ 1,\ 0,\ 0,\ 0,\ 0,\ 1]\\ \alpha &\equiv 1\\ \beta &\equiv 1\end{aligned}$$

Die Referenzbewegung sei X. Die zugehörige Kostenmatrix mit Warping-Pfad ist in Abbildung 3.8 zu sehen. Dabei sind die Kosten farblich codiert: Kosten 0 sind mit einem schwarzen Block und Kosten 1 mit einem weißen Block kenntlich gemacht. Der Warping-Pfad p ist

$$p = ((1,1),(2,2),(3,2),(4,2),(5,3),(5,4),(5,5),(5,6),(6,7)).$$

In Abbildung 3.9(a) sind die Templates X und Y, sowie das mit Hilfe von p auf die Länge von X verzerrte Template Y_p zu sehen. Die Gewichte der einzelnen Templates sind in der unteren Zeile aufgezeichnet. Aus dem Warping-Pfad p geht hervor, dass die zweite Spalte von Y in die Länge gezogen wird und die dritte bis sechste Spalte verschmolzen werden. Daher kommen die geringen Gewichte in $Y_p(2), Y_p(3)$ und $Y_p(4)$, sowie das hohe Gewicht von $Y_p(5)$ zustande. Wie in Abbildung 3.9(a) zu sehen ist, führt die Verdreifachung der Spalte 2 aus Y zu den Spalten $2-4$ in Y_p. Die Spalten 3 bis 6 aus Y werden mit $\oplus$ zu der Spalte 5 in Y_p verschmolzen. Nun werden die Templates Y_p und X zum Template Z mit dem Operator $\oplus$ gemittelt. Das so entstandene Template Z wird nun noch ausbalanciert durch

$$X' = \Omega^E(\Omega^{C\&}(\Omega^E(Z,\gamma)))$$



(a) (b)

Abbildung 3.9. Die beiden Templates in der Trainingsmenge werden gemittelt. In (a) ist der Mittelungsprozess im Überblick zu sehen. Abbildung (b) zeigt den Schritt der Expansion und Kontraktion, welche in der Generierung von X' aus Z geschieht, in Einzelschritten dargestellt. Die roten Linien zeigen den Bereich, in dem sich die Gewichte nach der Expansion und Kontraktion befinden müssen.

Das Ergebnis X' ist Abbildung 3.9(a) ganz rechts dargestellt. Die einzelnen Schritte der Expansion und Kontraktion werden im Folgenden dargestellt, der Prozess ist in Abbildung 3.9(b) dargestellt.

Zuerst wird auf Z der Expansionsoperator Ω^E angewandt. Nenne das Ergebnis Z^2.

$$Z^2 = \Omega^E(Z)$$

Dabei wird der Frame 5 in Z aufgrund seines hohen Gewichtes von 2.5 zu den Frames 5 bis 7 verlängert. Im nächsten Schritt wird der Operator $\Omega^{C\&}$ angewandt. Dabei werden zuerst die Spalten 2 und 3 aus Z^2 verschmolzen. Das Ergebnis ist als Z^3 abgebildet. Dann werden die Spalten die Spalten 3 und 4 aus Z^3 verschmolzen. Das Ergebnis des Kontraktionsoperators ist als Z^4 zu sehen. Im letzten Schritt wird noch einmal der Expansionsoperator Ω^E angewandt:

$$X' = \Omega^E(Z^4)$$

Dabei wird die Spalte 3 verdoppelt, da sie genau das Gewicht 1.5 hat. Bei der Transformation von Y zu X' wurden ausschließlich die in Beweis 3.2 angegebenen Operationen verwendet. Ein möglicher Warping-Pfad mit Kosten 0 zwischen Y und X' ist

$$p = ((1,1),\ (2,2),\ (3,3),\ (4,4),\ (5,5),\ (6,6),\ (7,7))$$

3.3.3 Weiche MTs

Im Gegensatz zu harten MTs werden Inkonsistenzen in den Trainingstemplates bei weichen MTs abhängig von der Stärke der Inkonsistenz mit reellen Zahlen zwischen 0 und 1 codiert. Ein weiches MT ist eine Matrix $X \in [0,1]^{f \times T}$ der Länge T bei einer Featurefunktion F der Länge f. Beim Lernen der weichen MTs sind die wesentlichen Unterschiede zum Lernverfahren der harten MTs, dass eine andere DTW-Kostenfunktion c, um die zeitliche Korrespondenz der Templates herzustellen, und eine andere Mittelung zwischen den Templates verwendet wird. Als Kostenfunktion c zwischen zwei Featurevektoren wird die ℓ^1 (auch Manhattan-Distanz genannt) verwendet. Im ersten Iterationsschritt des Lernverfahrens, in welchem nur boolsche Featurevektoren betrachtet werden (da die Featurematrizen nur boolsche Vektoren enthalten), entspricht dieses Distanzmaß der Hamming-Distanz zwischen boolschen Vektoren.

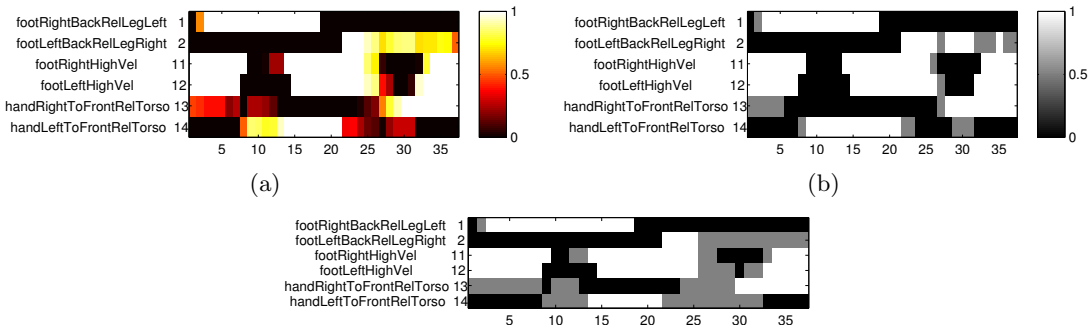


Abbildung 3.10. In (a) ist das weiche MT zu sehen, das mit den gleichen 4 Beispieltemplates wie bei dem harten MT in Abbildung 3.7(c) gebildet wird. Abbildung (b) zeigt die quantisierte Version des weichen MTs unter Verwendung des Quantisierungsschwellwertes $\delta = 0.25$ zu sehen. Schließlich ist zum direkten Vergleich das harte MT der Klasse in (c) erneut abgebildet.

Die Mittelung geschieht, indem eine gewichtete Mittelung zwischen allen korrespondierenden Featurevektoren gebildet wird. Dadurch wird mit den Werten in dem weichen MT der Grad der Übereinstimmung aller Trainingsbewegungen ablesbar. Ein Wert von 0.9 bedeutet beispielsweise, dass 90% der Trainingstemplates an dieser Stelle den Wert 1 und 10% der Templates den Wert 0 haben. Diese Überlegung ist nicht exakt, sondern nur eine Orientierungshilfe, da der Lernprozess keine Mittelung in einem Schritt, sondern ein iteratives Verfahren mit mehreren Schritten ist, in denen jeweils zeitliche Korrespondenzen hergestellt werden.

Um die Semantik der Templates zu verstärken werden nun die Regionen in einem Template, bei denen die meisten der Trainingstemplates Inkonsistenzen zeigen, auf 0.5 gesetzt. Dies bietet zusätzliche Toleranz bezüglich der Trainingsmenge: Wenn beispielsweise 3 der 10 Trainingstemplates in der jeweiligen zeitlich korrespondierenden Spalte den Featurewert 0 und die restlichen 7 den Featurewert 1 haben, so liegt der Featurewert des weichen MTs bei 0.7. Dies würde beim Retrieval einen Tendenz verursachen, den Featurewert 1 an dieser Position mit weniger Kosten zu versehen als den Featurewert 0. Wäre die Trainingsmenge genau umgekehrt verteilt, so würde der Featurewert 0 an dieser Position weniger Kosten verursachen. Im Wesentlichen sagt die Trainingsmenge jedoch aus, dass an dieser Position die Trainingsmenge inkonsistent ist und der Featurewert an dieser Position also keinen Einfluss auf die Semantik der Bewegungsklasse hat. Aus dieser Motivation ist es gut, die Feature-Position mit dem Wert 0.5 als inkonsistent zu markieren und beim Retrieval den mit 0.5 markierten Positionen keine Kosten zuzuweisen.

Um effizientes Retrieval betreiben zu können, ist es ebenfalls wichtig, aus dem reellwertigen Template ein Template mit einem kleinen, endlichen Wertebereich zu bilden. Mit einem endlichen Wertebereich kann eine Indexierung der Datenbank vorgenommen werden. Soll nun aus einem weichen MT ein Template aus $\{0, 0.5, 1\}^{\{f \times T\}}$ generiert werden, so kann dies durch eine Quantisierung mit $\delta \in [0, 0.5)$ geschehen. Dabei werden alle Werte in dem weichen MT, die kleiner als δ sind, durch 0 ersetzt. Alle Werte, die größer als $1 - \delta$ sind, werden durch 1 ersetzt, die restlichen Positionen werden mit 0.5 überschrieben.

Abbildung 3.10(a) zeigt das weiche MT, welches mit den vier Trainingsbewegungen, die auch beim Lernen des harten MTs in Abbildung 3.7 auf Seite 23 verwendet wurden, berechnet wird.

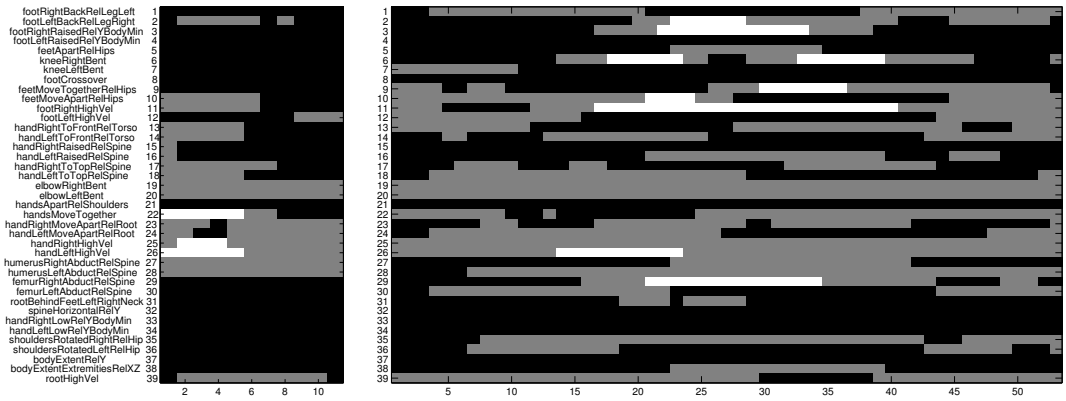
Dabei sind die Werte farblich mit der angegebenen Farbskala codiert. Um ein quantisiertes MT zu erhalten wird in diesem Beispiel der Schwellwert $\delta = 0.25$ gewählt. Semantisch bedeutet dies bei 4 Trainingstemplates, dass Werte als konsistent erachtet werden, auch wenn eines der vier Templates einen anderen Wert als die drei anderen hat. Es wird damit also ein „Ausreißer“ erlaubt, bevor eine Inkonsistenz markiert wird. Wie schon vorher festgestellt wurde ist diese Überlegung nicht exakt, sondern nur eine Orientierungshilfe. Im direkten Vergleich mit dem harten MT der Klasse, abgebildet in 3.10(c), fällt auf, dass in dem quantisierten weichen MT weniger Regionen als inkonsistent markiert sind. Hier macht sich deutlich die Toleranz gegenüber „Ausreißern“ bemerkbar.

3.4 Retrieval mit MTs

Das Ziel einer inhaltsbasierten Suche nach Bewegungsdaten ist es, Bewegungen einer bestimmten Klasse automatisch aus einer Datenbank von unbekannten Bewegungen zu extrahieren. Man nimmt an, dass die Bewegungsdatenbank $\mathcal{D}$ nur aus einem Dokument D besteht. Dies kann durch Konkatenation der enthaltenen Dokumente geschehen, wobei in einer zusätzlichen Datenstruktur die Positionen der einzelnen Dokumente gespeichert werden. Man möchte nun Teilfolgen von D finden, die ähnlich zu dem MT der Klasse sind. Dazu wird der Teilfolgen-DTW-Algorithmus verwendet, wie er in Abschnitt 3.2 beschrieben wurde. Das Kostenmaß, welches bei dem Teilfolgen-DTW-Algorithmus verwendet wird, ist das Kostenmaß $c_{\&}$ aus 3.2. Es weist beim Vergleich zweier Vektoren Bereichen, die mit dem Wert 0.5 als Inkonsistenzen in den Trainingsbewegungen markiert sind, keine Kosten zu. Somit werden nur die konsistenten Aspekte der Trainingssequenzen mit der Datenbank verglichen. Alle Bereiche der Datenbank, deren Kosten beim Vergleich mit dem MT der Anfrage unterhalb eines gewissen Schwellwertes τ liegen, werden als Treffer zurückgegeben.

3.5 Experimentelle Resultate

In den praktischen Experimenten zum Retrieval mit MTs wurden weiche MTs verwendet, die mit einem Schwellwert $\delta = 0.1$ quantisiert wurden. Diese eignen sich besser für Retrieval-Aufgaben als die harten MTs. Der Grund hierfür ist, dass die harten MTs sämtliche Inkonsistenzen markieren und keine Spielraum für kleine Abweichungen innerhalb der Trainingstemplates lassen. Dadurch entstehen Templates mit oft hohem Anteil an 0.5-Werten („Grauwerten“). In der Praxis zeigt sich, dass diese Templates nicht so gut wie die weichen MTs geeignet sind, um die Kosten eines richtigen Treffers von den Kosten eines falschen Treffers zu separieren. Schon allein die in Abschnitt 3.3.2 bewiesene Tatsache, das der DTW-Abstand des harten MTs zu den Trainingstemplates 0 beträgt, zeigt, dass die beim Retrieval auftretenden Kosten oft sehr nahe an 0 liegen. Im Gegensatz dazu wird beim Retrieval mit quantisierten weichen MTs stärker berücksichtigt, wie gut die Ausschnitte der Datenbank zu den Trainingstemplates passen, da die weichen MTs auch bei kleinen Inkonsistenzen der Trainingstemplates eine Konsistenz anzeigen können und die Bereiche somit beim Retrieval zu kleinen Kosten führen können.



(a) (b)

Abbildung 3.11. In (a) ist das quantisierte weiche MT der Klasse clap1Reps zu sehen (Schwellwert $\delta = 0.1$), das mit den Trainingstemplates aus der $\mathcal{D}^{57}$ (Anhang A) trainiert wird. In (b) ist das entsprechende MT für die Klasse kickRFront1Reps abgebildet.

Es wurden ausgedehnte Retrieval-Tests auf der Evaluationsdatenbank der $\mathcal{D}^{57}$ (siehe Tabelle A.1 im Anhang A) durchgeführt. Die Ergebnisse der Tests sind in Tabelle 3.1 (Seite 32) aufgeführt. In der Tabelle gibt die erste Spalte die angefragte Bewegungsklasse an. Mit γ wird die Anzahl der relevanten Treffer pro Klasse bezeichnet. K ist Länge des Motion Templates in Frames in Bezug auf 30Hz. Für die Klasse cartwheelLHandStart1Reps und $K = 106$ hat das weiche MT eine Länge von $106/30 \approx 3.5s$. Im Durchschnitt dauern die Trainings-Radschlag-Bewegungen also 3.5 Sekunden. t^T gibt die Zeit in Sekunden an, die das Durchsuchen der Datenbank mit dem MT beansprucht. Dabei wurden die Tests auf einem Laptop mit Intel® Pentium® M Prozessor mit 1500 MHz und 1.25 GB RAM durchgeführt. Schließlich gibt die Spalte $|H_\tau|/|H_\tau^+|$ einen Hinweis auf die Qualität der Suchergebnisse. Die jeweils erste Zeile zeigt die Anzahl der zurückgelieferten Treffer. Die jeweils zweite Zeile zeigt die Anzahl der relevanten zurückgelieferten Treffer. Dabei werden in den verschiedenen Spalten unterschiedliche Schwellwerte für den DTW-Abstand der Treffer zugrunde gelegt: Es werden nur Treffer betrachtet, deren DTW-Kosten beim Vergleich mit dem MT kleiner als ein bestimmter Schwellwert τ sind. Die verwendeten Schwellwerte sind $\tau = 0.01$, 0.02, 0.04, 0.06, 0.08 und 0.1 (von links nach rechts).

Bei der Analyse fällt auf, dass mit dem Treffer-Schwellwert $\tau = 0.1$ in fast allen Klassen die relevanten Dokumente gefunden werden. Ausnahmen sind die Klassen clap1Reps, depositHighR, kickRFront1Reps und punchRFront1Reps. Bei diesen Klassen ist gleichzeitig auch die Präzision der Ergebnisse sehr gering: Bei der Bewegungsklasse clap1Reps wird sogar die maximal mögliche Trefferanzahl von 500 erreicht. Weitere Klassen mit schlechter Präzision sind jogOnPlaceStartFloor2StepsRStart, standUpSitTable, turnLeft, turnRight und walkOnPlace2StepsLStart. Trotzdem sind bei diesen Klassen die Retrieval-Ergebnisse immer noch akzeptabel, da unter den ersten 30 Treffern der Großteil der relevanten Dokumente vorhanden ist. Hierbei sind die Gründe für die bei diesen Klassen sehr vielen zurückgelieferten Treffer klar. Die Bewegungen sind mit den benutzten Features aus Tabelle 5.1 schwer zu beschreiben: Hier sind keine charakteristischen Muster im gelernten MT vorhanden. Bei den Klassen clap1Reps und kickRFront1Reps gibt es zwar kleine charakteristische Muster, wie in Abbildung 3.11 gezeigt (clap1Reps: Hände bewegen schnell aufeinander zu, ausgedrückt mit den

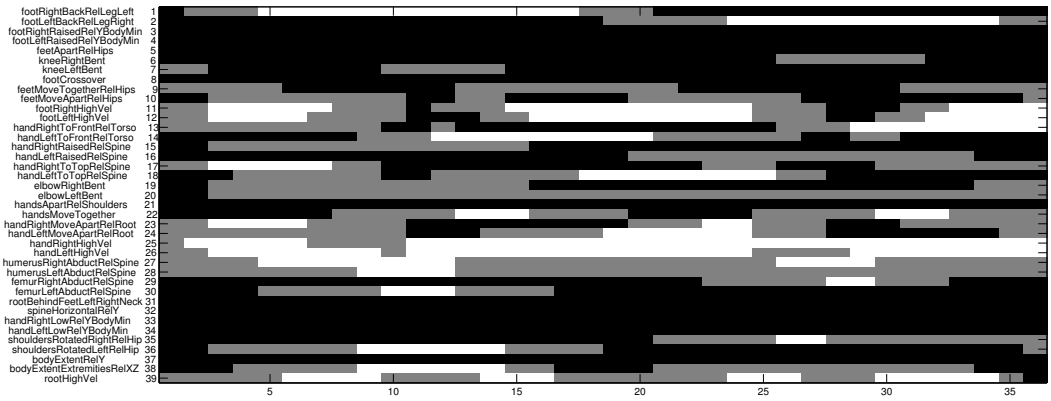


Abbildung 3.12. Hier ist ein aussagekräftiges, quantisiertes ($\delta = 0.1$) weiches MT der Klasse skier1Reps zu sehen. Deutlich sind viele weiße Regionen, welche konsistente charakteristische Eigenschaften der Trainingstemplates anzeigen, zu sehen.

Features $F22$, $F25$, $F26$; kickRFront1Reps: rechter Fuß ist vor dem linken Fuß angehoben und bewegt sich schnell, Oberschenkel ist angezogen, ausgedrückt mit den Features $F2$, $F3$, $F11$ und $F29$), jedoch ist der größte Teil der 39 verwendeten Features sehr inkonsistent innerhalb der Trainingstemplates. Damit sind die MTs insgesamt wenig aussagekräftig, enthalten sehr viele „grau"-Regionen und können somit die Klasse schlecht von anderen Klassen unterscheiden. Man sieht auf den ersten Blick, dass die mit $\delta = 0.1$ quantisierten weichen MTs in Abbildung 3.11 kaum charakteristische, weiße Regionen enthalten. Das kickRFront1Reps-Template enthält zwar mehr weiße Regionen, jedoch haben die inkonsistenten, grauen Bereiche einen sehr hohen Anteil am gesamten MT.

Bei allen anderen Klassen sind die Ergebnisse sehr gut: Den relevanten Dokumenten werden die geringsten Kosten zugewiesen und man kann ohne viele relevante Dokumente zu verlieren über alle Klassen den Schwellwert $\tau = 0.06$ festhalten. Nur bei 15 der 57 Klassen sind relevante Dokumente mit größeren Kosten als $\tau = 0.06$ unter den Treffern. Man verliert bei der Klasse depositFloorR 10 Treffer, bei punchRFront1Reps 5 Treffer, bei punchLSide1Reps 4 Treffer, bei kickRFront1Reps und cartwheelLHandStart1Reps 3 Treffer und bei den anderen 10 der 15 Klassen 2 oder weniger Treffer. Hier sind die mit Hilfe der Trainingsdatenbank gelernten weichen MTs also aussagekräftig für die entsprechenden Bewegungsklassen. Ein quantisiertes ($\delta = 0.1$) weiches MT der Klasse skier1Reps soll hier als positives Beispiel in Abbildung 3.12 dienen. Man erkennt direkt an den großen weißen Regionen, dass viele konsistente, charakteristische Eigenschaften mit den Features dargestellt werden können. Man kann gut die einzelnen Phasen der Bewegung erkennen: Erst befindet sich der rechte Fuß hinter dem linken (Feature $F1$). Dann geht die Bewegung in die Sprungphase über, in welcher beide Füße eine hohe Geschwindigkeit haben ($F11$, $F12$), die linke Hand sich nach vorne und die rechte Hand nach hinten mit hoher Geschwindigkeit bewegen ($F13$, $F14$, $F25$, $F26$). Schließlich wird der Ausfallschritt mit dem linken Fuß hinter dem rechten Fuß erreicht ($F2$). Dementsprechend sind auch die Retrieval-Ergebnisse zu dieser Klasse perfekt: Unter den ersten 15 Treffern sind alle 15 relevanten Dokumente enthalten. Die Retrieval-Ergebnisse der anderen Klassen haben eine ähnlich gute Qualität, wie in der Tabelle 3.1 zu sehen ist.

Ein Nachteil bei der verwendeten Retrieval-Technik (Teilfolgen-DTW, Abschnitt 3.2) ist die Laufzeit, welche linear im Produkt der Eingabegrößen ist: Bei einer Datenbanklänge von N

Frames und einer Anfragelänge von M Frames werden $O(N \cdot M)$ arithmetische Operationen benötigt, um die Suchergebnisse zu erhalten. In der Tabelle A.1 ist beschrieben, dass die Evaluationsdatenbank der D^{57} 146205 Frames bei 120 Hz enthält. Für die Retrieval-Experimente wurde die Abtastrate auf 30 Hz verändert, da eine höhere Abtastrate mit den verwendeten Features kaum Sinn macht und eher irrelevante Details der Featurematrizen verursacht. Somit entspricht dies bei der verwendeten Framerate von 30 Hz einer Eingabelänge von insgesamt 36551 Frames und ca. 20 Minuten Motion-Capture-Daten. Bei einer Anfragelänge von im Mittel 57 Frames benötige der verwendete Computer zur Berechnung der Ergebnisse im Mittel 8 Sekunden. Wenn man nun beachtet, dass die Laufzeit linear mit dem Produkt der Eingabegrößen wächst, so ist leicht vorstellbar, dass die Anfragezeiten bei einer Datenbankgröße von mehreren Stunden praxistaugliche Bereiche schnell übersteigen. Um Anfragen effizienter bearbeiten zu können, werden andere Retrievaltechniken benötigt. Eine mögliche Technik zur Verkürzung der Antwortzeiten wird in Kapitel 4 untersucht. Dort ist die Grundidee, in einem Vorverarbeitungsschritt die Datenbankgröße effizient zu reduzieren. Dabei werden so genannte Keyframes benutzt, welche charakteristische Eigenschaften der Bewegungsklasse kompakt repräsentieren. Diese Keyframes können semi-automatisch für jede Klasse erstellt werden. Ein erster Ansatz zur automatischen Generierung der Keyframes wird kurz in Kapitel 6 behandelt. Hier können die harten MTs als Grundlage dienen, weil sie garantieren, dass jede Spalte in dem harten MT auch in den Trainingsbewegungen vorkommen. So kann die Anzahl der durch Keyframes verursachten False Negatives minimiert werden. Nach diesen Keyframes kann effizient in einem Index gesucht werden. Schließlich kann auf der reduzierten Datenbank mit dem DTW-Algorithmus eine Rangordnung der Treffer hergestellt werden.

Nr.	Bewegungsklasse	γ	$\lvert H_\tau \rvert$ / $\lvert H_\tau^+ \rvert$						K	t^T
1	cartwheelLHandStart1Reps	10	1 1	4 4	5 5	7 7	9 9	10 10	106	4.59
2	clap1Reps	8	41 5	121 6	461 6	500 6	500 6	500 6	11	5.04
3	clapAboveHead1Reps	8	5 5	6 6	17 8	34 8	59 8	85 8	23	3.64
4	depositFloorR	16	3 0	5 1	12 4	14 6	24 13	37 15	73	5.79
5	depositHighR	14	12 8	20 10	25 13	29 13	46 13	110 13	69	7.02
6	elbowToKnee1RepsLelbowStart	13	3 3	7 7	13 13	13 13	15 13	22 13	38	3.04
7	elbowToKnee1RepsRelbowStart	13	8 8	11 11	12 12	13 13	13 13	21 13	38	2.93
8	grabFloorR	8	5 5	7 6	9 7	11 7	18 8	34 8	61	4.80
9	grabHighR	14	16 8	18 8	25 12	29 13	57 14	117 14	69	6.98
10	hopBothLegs1hops	18	12 12	17 15	22 17	28 18	81 18	273 18	24	5.70
11	hopLLeg1hops	20	17 17	18 18	23 20	47 20	87 20	164 20	19	5.37
12	hopRLeg1hops	21	18 18	19 19	21 21	34 21	68 21	117 21	18	5.28
13	jogLeftCircle4StepsRstart	8	8 6	15 7	31 8	35 8	59 8	81 8	63	5.47
14	jogOnPlaceStartFloor2StepsRStart	7	10 3	40 5	227 7	476 7	500 7	500 7	29	7.10
15	jogRightCircle4StepsRstart	8	7 6	14 8	28 8	33 8	50 8	74 8	58	4.79
16	jumpDown	7	3 3	3 3	6 6	6 6	8 6	21 7	65	6.75
17	jumpingJack1Reps	26	23 23	26 26	26 26	26 26	30 26	36 26	35	1.98
18	kickLFront1Reps	16	5 5	13 12	37 15	137 16	271 16	377 16	52	9.14
19	kickLSide1Reps	11	0 0	3 2	11 8	15 9	36 11	102 11	60	7.31
20	kickRFront1Reps	15	4 4	5 5	26 11	83 12	217 13	318 14	53	9.50
21	kickRSide1Reps	14	7 7	11 10	25 12	27 13	57 14	150 14	52	8.46
22	lieDownFloor	10	4 4	5 5	8 8	11 9	15 10	23 10	170	8.54
23	punchLFront1Reps	15	2 1	6 5	18 8	34 13	140 14	319 15	49	8.68
24	punchLSide1Reps	15	5 3	15 9	20 11	37 11	132 13	380 15	42	7.88
25	punchRFront1Reps	15	5 4	6 5	16 8	47 10	200 12	315 12	55	8.86
26	punchRSide1Reps	14	1 1	8 7	31 10	56 14	166 14	373 14	42	7.88
27	rotateArmsBothBackward1Reps	8	7 7	7 7	8 8	8 8	11 8	30 8	29	2.39
28	rotateArmsBothForward1Reps	8	8 8	8 8	8 8	8 8	16 8	46 8	31	2.57
29	rotateArmsLBackward1Reps	8	5 5	6 6	10 8	13 8	34 8	89 8	30	7.60
30	rotateArmsLForward1Reps	8	6 6	6 6	10 8	20 8	49 8	136 8	30	7.90
31	rotateArmsRBackward1Reps	8	6 6	7 7	7 7	25 8	50 8	101 8	28	6.92
32	rotateArmsRForward1Reps	8	6 6	6 6	7 7	30 8	49 8	107 8	27	7.47
33	runOnPlaceStartFloor2StepsRStart	7	7 6	11 6	44 7	144 7	306 7	500 7	22	6.26
34	shuffle2StepsRStart	6	17 5	27 5	61 6	118 6	154 6	197 6	63	8.55
35	sitDownChair	10	4 4	9 8	16 9	25 10	41 10	49 10	84	7.42
36	sitDownFloor	10	8 4	15 6	23 7	34 10	46 10	54 10	106	7.69
37	sitDownKneelTieShoes	8	5 5	7 7	8 8	9 8	13 8	19 8	165	8.68
38	sitDownTable	10	15 9	19 10	49 10	74 10	113 10	165 10	70	9.55
39	skier1RepsLstart	15	12 12	13 13	15 15	18 15	30 15	58 15	36	4.09
40	sneak2StepsLStart	8	4 4	7 5	35 7	68 8	146 8	220 8	58	9.23
41	sneak2StepsRStart	8	6 5	23 6	51 8	110 8	161 8	190 8	63	8.67
42	squat1Reps	26	22 22	24 24	26 26	26 26	26 26	27 26	47	2.55
43	staircaseDown3Rstart	7	12 6	19 6	40 7	94 7	174 7	234 7	52	7.93
44	staircaseUp3Rstart	14	9 9	11 11	15 13	29 13	52 14	113 14	78	6.48
45	standUpLieFloor	10	3 3	7 7	11 8	15 9	18 10	20 10	134	7.01
46	standUpSitFloor	10	9 6	10 6	16 9	19 10	28 10	46 10	96	5.71
47	throwBasketball	7	3 3	5 5	6 6	11 7	24 7	36 7	95	6.24
48	turnLeft	15	32 14	52 15	110 15	190 15	263 15	351 15	49	10.62
49	turnRight	14	33 10	62 13	118 13	179 14	255 14	319 14	51	11.06
50	walk2StepsLstart	15	20 13	31 13	43 15	107 15	205 15	324 15	41	8.92
51	walk2StepsRstart	15	38 13	52 14	64 14	100 14	179 15	300 15	39	8.78
52	walkBackwards2StepsRstart	7	6 6	6 6	12 7	74 7	134 7	189 7	56	7.98
53	walkLeft2Steps	8	8 8	8 8	9 8	19 8	43 8	82 8	80	6.11
54	walkLeftCircle4StepsRstart	9	9 7	14 8	16 9	28 9	61 9	108 9	82	5.77
55	walkOnPlace2StepsLStart	7	62 7	98 7	191 7	306 7	438 7	500 7	39	9.63
56	walkRightCircle4StepsRstart	7	6 5	11 6	15 7	33 7	65 7	113 7	84	5.76
57	walkRightCrossFront2Steps	8	7 7	14 7	42 8	90 8	120 8	146 8	83	6.83

Tabelle 3.1. Retrieval mit gelernten weichen Motiontemplates auf der Evaluationsdatenbank der $\mathcal{D}^{57}$ (Länge: 36551 Frames bei 30 Hz, ca. 20 Minuten Motion-Capture-Daten). Spalte γ: Anzahl der relevanten Dokumente für jede Klasse. $|H_\tau|/|H_\tau^+|$: Anzahl der Treffer / Anzahl der relevanten Treffer bei $\tau = 0.01, 0.02, 0.04, 0.06, 0.08$ und 0.1 von links nach rechts. Es werden maximal 500 Treffer betrachtet. K: Länge des MTs in Frames. t^T: Laufzeit des MT-Retrievals in Sekunden.

Kapitel 4

Keyframebasierte Suche

Wie bereits im vorhergehenden Kapitel erwähnt übersteigen die Laufzeiten bei einer Anfrage an die Motion-Capture-Datenbank mit klassischem DTW praxistaugliche Bereiche. Bei M Frames in der Datenbank und N Frames in der Anfrage ist die theoretische Laufzeit in $O(N \cdot M)$ und auch die praktische Laufzeit ist oft sehr lang. In Szenarien mit Datenbankgrößen von mehreren Stunden kann die Berechnung auf einem handelsüblichen Desktop-PC mehrere Minuten in Anspruch nehmen. Mit der Beobachtung, dass eine Bewegungsklasse oft durch wenige charakteristische Posen von den meisten Klassen unterschieden werden kann, ergibt sich eine Idee zur Beschleunigung der Berechnung. Mit Hilfe von Keyframes, welche die charakteristischen Posen beschreiben, wird eine Vorauswahl von Treffern effizient mit Hilfe eines Indexes bestimmt. Auf dieser Vorauswahl, die typischerweise nur noch einen Bruchteil der Datenbankgröße enthält, kann in einem anschließenden Schritt mit dem in Kapitel 3.4 vorgestellten Verfahren ein Ranking der Treffer durchgeführt werden. Ist die Vorauswahl entsprechend klein, so trägt die Tatsache, dass die Laufzeit des Rankings linear im Produkt der Eingabelängen ist, kaum zur Gesamtlaufzeit bei.

In diesem Kapitel wird ein Algorithmus vorgestellt, der effizient mit Hilfe eines Indexes und einer Menge von Keyframes nach Treffern suchen kann. Grundlagen zu den verwendeten Techniken werden in dem Abschnitt 4.1 behandelt. Abschnitt 4.2 erklärt den entwickelten Algorithmus zur keyframebasierten Suche im Detail. Schließlich wird in Abschnitt 4.3 ein früherer Ansatz zur keyframebasierten Suche vorgestellt und ein Vergleich der Ansätze in 4.4 durchgeführt.

4.1 Grundlagen der keyframebasierten Suche

Die keyframebasierte Suche basiert auf einer Technik, die einem Index in einem Buch ähnlich ist. Die so genannten invertierten Listen speichern für jeden möglichen Featurevektor die Positionen, an denen dieser in der Datenbank vorkommt. Zunächst wird die Notation festgelegt, wie sie in Kapitel 4 durchgehend verwendet wird. Anschließend wird die Technik der invertierten Listen erklärt und ihre Anwendung mit Keyframes dargelegt.

4.1.1 Notation

Die Notation sei wie folgt festgelegt: Die Datenbank $\mathcal{D}$ besteht aus einer Dokumentenkollektion $\mathcal{D} = (D_1, D_2, \ldots, D_I)$, wobei $D_i, i \in [1 : I]$ einen Motion-Capture-Datenstrom darstellt.

Um die Notation zu vereinfachen, nimmt man an, dass $\mathcal{D}$ nur aus einem einzigen Dokument $D = [1 : T] \to \mathcal{P}$ besteht, welches durch Konkatenation der Dokumente $D_i, i \in [1 : I]$, entsteht. Wenn man die Grenzen der Dokumente abspeichert, lassen sich aus D leicht die einzelnen Dokumente wiederherstellen. Dadurch muss nicht mehr zwischen der Datenbank $\mathcal{D}$ und einem Dokument D unterschieden werden. Die Anfrage Q sei ein kurzes Motion-Capture-Dokument.

Die Featurefunktion $F : \mathcal{P} \to \{0,1\}^f$ sei festgelegt und die extrahierten Features seien $F[D] = \vec{w} = (w_0, w_1, \ldots, w_M)$ und $F[Q] = \vec{v} = (v_0, v_1, \ldots, v_N)$. Man beachte, dass hier Darstellung der Features wie in Abschnitt 2.4 beschrieben segmentbasiert ist. Es gilt also $M \leq T$. Wenn es nur um die extrahierten Features geht, wird einfach von der Anfrage $\vec{v}$ und der Datenbank $\vec{w}$ gesprochen.

In dem Dokument D lässt sich mit Hilfe eines Parameters t die Pose $P \in \mathcal{P}$ zum Zeitpunkt t angeben: $P = D(t)$. In der gleichen Weise kann man die Features des Dokumentes $\vec{w} = F[D]$ angeben: $\vec{w}(m) = w_m$ mit $m \in [1 : M], w_m \in \{0,1\}^f$.

4.1.2 Invertierte Listen

Die Repräsentation eines Dokumentes als eine Folge von Posen bzw. Featurevektoren erlaubt einen wahlfreien Zugriff auf die Posen bzw. Featurevektoren jedes Zeitpunktes und ist somit gut geeignet, um einen Ausschnitt der Datenbank zu betrachten, zu vergleichen oder zu modifizieren. Möchte man jedoch wissen, zu welchen Zeitpunkten t ein Featurevektor v auftritt, so ist diese Repräsentation nicht gut geeignet, da im schlimmsten Fall jeder einzelne Featurevektor der Datenbank mit v verglichen werden muss. Genau diese Suche nach den Zeitpunkten des Auftretens eines Featurevektors wird bei dem vorgestellten Algorithmus zur keyframebasierten Suche benötigt: Es werden die Positionen von bestimmen Featurevektoren, die charakteristisch für eine Bewegung sind, benötigt.

Um diese Suche effizient lösen zu können, kann man die Datenbank in eine andere Repräsentation überführen, in welcher diese Suche durch einen einfachen Zugriff geschehen kann. Diese Repräsentation sind die invertierten Listen. Für jeden Featurevektor $v \in \{0,1\}^f$ wird eine invertierte Liste $L(v)$ angelegt. $L(v)$ enthält alle Positionen in dem Dokument, an welchen der Featurevektor v vorkommt. Dadurch können mit einem Zugriff auf die entsprechende invertierte Liste alle Positionen eines Featurevektors in der Datenbank gefunden werden. Man beachte, dass diese Repräsentation zwar gut geeignet ist um Positionen von Featurevektoren zu suchen, jedoch ist sie schlecht geeignet für Anwendungen wie das Darstellen eines Ausschnittes des Dokumentes.

Erstellt man für jeden möglichen Featurevektor eine invertierte Liste, so erhält man den Index $I_F^{\mathcal{D}}$. Dieser Index enthält 2^f invertierte Listen $L(v)$, da es 2^f verschieden Vektoren $v \in \{0,1\}^f$ gibt. Man erkennt, dass die Anzahl der invertierten Listen exponentiell mit der Anzahl der ausgewählten Features wächst. Somit ist es nicht praktikabel, einen Index für eine große Featuremenge zu berechnet. Da jeder Frame aus D genau einmal im Index $I_F^{\mathcal{D}}$ vorkommt, ist die Größe des Indexes proportional zu der Länge des Dokumentes, wobei der Verwaltungsaufwand für die Listen hier nicht berücksichtigt wird.

Wie bereits in Abschnitt 2.4 erwähnt wird die Datenbank $\vec{w}$ F-segmentiert. Daher werden auch in dem Index $I_F^{\mathcal{D}}$ nicht die Framepositionen, sondern die Segmentpositionen bezüglich der F-Segmentierung des Dokumentes gespeichert. Die Erstellung der invertierten Listen geschieht dann in einem Algorithmus mit der Zeitkomplexität $O(M)$, wobei M die Anzahl der Segmente der Datenbank $\vec{w}$ bezeichnet. Auch die Größe des Indexes ist nun proportional zu M. Die Framepositionen aller Segmente können mit Hilfe der gespeicherten Längen der Segmente rekonstruiert werden. Die invertierten Listen $L(v)$ können nun sowohl die Framepositionen und -längen der Segment, als auch die Segmentposition der F-Segmentierung von D zurückgeben. Der hier vorgestellte Algorithmus benötigt die Framepositionen der Segmente, da die Abstände der Segmente für die Berechnungen benötigt werden, um Treffer zu finden: Bestimmte Featurevektoren in einem Treffer müssen Abstandsbedingungen genügen, welche nur berechnet werden können, wenn die Framepositionen der Segmente bekannt sind. Daher wird angenommen, dass die invertierten Listen $L(v)$ Segmente gemäß der folgenden Definition enthalten:

Definition 4.1. *Ein Segment ist ein Element der Menge*

$$S := \{(s,t) \in (\mathbb{Z} \cup -\infty) \times (\mathbb{Z} \cup \infty) : s \leq t\} \cup \{\emptyset\}, \tag{4.1}$$

wobei s die Startposition und t die Endposition des Segmentes bezeichnet. Die Länge von S in Frames gemessen ist $t - s + 1$. Für $s = -\infty$ oder $t = \infty$ ist die Länge des Segmentes ∞ und für $S = \emptyset$ ist die Länge 0. Ist $S = (s,t)$ gegeben, so wird $S.\mathrm{start}$ und $S.\mathrm{end}$ definiert durch

$$S.\mathrm{start} = s$$
$$S.\mathrm{end} = t$$

Wenn die invertierten Listen nicht in dieser Form sind, sondern nur die Segmentposition bezüglich der F-Segmentierung enthalten, so kann man sie mit einem Algorithmus der Zeitkomplexität $O(M)$ in diese Form bringen, in dem die gespeicherten Längen der Segmente benutzt werden, um die Framepositionen zu rekonstruieren.

Der Schnitt zweier Segmente ist durch die folgende Definition gegeben.

Definition 4.2. *Der Schnitt zweier Segmente $(s_1,t_1) \in S$ und $(s_2,t_2) \in S$ ist definiert durch*

$$(s_1,t_1) \cap (s_2,t_2) := \begin{cases} \emptyset, & \text{falls } t_1 < s_2 \text{ oder } t_2 < s_1 \\ (\max(s_1,s_2), \min(t_1,t_2)), & \text{sonst} \end{cases} \tag{4.2}$$

Man beachte, dass der Schnitt zweier Segmente immer noch in S liegt, S ist also abgeschlossen unter der Schnitt-Operation.

Invertierte Listen von Keyframes. Ein für einen bestimmten Zweck besonders ausgezeichneter Frame in einem Motion-Capture-Datenstrom wird oft als Keyframe bezeichnet. Hier werden Featurevektoren, die für eine Bewegung charakteristisch sind, Keyframes genannt. Diese Keyframes werden in dem vorgestellten Algorithmus benutzt, um eine Treffermenge zu bilden. Die Keyframes können semi-automatisch, nach Heuristiken oder mit Lernverfahren ausgewählt werden.

Oft ist es sinnvoll, nicht nur einen einzelnen Keyframevektor zu bestimmen, sondern eine Menge von alternativen Keyframevektoren zuzulassen. Somit sei das Konzept des Keyframes durch die folgende Definition festgelegt.

Definition 4.3. *Ein Keyframe V ist eine Menge von Featurevektoren:*

$$V \subseteq \{0,1\}^f \tag{4.3}$$

Der vorgestellte Algorithmus benötigt die invertierten Listen der Keyframes, um effizient die Positionen finden zu können, an welchen die Keyframes in dem Dokument D auftreten. Bisher wurde nur das Konzept einer invertierten Liste für einen bestimmten Featurevektor vorgestellt. Wird ein Keyframe durch einen bestimmten Featurevektor repräsentiert, so ist die dazugehörige invertierte Liste in dem Index $I_f^{\mathcal{D}}$ zu finden. Um eine invertierte Liste für eine Menge von Featurevektoren V zu erhalten, genügt es, die invertierten Listen aller enthaltenen Featurevektoren zu vereinigen:

$$\begin{aligned}\Lambda(V) &= \bigcup_{v \in V} L(v) \\ &= ((s_{V,1}, t_{V,1}), \ldots, (s_{V,\ell_V}, t_{v,\ell_V})) \in S^{\ell_V}\end{aligned}$$

Die Länge der Liste $\Lambda(V)$ ist mit ℓ_V bezeichnet. Die Vereinigung kann mit einem linearen Merge der Zeitkomplexität $O(\sum_{v \in V} l_v)$ geschehen, wenn die invertierten Listen lexikographisch sortiert sind. Man beachte, dass durch die Konstruktion der invertierten Listen die zu vereinigenden Segmente paarweise disjunkt sind, somit also keine Überlappungen einzelner Segmente berücksichtigt werden müssen. Somit zeigt sich, dass die invertierten Listen von Keyframes, die durch eine Menge von Featurevektoren dargestellt werden, effizient mit Hilfe des Indexes $I_f^{\mathcal{D}}$ berechnet werden können.

4.2 Algorithmus zur keyframebasierten Suche

In diesem Abschnitt wird ein Algorithmus zur keyframebasierten Suche vorgestellt. Der Algorithmus ist in der Lage, Keyframes unter Berücksichtigung vorgegebener zeitlicher Abstandsbereiche in der Datenbank anzufragen. Dabei werden ähnlich wie bei DTW zeitliche Verschiebungen der Keyframes in einem definierbaren Rahmen erlaubt. Zuerst werden grundlegende Definitionen gegeben und der Algorithmus vorgestellt. In Abschnitt 4.2.1 wird dann ein Beispiel gegeben, in dem die Funktionsweise des Algorithmus nachvollzogen wird. Schließlich wird der Korrektheitsbeweis in 4.2.2 und die Analyse der Laufzeit in Abschnitt 4.2.3 gegeben. Das Kapitel wird mit einer Bemerkung in 4.2.4 geschlossen, wie in der Praxis die Laufzeit des Algorithmus verbessert werden kann.

Der Algorithmus bekommt als Eingabe eine Sequenz von Keyframes $(V_1 \ldots, V_K)$ und die geforderten Frameabstände $\delta_1, \ldots, \delta_{K-1}$ der Keyframes. Des Weiteren benötigt er Zugriff auf die invertierten Listen $L(v)$ aller Featurevektoren. Er berechnet nun alle Abschnitte der Datenbank, welche die Keyframes mit den richtigen Abständen enthalten. Was dabei ein „richtiger“ Abstand ist, kann mit einem Parameter beeinflusst werden. Dieser Parameter bestimmt, inwieweit die Abstände d_i der Keyframes in der Datenbank von den Abständen δ_i

abweichen dürfen. Man stellt sich benachbarte Keyframes wie mit einer Art Feder verbunden vor. Wie steif diese Federn sind, wird durch den Parameter $\sigma \in [0,1]$, hier mit Stiffness benannt, bestimmt. Setzt man $\sigma = 1$, so bedeutet dies, dass die Federn absolut hart sind und keine Variation der Abstände erlauben. Für $\sigma = 0$ erlauben die Federn jegliche Variation der Abstände. Es sei daran erinnert, dass $\delta_1, \ldots, \delta_{K-1}$ die geforderten Frameabstände der Keyframes bezeichnen und $d_1, \ldots, d_{K-1}$ die tatsächlichen Abstände der Keyframes in der Datenbank. Dann wird die erlaubte Variation der Abstände modelliert durch

$$\sigma \cdot \delta_k \leq d_k \leq \frac{1}{\sigma} \cdot \delta_k \tag{4.4}$$

für $k \in [1 : K-1]$. Für $\sigma = 0$ wird $\frac{1}{\sigma} = \infty$ angenommen, in diesem Fall existiert keine obere Schranke für die erlaubte Variation des Abstandes. Mit Hilfe dieser Vereinbarung über die geforderten Variationsbereiche der Keyframes kann man definieren, was das Ziel der Berechnungen sein soll:

Definition 4.4. *Es sei $(i_1, \ldots, i_K)$ eine aufsteigende Folge von Frameposition mit $1 \leq i_1 \leq \ldots \leq i_K \leq M$. $H = (i_1, \ldots, i_K)$ ist ein Hit in D bezüglich der Keyframes $(V_1 \ldots, V_K)$, der Abstände der Keyframes $\delta_1, \ldots, \delta_{K-1}$ und des Stiffness-Faktors σ, wenn die beiden folgenden Bedingungen erfüllt sind:*

$$1.) \quad \forall k \in [1 : K] : \vec{w}(i_k) \in V_k \tag{4.5}$$

$$2.) \quad \forall k \in [1 : K-1] : \sigma \cdot \delta_k \leq i_{k+1} - i_k \leq \frac{1}{\sigma} \cdot \delta_k \tag{4.6}$$

Ein Hit ist also eine Sequenz von Frames, so dass jeder Frame zu dem entsprechenden Keyframe passt und die Abstände der Frames den Abständen der Keyframes gleichen. Dabei kann mit σ eingestellt werden, inwieweit die tatsächlichen Abstände von den geforderten Abständen abweichen dürfen.

Der Algorithmus arbeitet mit Zeigern $p_1, \ldots, p_K$, welche die aktuellen Positionen in den invertierten Listen anzeigen. Beispielsweise bedeutet $p_1 = 5$, dass der Zeiger für den ersten Keyframe p_1 auf das fünfte Element der invertierten Liste des ersten Keyframes zeigt. Da die invertierten Listen aus den Segmenten der Datenbank bestehen, zeigt ein solcher Zeiger also auf ein Segment in der Datenbank (also einen Bereich von Frames) und im Allgemeinen nicht auf einen einzelnen Frame. Bezüglich dieser Zeiger wird nun ein Match definiert:

Definition 4.5. *Es seien K Keyframes $(V_1, \ldots, V_K)$ und die invertierten Listen $\Lambda_1 := \Lambda(V_1), \ldots, \Lambda_K := \Lambda(V_K)$ gegeben. Eine invertierte Liste Λ_k sei durch die Notation $\Lambda_k = ((s_{k,1}, t_{k,1}), \ldots, (s_{k,\ell_k}, t_{k,\ell_k}))$ festgelegt. Die Längen der invertierten Listen (also die Anzahl der enthaltenen Segmente) werden mit $\ell_1, \ldots, \ell_K$ angegeben. Eine Vektor von Zeigern $p = (p_1, \ldots, p_K)$ mit $p_k \in [1 : \ell_k]$ für $k \in [1 : K]$ ist ein Match, wenn $\exists i_1, \ldots, i_K$ mit*

$$\begin{aligned} s_{1,p_1} \leq i_1 \leq t_{1,p_1} \quad & \wedge \\ s_{2,p_2} \leq i_2 \leq t_{2,p_2} \quad & \wedge \\ \ldots \quad & \wedge \\ s_{K,p_K} \leq i_K \leq t_{K,p_K} & \end{aligned}$$

so, dass $(i_1, \ldots, i_K)$ ein Hit ist.

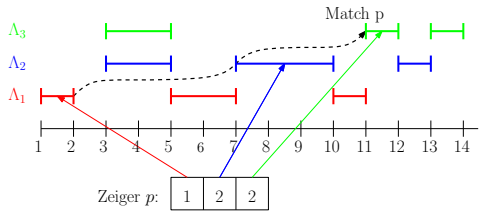


Abbildung 4.1. Zeiger zeigen in die invertierten Listen, welche die Segmente der Datenbank enthalten. Ein Match $p = (1, 2, 2)$ wird gefunden.

Mit anderen Worten beschreibt ein Match ein K-Tupel von Segmenten, so dass Frames in den Segmenten gefunden werden können, für welche die Hit-Definition gilt. Ein Match ist also ein Konstrukt eines Treffers auf Segment-Ebene und ein Hit ein Treffer auf Frame-Ebene. Es ist klar, dass es zu jedem Hit genau einen Match gibt, da jeder Frame in genau einem Segment enthalten ist. Umgekehrt kann es zu einem Match mehr als einen Hit geben, wenn es innerhalb der Segmente verschiedene Frames gibt, für welche die Abstandsbedingung gilt. Der Unterschied zwischen einem Hit und einem Match wird noch einmal anhand Abbildung 4.1 erklärt. Auf der X-Achse sind die Framepositionen der Bewegungsdaten aufgezeichnet. Parallel zu der X-Achse sind die invertierten Listen der Keyframes V_1, V_2 und V_3 veranschaulicht. Die invertierten Listen enthalten jeweils 3 Segmente:

$$\Lambda_1 = ((1, 2),\ (5, 7),\ (10, 11))$$
$$\Lambda_2 = ((3, 5),\ (7, 10),\ (12, 13))$$
$$\Lambda_3 = ((3, 5),\ (11, 12),\ (13, 14))$$

Nun wird beispielsweise der mit einer gestrichelten Linie eingezeichnete Match gefunden. Dieser Match p ist das K-Tupel $p = (1, 2, 2)$, da wie im Bild angedeutet der Zeiger auf die entsprechenden Segmente der invertierten Listen zeigt. Zu diesem Match kann es nun mehrere Hits geben, sofern es die Abstände der Keyframes und die Stiffness-Bedingung erlauben. Ein möglicher Hit wäre nun $H_1 = (2, 7, 11)$: Die angegebenen Framepositionen liegen innerhalb der Segmente des Matches. Ein weiterer Hit könnte $H_2 = (1, 7, 12)$ sein, sofern diese Frames der Abstandsbedingung genügen.

Der vorgestellte Algorithmus berechnet Matches. Wie aus diesen Matches die geforderten Hits berechnet werden können, wird im konstruktiven Korrektheitsbeweis dargestellt. In der Praxis wird der vom Match überspannte Hit-relevante Bereich gesucht. Das ist der Bereich, der durch alle zum Match passenden Hits überdeckt wird. So ist sichergestellt, dass der komplette Bereich, der semantisch zu den Keyframes passt, vom Algorithmus zurückgegeben wird. Mit der Motivation, dass mit der keyframebasierten Suche eine grobe Auswahl von Treffern gefunden werden soll, die möglichst keine relevanten Dokumente verlieren soll, ist die Überlegung sinnvoll.

Definition 4.6. *Sei M ein Match und $H_1, \ldots, H_N$ alle zum Match passenden Hits mit $\forall n \in [1 : N] : H_n = (i_{n,1}, i_{n,2}, \ldots, i_{n,K})$. Damit ist der von einem Match überspannte Hit-relevante Bereich R gegeben durch*

$$R = R(M) = (\min_{n \in [1:N]} i_{n,1},\ \max_{n \in [1:N]} i_{n,K}) \in S$$

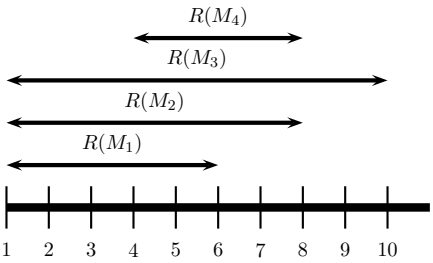


Abbildung 4.2. Der von den Matches 1 bis 4 überspannte Hit-relevante Bereich $(1, 10)$ wird wird allein durch den Bereich von Match 3 bestimmt. Die Matches 1 bis 4 liegen in einer Zusammenhangskomponente nach Definition 4.7.

Der folgende Algorithmus berechnet nicht alle möglichen Matches. Es kann aber garantiert werden, dass die Matches, die nicht gefunden werden, auch nicht relevant für das praktische Ergebnis sind. Die Bedeutung des Ausdrucks „nicht relevant“ wird im Folgenden näher erklärt. In der Praxis ist der größte zusammenhängende Bereich gesucht, der von Matches überspannt wird. Dies ist in Abbildung 4.2 dargestellt: Mit einem Pfeil ist dort jeweils der von einem Match überspannte Hit-relevante Bereich dargestellt. Der größte von allen Matches überspannte Hit-relevante Bereich ist der Bereich $(1, 10)$. Um diesen Bereich zu erhalten, würde es genügen, den Match mit der Nummer 3 zu finden, die anderen Matches sind für das Gesamtergebnis nicht relevant. Dieses Konzept wird durch so genannte Zusammenhangskomponenten wie folgt modelliert:

Definition 4.7. *Sei $M = (M_1, M_2, \ldots, M_N)$ eine lexikographisch aufsteigend sortierte Menge von Matches wie von Algorithmus 4.1 zurückgeliefert. Es seien $\forall i \in [1 : N] : R(M_i) = (s_i, t_i) \in S$ die von den Matches M_i überspannten Hit-relevanten Bereiche. Dann ist eine Zusammenhangskomponente (ZHK) Z_z^{z+L} der Länge $(L + 1)$ in M definiert durch $Z_z^{z+L} = (M_z, M_{z+1}, \ldots, M_{z+L})$ so, dass für Z_z^{z+L} die Bedingung*

$$\forall k \in [1 : L] \exists i_k \in [0 : k - 1], \text{ so dass } s_{z+k} \leq t_{z+i_k} \tag{4.7}$$

gilt und Z_z^{z+L} weder durch M_{z-1} noch durch M_{z+L+1} erweiterbar ist, ohne die Bedingung (4.7) zu verletzen.

Lemma 4.1. *Eine Menge von Matches $M = (M_1, M_2, \ldots, M_N)$, lexikographisch aufsteigend sortiert, kann in paarweise disjunkte Zusammenhangskomponenten zerlegt werden durch*

$$M^Z := (Z_1^{N_1}, Z_{N_1+1}^{N_2}, Z_{N_2+1}^{N_3}, \ldots, Z_{N_m+1}^{N}).$$

Nun wird ein Operator definiert, der eine Menge von Matches in Zusammenhangskomponenten zerlegt. Dieser Operator bildet die Zusammenhangskomponenten dann auf die von den einzelnen Komponenten überspannten Hit-relevanten Bereiche ab.

Definition 4.8. *Sei $M = (M_1, M_2, \ldots, M_N)$ eine Menge von Matches wie von Algorithmus 4.1 zurückgeliefert. Es seien $\forall i \in [1 : N] : R(M_i) = (s_i, t_i) \in S$ die von den Matches M_i überspannten Hit-relevanten Bereiche lexikographisch aufsteigend sortiert. Es seien $M^Z = (Z_1^{N_1}, Z_{N_1+1}^{N_2}, Z_{N_2+1}^{N_3}, \ldots, Z_{N_m+1}^{N})$ die Zusammenhangskomponenten von M. Dann ist*

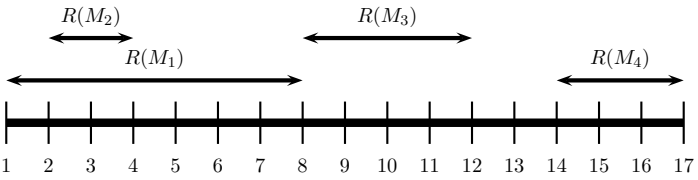


Abbildung 4.3. Die Matches 1 bis 3 liegen in einer Zusammenhangskomponente. Der Match 4 liegt in einer weiteren Zusammenhangskomponente.

Ω^Z *definiert durch*

$$\Omega^Z(M) = \big((s_1, \max_{j\in[1:N_1]} t_j), (s_{N_1+1}, \max_{j\in[N_1+1:N_2]} t_j), \\ (s_{N_2+1}, \max_{j\in[N_2+1:N_3]} t_j), \ldots, (s_{N_m+1}, \max_{j\in[N_m+1:N]} t_j)\big)$$

Beispiel. Die Bedingung (4.7) in Definition 4.7 bedeutet nichts anderes, als dass sich die überspannten Hit-relevanten Bereiche der Matches in der Zusammenhangskomponente alle überlappen: Für jeden Match P muss es einen anderen Match Q geben, dessen Endframe erst nach dem Startframe von P vorkommt. Dies wird auch durch Abbildung 4.3 verdeutlicht. Hier werden die überspannten Hit-relevanten Bereiche $R(M_1)$, $R(M_2)$, $R(M_3)$ und $R(M_4)$ jedes Treffers der Treffermenge $M = (M_1, M_2, M_3, M_4)$ gezeigt. Diese Matches können in Zusammenhangskomponenten zerlegt werden: $Z^M = (Z_1^3, Z_4^4)$. Die Matches 1 bis 3 liegen in einer Zusammenhangskomponente: Der Start von M_2 liegt vor dem Ende von M_1 und der Start von M_3 liegt vor dem Ende von M_1. Der Start von M_4 liegt jedoch hinter dem Ende von M_1, M_2 und M_3. Daher kann die Zusammenhangskomponente nicht erweitert werden. M_4 bildet somit eine eigene Zusammenhangskomponente. Der Operator Ω^Z hat folgendes Ergebnis:

$$\Omega^Z(M) = \big((1, 12), (14, 17)\big).$$

Man beachte, dass der Match M_2 keinen Einfluss auf das Ergebnis von Ω^Z hat. Beim Weglassen von M_2 überspannen die Zusammenhangskomponenten den selben Bereich:

$$\Omega^Z(M) = \Omega^Z(M \setminus \{M_2\}).$$

M_2 ist in diesem Fall ein nicht relevanter Match, da er auf das Ergebnis von Ω^Z keinen Einfluss hat. Sei nun M^{Complete} die Menge aller möglichen Matches. Für die vom Algorithmus gefundene Menge von Matches M muss garantiert werden, dass

$$\Omega^Z(M^{\text{Complete}}) = \Omega^Z(M).$$

Dies ist wichtig, da die Bereiche von $\Omega^Z(M^{\text{Complete}})$ von Interesse sind: Das sind alle Bereiche, die zu den Keyframes passen. Der Algorithmus muss also nicht alle Treffer finden, sondern nur diejenigen, so dass die Ω^Z-Bereiche korrekt sind. Das bedeutet, dass für nicht relevante Treffer nicht garantiert werden muss, dass der Algorithmus sie findet.

In dem Algorithmus wird eine Berechnung benötigt, die ausgehend von einem aktuell betrachteten Segment $(s, t) \in S$ den Bereich bestimmt, in dem der nächste Keyframe liegen muss, um der Abstandsbedingung eines Hits (Definition 4.4) zu genügen. Dieser Bereich wird als Segment (Definition 4.1) modelliert und mit der wie folgt definierten Funktion μ_k berechnet.

Definition 4.9. *Die geforderten Abstände der Keyframes $V_1, \ldots, V_K$ seien mit $\delta_1, \ldots, \delta_{K-1}$ gegeben. Der Stiffness-Parameter sei mit $0 \leq \sigma \leq 1$ gegeben. Dann ist die Funktion μ_k gegeben durch*

$$\begin{aligned} \mu_k &:= \mu_{\sigma,\delta_k} : S \to S \\ (s,t) &\mapsto \begin{cases} (s + \lceil \sigma \cdot \delta_k \rceil, t + \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor), & \sigma > 0 \\ (s, \infty), & \sigma = 0 \end{cases} \\ \emptyset &\mapsto \emptyset \end{aligned} \tag{4.8}$$

Für den Fall $\sigma = 0$ wird auf ein Segment abgebildet, dass nach rechts beliebig weit ausgedehnt ist.

Bemerkung 4.1. In dem vorgestellten Algorithmus wird der Stiffness-Parameter σ benutzt. Dieser gibt an, wie stark die tatsächlichen Abstände der Keyframes von den vorgegebenen Abständen abweichen dürfen. Um weitere Flexibilität erlauben zu können, kann man für je zwei nebeneinander liegende Keyframes unterschiedliche Stiffness-Werte erlauben. Dann wird der Parameter σ zu einem vom Keyframe k abhängigen Parameter σ_k. Der Algorithmus bleibt mit dieser Änderung im Wesentlichen gleich, der Korrektheitsbeweis und die Laufzeitbetrachtung ebenfalls. Insbesondere ändert sich die angegebenen Worst-Case-Laufzeit nicht. In den praktischen Experimenten wird diese vom Keyframe abhängige Stiffness benutzt. Die semi-automatisch erstellten Keyframes (siehe Anhang B) setzen für je zwei Keyframes unterschiedliche Stiffness-Werte ein, damit die Keyframes eine Klasse von Bewegungen besser beschreiben können. Besonders wichtig wird diese Abhängigkeit im Kapitel 5.4.2, in welchem nach einer Abfolge von Bewegungsklassen A und B gesucht wird. Innerhalb der Klassen kann die Stiffness relativ hoch sein, um die Klassen genau zu beschreiben. Möchte man ausdrücken, dass zwischen den Bewegungen eine stark variable Zeitspanne vergehen darf, so benötigt man einen niedrigen Stiffnesswert zwischen den Bewegungsklassen, um dies ausdrücken zu können.

Der nun vorgestellte Algorithmus arbeitet rekursiv. Der Rekursionsstart wird durch Algorithmus 4.1 und die eigentliche Rekursion durch Algorithmus 4.2 beschrieben. Im Algorithmus 4.2 wird in den Zeilen 4, 13 und 15 verursacht, dass der Zeiger der invertierten Listen auch stehen bleiben kann und nach einer Betrachtung nicht immer nur inkrementiert wird. Ohne diese Zeilen würden nicht alle relevanten Matches in der Datenbank gefunden, wie im folgenden Beispiel erläutert wird.

4.2.1 Beispiel

Das folgende Beispiel verdeutlicht die Funktionsweise des Algorithmus. Es seien $K = 3$ Keyframes gegeben. Die Distanzen der Keyframes seien wie in Abbildung 4.4 dargestellt

$$\begin{aligned} \delta_1 &= V_2 - V_1 = 6 \\ \delta_2 &= V_3 - V_2 = 10 \end{aligned}$$

Algorithmus 4.1 : Keyframebasierte Suche

Input : Keyframes $V_1 \ldots, V_K$ mit $V_k \subseteq \{0,1\}^f$ mit $K \geq 2$.
Frame-Abstände der Keyframes $\delta_1, ..., \delta_{K-1}$.
Invertierte Listen $L(v)$ der segmentierten Datenbank für jeden Featurevektor $v \in \{0,1\}^f$.
Stiffness-Parameter $0 \leq \sigma \leq 1$.

Output : Sei M^{Complete} die Menge aller möglichen Matches. Es wird eine Menge M von Matches (Definition 4.5) zurückgegeben, so dass $\Omega^Z(M^{\text{Complete}}) = \Omega^Z(M)$ (siehe Definition 4.8) gilt.

Global : Modelliere den Zeigervektor $p = (p_1, \ldots, p_K)$ in die Keyframelisten als globale Variable, die in Algorithmus 4.1 und Algorithmus 4.2 sichtbar ist.

```
1  begin
       /* Berechnung der invertierten Listen der Keyframes                    */
2      forall k ∈ [1 : K] do  Λ_k ← ⋃_{v∈V_k} L(v)
3      forall k ∈ [1 : K] do p_k ← 1; /* Zeiger in Λ_k initialisieren          */
4      forall k ∈ [1 : K] do ℓ_k ← Länge von Λ_k;
5      for i ← 1 to ℓ_1 do
6          p_1 ← i
           /* Berechne den erlaubten Bereich für den zweiten Keyframe         */
7          allowedRange ← μ_1(Λ_1(p_1))
           /* Starte Rekursion, um Segmente in allowedRange zu finden         */
8          sucheRekursiv(2, allowedRange)
9  end
```

Algorithmus 4.2 : sucheRekursiv(k, allowedRange)

Input : Die Nummer des aktuell betrachteten Keyframes $k \in [1 : K]$. Das Segment allowedRange $\in S$ gibt den Bereich an, in dem ein Auftreten von V_k gesucht werden soll.

Global : Modelliere $p = (p_1, \ldots, p_K)$ als globale Variable, die in Algorithmus 4.1 und Algorithmus 4.2 sichtbar ist.

```
1  begin
       /* p_k in den allowedRange vorspulen                                   */
2      while  p_k ≤ ℓ_k ∧ Λ_k(p_k).end < allowedRange.start  do
3          p_k ← p_k + 1
       /* Suche Segmente innerhalb allowedRange und gehe in die nächtes Rekursion. */
4      pointerIncremented ← false
5      intersection ← Λ_k(p_k) ∩ allowedRange
6      while intersection ≠ ∅ do
7          if k = K then
               /* Rekursion ist hier zu Ende. p zeigt nun auf die Segmente in der Datenbank,
                  die einen Match darstellen. Dieser Match wird nun in einer geeigneten
                  Datenstruktur gespeichert, beispielsweise einer globalen Variable, die
                  eine Listendatenstruktur repräsentiert.                      */
8              matches ← matches ∪ p
9          else
10             newAllowedRange ← μ_k(intersection)
11             sucheRekursiv(k + 1, newAllowedRange)
12         p_k ← p_k + 1
13         pointerIncremented ← true
14         if p_k > ℓ_k then intersection ← ∅ else intersection ← Λ_k(p_k) ∩ allowedRange
15     if pointerIncremented then p_k ← p_k − 1
16 end
```

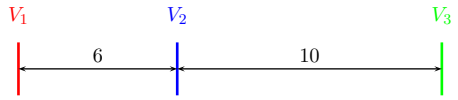


Abbildung 4.4. Abstände der drei Keyframes V_1, V_2 und V_3.

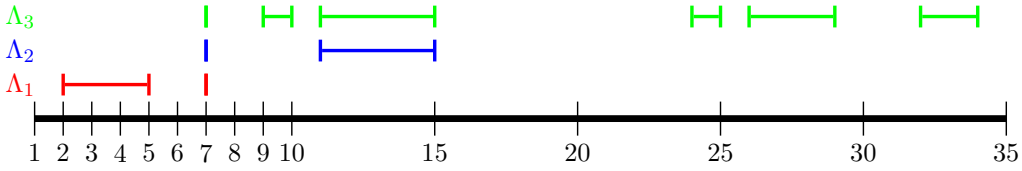


Abbildung 4.5. Invertierte Listen der Keyframes V_1, V_2 und V_3. Man beachte, dass manche Bereiche der Datenbank zu mehreren Keyframes passen. Dies ist dadurch möglich, dass die Keyframes als Mengen von Featurevektoren definiert worden sind, so dass ein Featurevektor in mehreren Keyframes vorkommen kann. Somit kann auch ein Bereich in der Datenbank zu mehreren Keyframes passen.

Der Stiffness-Faktor sei $\sigma = 0.8$. Die invertierten Listen seien nach der Listenvereinigung in Zeile 2 in Algorithmus 4.1 gegeben durch

$$\begin{aligned}\Lambda_1 &= ((2,5),\ (7,7))\\ \Lambda_2 &= ((7,7),\ (11,15))\\ \Lambda_3 &= ((7,7),\ (9,10),\ (11,15),\ (24,25),\ (26,29),\ (32,34))\end{aligned}$$

Die Positionen, an denen die Keyframes in der Datenbank vorkommen, sind in Abbildung 4.5 dargestellt. Jedes Tupel gibt hier ein Segment in der Datenbank an, das zu dem entsprechenden Keyframe „passt“. Dieses „passen“ wird mathematisch ausgedrückt durch $\forall i \in [s:t]: \vec{w}(i) \in V_k$ für das Segment $(s,t) \in \Lambda_k$. Ein Segment ist die längste Abfolge von Frames in der Datenbank, in der sich die Featurevektoren nicht ändern. Dabei ist das erste Element in einem Tupel der Startframe und das zweite Element der Endframe des Segments.
Nach der Listenvereinigung (Zeile 2 in Algorithmus 4.1) wird der Zeigervektor p für die Keyframelisten für jede Keyframeliste auf das erste Listenelement gesetzt. In der Auflistung 4.3 auf Seite 44 werden die ausgeführten Anweisungen des Algorithmus und die rekursiven Aufrufe dargestellt. Dabei wird die Rekursionstiefe durch die Einrückung deutlich gemacht. Das Zeichen $\hookrightarrow$ gibt den Aufruf der Prozedur *sucheRekursiv* an, das Zeichen $\hookleftarrow$ die Rückkehr aus der Prozedur. Man stellt beim Nachvollziehen der Auflistung 4.3 auf Seite 44 fest, dass in diesem Beispiel alle möglichen Matches gefunden wurden: hier gilt also $M = M^{\text{Complete}}$. Des Weiteren stellt man fest, dass es zu jedem gefundenen Match auch tatsächlich mindestens einen Hit gibt. Für den Match $p = (1,1,3)$ existiert der Hit $H = (i_1, i_2, i_3) = (2,7,15)$. Dass dies ein Hit nach Definition 4.4 ist, stellt man wie folgt fest: Der Frame $i_1 = 2$ liegt in dem Segment $\Lambda_1(p_1) = (2,5)$, Frame $i_2 = 7$ liegt in dem Segment $\Lambda_2(p_2) = (7,7)$ und Frame $i_3 = 15$ liegt in dem Segment $\Lambda_3(p_3) = (11,15)$. Des Weiteren gilt die in Definition 4.4 geforderte Abstandsbedingung:

$$\begin{aligned}\sigma \cdot \delta_1 = 4.8 \quad &\leq \quad 7 - 2 = 5 \quad \leq \quad \frac{1}{\sigma} \cdot \delta_1 = 7.5\\ \sigma \cdot \delta_2 = 8 \quad &\leq \quad 15 - 7 = 8 \quad \leq \quad \frac{1}{\sigma} \cdot \delta_2 = 12.5\end{aligned}$$

Auflistung 4.3: Befehlsabfolge von Algorithmus 4.1 unter Benutzung der Datenbank aus Abbildung 4.5

```
1  p_1 = 1; p_2 = 1; p_3 = 1
2  allowedRange = μ_{0.8,6}(2,5) = (2 + ⌈0.8 · 6⌉ , 5 + ⌊1/0.8 · 6⌋) = (2 + 5, 5 + 7) = (7, 12)
3  ↪ (2, allowedRange)
       /* Kein Vorspulen des p_2 nötig.                                              */
4      pointerIncremented = false; intersection = (7,7) ∩ (7,12) = (7,7)
5      newAllowedRange = μ_{0.8,10}(7,7) = (7 + ⌈0.8 · 10⌉ , 7 + ⌊1/0.8 · 10⌋) = (7 + 8, 7 + 12) = (15, 19)
6      ↪ (3, newAllowedRange)
7          p_3 = 3;                                          /* Vorspulen des p_3 auf 3 */
8          pointerIncremented = false; intersection = (11,15) ∩ (15,19) = (15,15)
           /* Wir haben hier den letzten Keyframe. Ein Match wird gefunden           */
9          matches = matches ∪ p = matches ∪ (1,1,3) = {(1,1,3)}
10         p_3 = 4; pointerIncremented = true; intersection = (24,25) ∩ (15,19) = ∅
11         p_3 = 4 − 1 = 3;
12         ↩
13     p_2 = 2; pointerIncremented = true; intersection = (11,15) ∩ (7,12) = (11,12)
14     newAllowedRange = μ_{0.8,10}(11,12) = (11 + ⌈0.8 · 10⌉ , 12 + ⌊1/0.8 · 10⌋) = (19, 24)
15     ↪ (3, newAllowedRange)
16         p_3 = 4 ;                                                  /* Vorspulen */
17         pointerIncremented = false; intersection = (24,25) ∩ (19,24) = (24,24)
18         matches = matches ∪ (1,2,4) = {(1,1,3), (1,2,4)}
19         p_3 = 5; pointerIncremented = true; intersection = (26,29) ∩ (19,24) = ∅
20         p_3 = 5 − 1 = 4
21         ↩
22     p_2 = 3; pointerIncremented = true; intersection = ∅
23     p_2 = 3 − 1 = 2
24     ↩
25 p_1 = 2
26 allowedRange = μ_{0.8,6}(7,7) = (7 + ⌈0.8 · 6⌉ , 7 + ⌊1/0.8 · 6⌋) = (7 + 5, 7 + 7) = (12, 14)
27 ↪ (2, allowedRange)
28     pointerIncremented = false; intersection = (11,15) ∩ (12,14) = (12,14)
29     newAllowedRange = μ_{0.8,10}(12,14) = (12 + ⌈0.8 · 10⌉ , 14 + ⌊1/0.8 · 10⌋) = (20, 26)
30     ↪ (3, newAllowedRange)
31         pointerIncremented = false; intersection = (24,25) ∩ (20,26) = (24,25)
32         matches = matches ∪ (2,2,4) = {(1,1,3), (1,2,4), (2,2,4)}
33         p_3 = 5; pointerIncremented = true; intersection = (26,29) ∩ (20,26) = (26,26)
34         matches = matches ∪ (2,2,5) = {(1,1,3), (1,2,4), (2,2,4), (2,2,5)}
35         p_3 = 6; pointerIncremented = true; intersection = (31,34) ∩ (20,26) = ∅
36         p_3 = 6 − 1 = 5
37         ↩
38     p_2 = 3; pointerIncremented = true; intersection = ∅; p_2 = 3 − 1 = 2
39     ↩
40 Ende der ersten Liste erreicht.
```

In diesem Fall ist der angegebene Hit der einzige mögliche Hit zu diesem Match. Im Allgemeinen kann es jedoch zu einem Match mehrere Hits geben. Tabelle 4.1 listet für jeden Match alle möglichen Hits auf. In diesem Beispiel gibt es nur für den Match $(2, 2, 4)$ mehrere mögliche Hits. Man rechnet wie im oben gezeigten Fall nach, dass die aufgezeigten Hits der Definition 4.4 genügen.

Im Algorithmus 4.2 (*sucheRekursiv*) fällt auf, dass der Zeiger einer invertierten Liste nicht nach jedem Betrachten eines Segmentes erhöht wird, sondern dass er auch stehen bleiben

Match	Hits
$(1, 1, 3)$	$(2, 7, 15)$
$(1, 2, 4)$	$(5, 12, 24)$
$(2, 2, 4)$	$(7, 14, 24),\ (7, 13, 24),\ (7, 12, 24),\ (7, 14, 25),\ (7, 13, 25)$
$(2, 2, 5)$	$(7, 14, 26)$

Tabelle 4.1. Gefundene Matches und mögliche Hits

kann und somit ein Segment mehrfach betrachtet werden kann. Dies wird durch die Zeilen 4, 13 und 15 von Algorithmus 4.2 gesteuert. Es hat zur Folge, dass mehr als eine Rekursion beim Betrachten eines Segmentes gestartet werden kann. Trotzdem kann gezeigt werden, dass die Laufzeit linear von der Summe der Listenlängen abhängt (Abschnitt 4.2.3). Es ist nicht zulässig, den Zeiger nach jeder Betrachtung eines Segmentes zu erhöhen, was damit einher ginge, jedes Segment nur ein einziges Mal zu betrachten. Die folgenden Ausführungen zeigen, dass eine Erhöhung des Zeigers nach dem Betrachten jedes Segmentes zur Auslassung von relevanten Matches führen würde. Dazu stellt man zuerst fest, dass in diesem Beispiel alle möglichen Matches in einer Zusammenhangskomponente (siehe Definition 4.7, Seite 39) liegen, da sie sich überlappen. Der durch Ω^Z (siehe Definition 4.8, Seite 39) definierte Hit-relevante Bereich dieser Zusammenhangskomponente überspannt die Frames 2 bis 26. Das Weglassen der Zeilen 4, 13 und 15 von Algorithmus 4.2 würde verursachen, dass die Matches $(2, 2, 4)$ und $(2, 2, 5)$ nicht gefunden würde. Genauer würde in der Auflistung 4.3 in Zeile 20 der Zeiger p_3 nicht wieder auf 4 dekrementiert und somit weitere mögliche Treffer mit $p_3 = 4$ (also den Match $p = (2, 2, 4)$) ausgeschlossen. In Zeile 23 würde der Zeiger p_2 nicht wieder auf 2 dekrementiert werden und somit die Matches $p = (2, 2, 4)$ und $p = (2, 2, 5)$ ausschließen. Der Bereich der sich ergebenden Zusammenhangskomponente würde dann nur noch die Frames $(1, 24)$ überspannen. Es würde also gelten

$$\Omega^Z(M^{\text{Complete}}) \neq \Omega^Z(M)$$

und das Ergebnis des Algorithmus wäre damit nicht korrekt.

4.2.2 Korrektheitsbeweis

Im Folgenden wird der Korrektheitsbeweis des Algorithmus gegeben. Zuerst wird bewiesen, dass für alle zurückgelieferten Matches Hits gefunden werden können, welche die Abstands-Bedingung erfüllen. Es wird also gezeigt, dass die Matches der Definition 4.5 auf Seite 37 genügen. Danach (Seite 48)wird bewiesen, dass alle relevanten Matches gefunden werden und $\Omega^Z(M^{\text{Complete}}) = \Omega^Z(M)$ gilt.

Behauptung 4.1. *Es seien $K \geq 2$ Keyframes $V_1, \ldots, V_K$ vorgegeben. Für jedes von Algorithmus 4.1 gefundene K-Tupel $p = (p_1, \ldots, p_K)$ gilt: p ist ein Match nach Definition 4.5.*

Man beachte, dass Folgendes gezeigt werden muss:

$$
\begin{aligned}
&\forall \text{ gefundenen K-Tupel } (p_1, \ldots, p_K) \exists \text{ Frames } (i_1, \ldots, i_K), \text{ so dass gilt:} \\
&\quad (\\
&\qquad \forall k \in [1:K] : i_k \in \Lambda_k(p_k) \text{ und} && (4.9) \\
&\qquad \forall k \in [1:K] : \vec{w}(i_k) \in V_k \text{ und} && (4.10) \\
&\qquad \forall k \in [1:K-1] : \sigma \cdot \delta_k \leq i_{k+1} - i_k \leq \frac{1}{\sigma} \cdot \delta_k && (4.11) \\
&\quad).
\end{aligned}
$$

In Worten ausgedrückt bedeutet es, dass für jedes zurückgegebene Tupel Frames gefunden werden müssen, die innerhalb der durch das Tupel beschriebenen Segmente liegen, die Featurevektoren an den Framepositionen müssen zu den Keyframes passen und die Frames müssen die Abstandsbedingung einhalten.

Der folgende Beweis von Behauptung 4.1 ist ein konstruktiver Beweis. Es wird das Tupel $(i_1, \ldots, i_K)$ konstruiert und bewiesen, dass die geforderten Eigenschaften gelten. In dem Beweis wird eine Funktion benötigt, die gewissermaßen eine Umkehrfunktion von μ (Definition 4.9) darstellt. μ wird dafür benutzt, um ausgehend von einem Segment denjenigen Bereich zu berechnen, in dem der nachfolgende Keyframe liegen muss. Die nun zu definierende Funktion λ berechnet für ein Segment, in welchem Bereich der vorhergehende Keyframe gelegen haben muss, um der Abstandsbedingung zu genügen.

Definition 4.10. *Die geforderten Abstände der Keyframes $V_1, \ldots, V_K$ seien mit $\delta_1, \ldots, \delta_{K-1}$ gegeben. Der Stiffness-Parameter sei mit $0 \leq \sigma \leq 1$ gegeben. Dann ist die Funktion λ_k gegeben durch*

$$
\begin{aligned}
\lambda_k := \lambda_{\sigma,\delta_k} &: S \to S \\
(s,t) &\mapsto \begin{cases} (s - \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor, t - \lceil \sigma \cdot \delta_k \rceil) & \sigma > 0 \\ (-\infty, t), & \sigma = 0 \end{cases}
\end{aligned}
$$

Für den Fall $\sigma = 0$ wird auf ein Segment abgebildet, dass nach links beliebig weit ausgedehnt ist.

Beweis 4.1. Betrachte die Befehlsfolge des Algorithmus mit den Aufrufen

$$
\begin{aligned}
&\text{sucheRekursiv}(2, \text{range}_1) \\
&\text{sucheRekursiv}(3, \text{range}_2) \\
&\ldots \\
&\text{sucheRekursiv}(K, \text{range}_{K-1}),
\end{aligned}
$$

welche dazu führen, dass ein K-Tupel $p = (p_1, \ldots, p_K)$ zur Ergebnisliste hinzugefügt wird. Die Variablen intersection_k für $k \in [2:K]$ sind dadurch definiert, dass sie in Zeile 5 oder 14 von Algorithmus 4.2 zu der Berechnung von range_k führen. Für $k = 1$ wird intersection_1 definiert als $\text{intersection}_1 = \Lambda_1(p_1)$.

Nun werden die Frames i_k für $k \in [1 : K]$ nach einem Schema gewählt und anschließend bewiesen, dass die geforderten Bedingungen aus Behauptung 4.1 gelten. Für $k = K$ wähle

$$i_K \in \text{intersection}_K$$

beliebig. Damit ist i_K wohldefiniert, da durch Zeile 6 sichergestellt ist, dass $\text{intersection}_K \neq \emptyset$ ist. Für $k = K - 1, \ldots, 1$ wähle sukzessive

$$i_k \in \lambda_k((i_{k+1}, i_{k+1})) \cap \text{intersection}_k$$

Zuerst ist zu zeigen, dass die i_k wohldefiniert sind. Das bedeutet in diesem Fall, dass die Schnitte $\forall k \in [K-1:1] : \lambda_k((i_{k+1}, i_{k+1})) \cap \text{intersection}_k$ nicht leer sein dürfen. Zeige nun mit einem Widerspruchsbeweis, dass $\forall k \in [K-1:1] : \lambda_k((i_{k+1}, i_{k+1})) \cap \text{intersection}_k \neq \emptyset$.

Begründe nun sukzessive für $k = K-1, ..., 1$ unter der jeweiligen Annahme, dass i_{k+1} bereits gewählt wurde. Diese Annahme kann getroffen werden, da i_K bereits zulässig gewählt wurde. Angenommen, $\lambda_k((i_{k+1}, i_{k+1})) \cap \text{intersection}_k = \emptyset$.

Fall 1: $\sigma > 0$**.** Es gilt

$$\lambda_k((i_{k+1}, i_{k+1})) = (i_{k+1} - \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor, i_{k+1} - \lceil \sigma \cdot \delta_k \rceil)$$

Der Schnitt ist leer, genau dann wenn folgende Bedingung gilt (siehe auch Definition 4.2):

$$\begin{aligned} & i_{k+1} - \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor > \text{intersection}_k \,.\, \text{end} \quad \vee \\ & i_{k+1} - \lceil \sigma \cdot \delta_k \rceil < \text{intersection}_k \,.\, \text{start} \end{aligned} \tag{4.12}$$

Im Folgenden wird das Symbol $\circ$ als Platzhalter für ein beliebiges, für die Begründungen nicht relevantes Segment eingesetzt. Nun betrachte man, dass $i_{k+1} \in \text{intersection}_{k+1}$ gilt. Aus $\text{intersection}_{k+1} = \circ \cup \text{range}_k$ folgt direkt, dass $i_{k+1} \in \text{range}_k$. Man beachte, dass in Zeile 10 range_k durch $\text{range}_k = \mu_k(\text{intersection}_k)$ definiert wird (für $k = 1$ wird range_1 in Algorithmus 4.1 Zeile (7) definiert durch $\text{range}_1 = \mu_1(\text{intersection}_1)$ mit der oben getroffenen Wahl von intersection_1). Somit gilt

$$\begin{aligned} & i_{k+1} \in \mu_k(\text{intersection}_k) \\ \Leftrightarrow \quad & i_{k+1} \geq \text{intersection}_k \,.\, \text{start} + \lceil \sigma \cdot \delta_k \rceil \quad \wedge \\ & i_{k+1} \leq \text{intersection}_k \,.\, \text{end} + \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor \\ \Leftrightarrow \quad & i_{k+1} - \lceil \sigma \cdot \delta_k \rceil \geq \text{intersection}_k \,.\, \text{start} \quad \wedge \\ & i_{k+1} - \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor \leq \text{intersection}_k \,.\, \text{end} \end{aligned} \tag{4.13}$$

Man sieht, dass Gleichung (4.13) im direkten Widerspruch zu Gleichung (4.12) steht. Somit ist die Annahme widerlegt.

Fall 2: $\sigma = 0$**.** Es gilt

$$\lambda_k((i_{k+1}, i_{k+1})) = (-\infty, i_{k+1})$$

Der Schnitt ist leer, genau dann wenn folgende Bedingung gilt (siehe auch Definition (4.2)):

$$i_{k+1} < \text{intersection}_k \,.\, \text{start} \tag{4.14}$$

Nun betrachte man, dass $i_{k+1} \in \text{intersection}_{k+1}$ gilt. Dies ist ein direkter Widerspruch zu Gleichung (4.14). Somit ist die Annahme widerlegt.

Es wurde gezeigt, dass $\forall k \in [1 : K-1] : \lambda_k((i_{k+1}, i_{k+1})) \cap \text{intersection}_k \neq \emptyset$. Damit ist die oben angegebene Wahl von i_k zulässig und alle i_k wohldefiniert.

Es ist nun zu zeigen, dass (4.9) und (4.10) gelten, also dass $\forall k \in [1 : K] : (i_k \in \Lambda_k(p_k)$ und $\vec{w}(i_k) \in V_k)$. Dazu erkennt man, dass $i_k \in \text{intersection}_k = \Lambda_k(p_k) \cap \circ$ und dass der Schnitt $\Lambda_k(p_k) \cap \circ$ wegen der while-Bedingung in Zeile 6, Algorithmus 4.2, nicht leer ist. Für $k = 1$ ist intersection_1 per Definition nicht leer. Somit ist $i_k \in \Lambda_k(p_k)$ gezeigt. Da Λ_k durch Konstruktion in Zeile 2 (Algorithmus 4.1) nur Bereiche der Datenbank $\vec{w}$ enthält, die aus Featurevektoren von V_K bestehen, gilt

$$\vec{w}(i_K) \in V_K.$$

Damit ist gezeigt, dass Gleichung (4.9) und (4.10) gilt, also dass $\forall k \in [1 : K] : (i_k \in \Lambda_k(p_k)$ und $\vec{w}(i_k) \in V_k)$ gilt.

Schließlich ist zu zeigen, dass die Wahl der i_k der Abstandsbedingung (4.11) genügen, damit die i_k einen Hit darstellen. Dazu ist zu zeigen, dass $\forall k \in [1 : K-1]$: $\sigma \cdot \delta_k \leq i_{k+1} - i_k \leq \frac{1}{\sigma} \cdot \delta_k$ gilt. Sei $k \in [1 : K-1]$. Der Frame i_k wurde gewählt durch $i_k \in \lambda_k((i_{k+1}, i_{k+1})) \cap \circ$. Mit $\lambda_k((i_{k+1}, i_{k+1})) = (i_{k+1} - \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor, i_{k+1} - \lceil \sigma \cdot \delta_k \rceil)$ gilt

$$\begin{aligned} i_k &\geq i_{k+1} - \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor \quad \wedge \\ i_K &\leq i_{k+1} - \lceil \sigma \cdot \delta_k \rceil \\ &\Leftrightarrow \\ i_{k+1} - i_k &\leq \left\lfloor \frac{1}{\sigma} \cdot \delta_k \right\rfloor \leq \frac{1}{\sigma} \cdot \delta_k \quad \wedge \\ i_{k+1} - i_k &\geq \lceil \sigma \cdot \delta_k \rceil \geq \sigma \cdot \delta_k. \end{aligned}$$

Damit ist gezeigt, dass die i_k die Abstandsbedingung erfüllen.

Insgesamt wurde gezeigt, dass die Wahl der i_k zulässig ist und dass die i_k die Bedingungen aus Gleichung (4.9), (4.10) und (4.11) erfüllen. □

Im nun folgenden zweiten Teil des Korrektheitsbeweises wird bewiesen, dass alle relevanten Matches gefunden werden.

Behauptung 4.2. *Sei $M^{Complete}$ die Menge aller möglichen Matches. Dann gilt für die vom Algorithmus gefundene Menge von Matches M*

$$\Omega^Z(M^{Complete}) = \Omega^Z(M).$$

Beweis 4.2. Angenommen nicht, also angenommen,

$$\Omega^Z(M^{\text{Complete}}) \neq \Omega^Z(M).$$

Im Beweis 4.1 wurde bereits gezeigt, dass alle gefundenen Matches der Definition 4.5 genügen. Also gilt: $M \subseteq M^{\text{Complete}}$. Aus der Annahme folgt nun: $\exists Q \in M^{\text{Complete}} : Q \notin M$. Sei Q ein Match, der nicht in M liegt. Dass Q ein Match ist, bedeutet auch, dass es eine theoretisch mögliche Aufrufkaskade

$$\begin{aligned}&\text{sucheRekursiv}(2, \text{allowedRangeQ}_1)\\&\text{sucheRekursiv}(3, \text{allowedRangeQ}_2)\\&\dots\\&\text{sucheRekursiv}(K, \text{allowedRangeQ}_{K-1}),\end{aligned}$$

gibt, bei welcher der Treffer Q zur Trefferliste hinzugefügt würde.

Es sei p der im Algorithmus 4.1 benutzte Zeigervektor in die Keyframelisten. $\forall k \in [1 : K-1]$ seien allowedRange_k im Algorithmus 4.1 Zeile 7 oder im Algorithmus 4.2 Zeile 10 durch

$$\text{allowedRange}_k = \mu_k(\text{intersection}_k)$$

definiert. Für den Beweis beachte man, dass für jedes $k \in [1 : K-1]$ $\text{allowedRange}_k.\text{start}$ im Verlaufe des Algorithmus mit p_k monoton wächst, da die invertierten Listen der Keyframes lexikographisch geordnet sind und sich allowedRange_k ergibt als

$$\text{allowedRange}_k = \mu_k(\text{intersection}_k) = \mu_k(\Lambda_k(p_k) \cap \text{allowedRange}_{k-1})$$

Mit anderen Worten heißt dies, dass sich die allowedRange_k aus den $\Lambda_k(p_k)$ ergeben. Da die Λ_k lexikographisch geordnet sind, wachsen die allowedRange_k mit p_k monoton.

Wird ein möglicher Treffer $Q = (p_{Q,1}, \dots, p_{Q,K})$ nicht gefunden, so gibt es einen Zeitpunkt im Algorithmus, an dem für ein $i \in [1 : K]$ gilt:

$$p_i > p_{Q,i} \wedge \forall k \in [1 : K] \setminus \{i\} : p_k \leq p_{Q,k}.$$

Nun beachte man, dass $i = 1$ nicht gelten kann. Denn wenn $p_1 > p_{Q,1} \wedge \forall k \in [2 : K] : p_k \leq p_{Q,k}$ gelte, so wäre vorher mit $p_1 = p_{Q,1}$ eine Rekursion gestartet worden (Algorithmus 4.1 Zeile 8: Diese Aufrufen einer Rekursion mit jedem Element der ersten Keyframeliste kann garantiert werden). Dann würde entweder der Zustand $\forall k \in [1 : K] : p_k = p_{Q,k}$ erreicht und der Treffer gemeldet oder es gelte während der Ausführung $p_k > p_{Q,k}$ für ein $k \in [2 : K]$, ohne dass der Treffer gefunden wird.

Argumentiere nun sukzessive für $i = 2, \dots, K-1$:

Angenommen, $i = 2, \dots, K-1$ sei der Zeigerindex, für den gilt, dass

$$p_i > p_{Q,i} \wedge \forall k \in [1 : K] \setminus \{i\} : p_k \leq p_{Q,k} \tag{4.15}$$

Dann muss es vorher ein Zustand $p_i = p_{Q,i} \wedge \forall k \in [1 : K] \setminus \{i\} : p_k \leq p_{Q,k}$ gegeben haben, aus dem p_i erhöht wurde. Betrachte diesen Zustand. In Algorithmus 4.2 Zeile 3 kann der Zeiger

$p_i = p_{Q,i}$ nicht überspult werden: Es gilt $p_{i-1} \leq p_{Q,i-1}$. Da die $\text{allowedRange}_k\,.\,\text{start}$ monoton mit p_k wachsen, gilt damit auch

$$\text{allowedRange}_{i-1}\,.\,\text{start} \leq \text{allowedRangeQ}_{i-1}\,.$$

Da bei der theoretisch möglichen Aufrufkaskade in Zeile 2

$$\Lambda_i(p_{Q,i}).\,\text{end} \geq \text{allowedRangeQ}_{i-1}\,.\,\text{start}$$

gegolten haben muss (ansonsten wäre keine Rekursion mit $p_i = p_{Q,i}$ gestartet, sondern der Zeiger übersprungen worden), gilt auch

$$\Lambda_i(p_{Q,i}).\,\text{end} \geq \text{allowedRangeQ}_{i-1}\,.\,\text{start} \geq \text{allowedRange}_{i-1}\,.\,\text{start}$$

und in dem betrachteten Zustand kann $p_i = p_{Q,i}$ nicht überspult werden, da die while-Bedingung aus Zeile 6 verletzt ist. Es kann also $p_i = p_{Q,i}$ so lange nicht überspult werden, wie $p_{i-1} \leq p_{Q,i-1}$ gilt. Dass $p_{i-1} \leq p_{Q,i-1}$ gilt, wurde bereits gezeigt. Also kann $pi = p_{Q,i}$ nicht überspult werden. Dann kann der Zeiger nur erhöht werden, wenn $\text{intersection}_i \neq \emptyset$ in Zeile 5 und dann mit $p_i = p_{Q,i}$ eine Rekursion gestartet wird. Bei dieser Rekursion kommt es entweder zu einem Zustand, in dem $\forall k \in [1:K] : p_k = p_{Q,k}$ gilt und der Treffer Q gefunden wird (Widerspruch zur Annahme), oder die Annahme, dass p_i der Zeiger ist, für den Bedingung (4.15) gilt, ist falsch. Also ist p_i nicht der erste Zeiger, für den Bedingung (4.15) gilt.

Wenn nun K der Zeiger ist, für den $p_K > p_{Q,K} \wedge \forall k \in [1:K-1] : p_k \leq p_{Q,k}$ gilt, so muss es vorher einen Zustand mit $p_K = p_{Q,K} \wedge \forall k \in [1:K-1] : p_k \leq p_{Q,k}$ gegeben haben. Wie oben wird nun argumentiert, dass

$$\Lambda_K(p_{Q,K}).\,\text{end} \geq \text{allowedRangeQ}_{K-1}\,.\,\text{start} \geq \text{allowedRange}_{K-1}\,.\,\text{start}$$

gilt und daher der Zeiger nicht in Zeile 3 überspult werden kann. Nun gibt es zwei Möglichkeiten: Entweder es gilt $\text{intersection}_K = \emptyset$ in Zeile 5 so lange, bis $\forall k \in [1:K] : p_k = p_{Q,k}$ gilt und der Treffer Q gefunden wird (Widerspruch zur Annahme), oder es gilt $\text{intersection}_K \neq \emptyset$ und es wird ein Match Q' gefunden mit $p_1 \leq p_{Q,1}$ und $p_K = p_{Q,k}$. Da der überspannte Hit-relevante Bereich von Q' den überspannten Hit-relevanten Bereich von Q vollständig enthält, trägt Q nichts zu den Bereichen in $\Omega^Z(M)$ bei und es gilt

$$\Omega^Z(M^{\text{Complete}}) = \Omega^Z(M).$$

□

Der Beweis hat Folgendes gezeigt: Wenn man annimmt, dass ein beliebiger Match $Q = (p_{Q,1}, p_{Q,2}, \ldots, p_{Q,K})$ nicht gefunden wird, so kann garantiert werden, dass ein anderer Match $Q' = (p_{Q',1}, p_{Q',2}, \ldots, p_{Q',K})$ gefunden wird, für den gilt: $p_{Q',K} = p_{Q,K}$ und $p_{Q',1} \leq p_{Q,1}$. Der Match Q' überspannt somit den Match Q komplett und Q' muss damit nicht gefunden werden, da er nichts zu Ω^Z beiträgt.

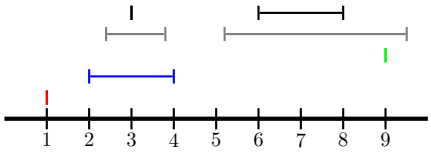


Abbildung 4.6. Positionen der Keyframes in der Datenbank. Die Keyframes sind in den Farben rot, blau und grün dargestellt. In grau dargestellt sind die Bereiche, die sich bei Weglassen der Rundung ergeben. In schwarz dargestellt sind die Bereiche, die sich mit korrekter Rundung ergeben.

Auflistung 4.4: Fehlerhafte Befehlsabfolge von Algorithmus 4.1

```
1  p_1 = 1
2  allowedRange = μ_{0.7,2}(1,1) = (1 + ⌈0.7 · 2⌉ , 1 + ⌊1/0.7 · 2⌋) = (1 + 1.4, 1 + 2.86) = (2.4, 3.86)
3  ↪ (2, allowedRange)
4      pointerIncremented = false
5      intersection = (2, 4) ∩ (2.4, 3.86) = (2.4, 3.86)
6      newAllowedRange = μ_{0.7,4}(2.4, 3.86) = (2.4 + ⌈0.7 · 4⌉ , 3.86 + ⌊1/0.7 · 4⌋) = (2.4 + 2.8, 3.86 + 5.71) =
       (5.2, 9.57)
7      ↪ (3, newAllowedRange)
8          pointerIncremented = false
9          intersection = (9, 9) ∩ (5.2, 9.57) = (9.9)
10         matches = matches ∪ (1, 1, 1) = {(1, 1, 1)}
11         ...
```

Bedeutung der Rundungen für die Korrektheit Das Aufrunden und Abrunden der Intervallgrenzen, wie es im Algorithmus durch die Benutzung der Funktion μ_k (Definition 4.9, Seite 41) geschieht, ist für die Korrektheit des Algorithmus notwendig. Das folgende Beispiel zeigt einen Fall, in dem das Weglassen der Rundungen dazu führt, dass ein K-Tupel von Zeigern zurückgegeben wird, für das keine Frames gefunden werden können, die der Stiffness-Bedingung genügen. Der Grund für die Rundungen ist, dass Frames nur an ganzzahligen Positionen in der Datenbank vorliegen. Wenn die Rundungen weggelassen werden, dann geht der Algorithmus davon aus, dass auch Frames an gebrochen rationalen Positionen gewählt werden können, beispielsweise an Position 3.8. Da es in der Praxis keinen Frame an dieser Position gibt, kann das Fortführen der Berechnungen mit diesem Frame zu Treffern führen, welche die Stiffness-Bedingung verletzen.

Es seien drei Keyframes vorgegeben, die Abstände seien $\delta_1 = 2$ und $\delta_2 = 4$. Die Stiffness sei $\sigma = 0.7$. Die invertierten Listen der Keyframes seien wie folgt vorgegeben:

$$\Lambda_1 = (1, 1)$$
$$\Lambda_2 = (2, 4)$$
$$\Lambda_3 = (9, 9)$$

Diese sind in Abbildung 4.6 dargestellt. Würde man die Rundungen in der Definition 4.9 von μ weglassen, so ergäbe sich die Befehlsabfolge in Auflistung 4.4 auf Seite 51: Es wird ein Match gefunden mit $p = (1,\ 1,\ 1)$. Es lassen sich jedoch keine Keyframes aus den Bereichen der Datenbank auswählen, für welche die Stiffness-Bedingung gilt: Für den ersten Keyframe bleibt keine Wahl, es muss gelten $i_1 = 1$. Für den zweiten Keyframe bleibt ebenfalls keine Wahl. Der zugehörige Frame muss im Bereich $[i_1 + \sigma \cdot \delta_1 : i_1 + \frac{1}{\sigma} \cdot \delta_1] \approx [2.4 : 3.86]$ liegen, dort

bleibt nur $i_2 = 3$. Der dritte Keyframe muss nun im Bereich $[i_2+\sigma\cdot\delta_2 : i_2+\frac{1}{\sigma}\cdot\delta_2] \approx [4.8 : 8.71]$ liegen, dort befindet sich allerdings kein Frame, der zum dritten Keyframe passt. Dies zeigt, dass ohne die Rundungen falsche Matches gefunden werden können.

4.2.3 Laufzeitbetrachtung

Es soll die Worst-Case-Laufzeit Γ des Algorithmus in Abhängigkeit von der Länge der invertierten Listen der Keyframes bestimmt werden. Es wird also eine Funktion gesucht, die abhängig von den Listenlängen ℓ_k, $k \in [1 : K]$, eine Zahl liefert, die proportional zu der Anzahl der im Algorithmus benötigten Operationen zur Bestimmung der Matches ist. Man beachte dabei, dass ℓ_k die Anzahl der in den invertierten Listen enthaltenen Segmente bezeichnet und nicht die enthaltenen Frames. Als Maß für die Anzahl der Operationen wird die Anzahl der Aufrufe der rekursiven Funktion *sucheRekursiv* plus die Anzahl der gefundenen Matches genommen. Dabei wird die für die Listenvereinigung (Algorithmus 4.1, Zeile 2) benötigte Zeit vernachlässigt, da sie linear in der Summe der Listenlängen ist und somit - wie im Folgenden gezeigt wird - höchstens so groß ist wie die Worst-Case-Laufzeit für den Rest des Algorithmus.

Oberflächlich betrachtet ist die Worst-Case-Laufzeit proportional zur Summe der Längen der invertierten Listen, da bei jeder Betrachtung eines Listenelementes eine Rekursion gestartet werden kann:

$$\Gamma \in O\left(\sum_{k=1}^{K} \ell_k\right)$$

Bei genauer Betrachtung stellt man jedoch fest, dass nur der Zeiger der ersten invertierten Liste ausschließlich erhöht wird. Die Zeiger der anderen Listen können nach einer Betrachtung auch stehen bleiben und somit mehrfach betrachtet werden, also vielfach die rekursive Funktion aufrufen. Vor der genauen Analyse wird nun ein Beispiel gegeben, in dem ein mögliches Worst-Case Szenario konstruiert wird. Erst danach wird durch systematische Argumentation eine allgemeine obere Schranke für die Laufzeit des Algorithmus hergeleitet.

Beispiel. Zur Konstruktion des Szenarios ist zu beachten, dass möglichst kurze Listen aufgebaut werden müssen, welche eine möglichst große Anzahl von Rekursionen zur Folge haben.

Für die Elemente der invertierten Liste des ersten Keyframes Λ_1 wird jeweils höchstens eine Rekursion gestartet. Dies ist durch die for-Schleife im Algorithmus 4.1, Zeile 5, gewährleistet. Die einzigen Listenelemente, die mehrfach betrachtet werden können und somit zu einer überlineraren Laufzeit beitragen können, sind die Segmente der Listen $\Lambda_2, \ldots, \Lambda_K$. Mit einem Element der Listen $\Lambda_2, \ldots, \Lambda_{K-1}$ können mehrere Rekursionen gestartet werden und mit einem Element der Liste Λ_K können mehrere Matches gefunden werden. Man beachte, dass in diesen invertierten Listen nur diejenigen Kandidaten mehrfach betrachtet werden, die am „Rand“ von allowedRange liegen. Genauer gesagt wird das p_k-te Segment aus Λ_k für $k \in [2 : K]$ mehrfach betrachtet genau dann, wenn

$$\begin{aligned} \Lambda_k(p_k) \cap \text{allowedRange} \neq \emptyset \quad \wedge \\ \Lambda_k(p_k + 1) \cap \text{allowedRange} = \emptyset \end{aligned}$$

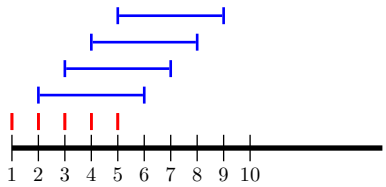


Abbildung 4.7. Positionen der Keyframes in der Datenbank in einem worst-case-Szenario für $\sigma = 1.0$.

Dabei wird der Schnitt auch dann als leer angesehen, wenn der Index von Λ_k das Ende der Liste überschritten hat. Es ist also darauf zu achten, dass die Segmente in den Listen $\Lambda_2, \ldots, \Lambda_K$ möglichst oft am „Rand“ vom allowedRange liegen. Schon mit einem Segment pro invertierter Liste lässt sich ein Szenario konstruieren, bei dem die Elemente maximal oft betrachtet werden. Dabei sei die Stiffness auf $\sigma = 1.0$ und die Abstände der Keyframes auf $\delta_k = 1$ festgelegt. So ein Szenario ist für $K = 5$ in Abbildung 4.7 dargestellt. Die invertierten Listen für $K \geq 2$ Keyframes sehen dabei wie folgt aus:

$$\begin{aligned}
\Lambda_1 &= ((1,1),\ (2,2),\ (3,3),\ \ldots, (\ell_1, \ell_1)) \\
\Lambda_2 &= ((2, \ell_1 + 1)) \\
\Lambda_3 &= ((3, \ell_1 + 2)) \\
&\ldots \\
\Lambda_{K-1} &= ((K-1, \ell_1 + K - 2)) \\
\Lambda_K &= ((K, \ell_1 + K - 1))
\end{aligned}$$

Man beachte, dass $\ell_k = 1$ für $k \in [2 : K]$. Die Keyframelisten sind so konstruiert, dass ℓ_1 Matches zurückgeliefert werden: Jedes Element aus Λ_1 führt zu genau einem Match. Dabei wird für jedes Segment aus Λ_1 mit allen Segmenten der Listen $\Lambda_2, \ldots, \Lambda_{K-1}$ eine Rekursion gestartet und ein Match gefunden. Die Laufzeit beträgt hier also für $K \geq 2$

$$\begin{aligned}
\Gamma &\in O(\ell_1 + \ell_1 \cdot (\ell_2 + \ldots + \ell_K)) \\
&= O(\ell_1 + \ell_1 \cdot (K - 1)) \\
&= O(K \cdot \ell_1)
\end{aligned}$$

In praktischen Beispielen kann man davon ausgehen, dass die Anzahl der Keyframes konstant ist oder sich in einem kleinen Bereich von $K \in [2 : 10]$ bewegt. Mit der Annahme $K = \text{const}$ gilt:

$$\Gamma \in O(\ell_1).$$

Die asymptotische Laufzeit ist also für dieses konstruierte Beispiel nur abhängig von der Länge der ersten invertierten Liste.

Beweis Nach diesem einführenden Beispiel wird eine obere Schranke für die Laufzeit Γ des Algorithmus angegeben und bewiesen.

Behauptung 4.3. *Eine obere Schranke für die Laufzeit Γ von Algorithmus 4.1 ist bei $K \geq 2$ Keyframes und invertierten Listen $\Lambda_1, \ldots, \Lambda_K$ sowie deren Längen $\ell_1, ..., \ell_K$ gegeben durch*

$$\begin{aligned}
\Gamma \leq\ & K \cdot \ell_1 + (K-1) \cdot \ell_2 + \ldots \\
& + 3 \cdot \ell_{K-2} + 2 \cdot \ell_{K-1} + \ell_K - \frac{K \cdot (K-1)}{2}
\end{aligned} \tag{4.16}$$

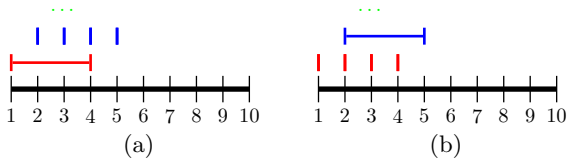


Abbildung 4.8. Szenarien, in denen die maximale Anzahl von rekursiven Aufrufen erreicht wird. In beiden Szenarien wird $\sigma = 1.0$ und $\delta_i = 1$ angenommen. In **(a)** gilt: $\ell_1 = 1$, $\ell_2 = 4$. Insgesamt gibt es bis zur zweiten Keyframeliste genau 5 Aufrufe der rekursiven Funktion. Hier wird die Gleichheit $\xi_2 + \xi_1 = 2 \cdot \ell_1 + \ell_2 - 1$ angenommen. In **(b)** gilt: $\ell_1 = 4$, $\ell_2 = 1$. Insgesamt gibt es bis zur zweiten Keyframeliste genau 8 Aufrufe der rekursiven Funktion. Hier wird ebenfalls die Gleichheit $\xi_2 + \xi_1 = 2 \cdot \ell_1 + \ell_2 - 1$ angenommen.

Dabei ist Γ *eine Zahl, welche die Anzahl der Aufrufe der Funktion* sucheRekursiv *plus die Anzahl der gefundenen Matches angibt.*

Beweis 4.3. Um die Behauptung 4.3 zu beweisen fragt man sich, wie oft die rekursive Funktion höchstens aufgerufen werden kann in Abhängigkeit von der Anzahl der Segmente in den invertierten Listen. Im Folgenden bezeichnet ξ_k die Zahl, welche die Liste Λ_k zur Laufzeit beiträgt.

Λ_1 Jedes Element von Λ_1 verursacht höchstens einen Aufruf. Dies ist durch die For-Schleife (Zeile 5, Algorithmus 4.1) garantiert. Also gilt: $\xi_1 \leq \ell_1$.

Λ_2 Jedes Listenelement kann einen Aufruf verursachen. Jedes Element kann auch mehr als einen Aufruf verursachen, jedoch ist die *Summe der Mehrfachaufrufe begrenzt.* Wenn beim Betrachten eines Segmentes aus Λ_2 mehr als ein rekursiver Aufruf getätigt wird, so kann das nur geschehen, wenn vor jedem zusätzlichem Aufruf p_1 inkrementiert wird. Somit kann es maximal $\xi_1 - 1$ zusätzliche Aufrufe geben (da p_1 höchstens so oft inkrementiert werden kann). Insgesamt gilt also: $\xi_2 \leq \ell_2 + \xi_1 - 1$. Abbildung 4.8 illustriert zwei Fälle, in denen die Gleichheit angenommen wird.

Λ_3 Jedes Listenelement kann einen Aufruf verursachen. Jedes Element kann auch mehr als einen Aufruf verursachen, jedoch ist die Summe der Mehrfachaufrufe begrenzt. Wird bei der Betrachtung eines Segmentes aus Λ_3 mehrfach die Funktion aufgerufen, so kann garantiert werden, dass mindestens einer der Zeiger p_1 oder p_2 sich verändert hat: Entweder wird p_1 oder p_2 oder es werden beide Zeiger inkrementiert. In Abbildung 4.9 sind drei Beispiele gegeben, in welchen die drei Fälle vorkommen. Somit kann es höchstens $\xi_2 - 1$ Mehrfachaufrufe geben. Also gilt: $\xi_3 \leq \ell_3 + \xi_2 - 1$. Die Abbildung 4.9(a),(b) zeigt zwei Fälle, in denen die Gleichheit angenommen wird. Mit dieser Argumentation kann weiter verfahren werden bis zur letzten invertierten Liste.

...

Λ_K Bei Betrachtung der Segmente der K-ten invertierten Liste wird keine Rekursion mehr gestartet, sondern Matches zur Ergebnisliste hinzugefügt. Jedes Listenelement kann zu einem Match führen. Jedes Element kann auch zu mehr als einem Match beitragen, jedoch ist die Summe der mehrfachen Beiträge begrenzt. Werden für ein Segment aus Λ_K mehrere Matches gefunden, so kann garantiert werden, dass mindestens einer der Zeiger p_k für $k \in [1 : K - 1]$ sich verändert hat. Somit kann es höchstens $\xi_{K-1} - 1$ Mehrfachmatches geben. Also gilt: $\xi_K \leq \ell_k + \xi_{K-1} - 1$.

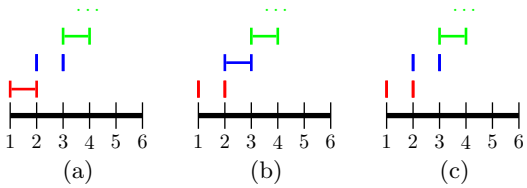


Abbildung 4.9. In allen Szenarien wird $\sigma = 1.0$ und $\delta_i = 1$ angenommen. Es sind Fälle aufgezeichnet, in denen das erste Element aus Λ_3 mehrfach betrachtet wird. In **(a)** wird vor der zweiten Betrachtung von $\Lambda_3(1)$ der zweite Keyframezeiger p_2 erhöht. In **(b)** wird vor der zweiten Betrachtung von $\Lambda_3(1)$ der erste Keyframezeiger p_1 erhöht. In **(c)** wird vor der zweiten Betrachtung von $\Lambda_3(1)$ sowohl p_1 als auch p_2 inkrementiert.
Des Weiteren gilt in **(a)**: $\xi_3 + \xi_2 + \xi_1 \leq 3 \cdot \ell_1 + 2 \cdot \ell_2 + \ell_3 - 3 = 3 + 4 + 1 - 3 = 5$. Es finden auch 5 Aufrufe der rekursiven Funktion statt, hier ist also die Gleichheit gegeben. In **(b)** ist dies ebenfalls der Fall, die Vorhersage von 6 Aufrufen findet statt. In **(c)** sind die tatsächlich stattfindenden Aufrufe mit 6 Stück kleiner als die errechneten 8 Aufrufe.

Um die Gesamtlaufzeit zu berechnen, müssen nur die ξ_k addiert werden:

$$
\begin{aligned}
\Gamma \quad &\leq \quad \sum_{k=1}^{K} \xi_k \\
&\leq \quad \ell_1 + (\ell_2 + \xi_1 - 1) + (\ell_3 + \xi_2 - 1) + \ldots \\
&\qquad +(\ell_{K-1} + \xi_{K-2} - 1) + (\ell_K + \xi_{K-1} - 1) \\
&\overset{\text{Einsetzen der } \xi_k}{=} \quad \ell_1 + (\ell_2 + \ell_1 - 1) + (\ell_3 + (\ell_2 + \ell_1 - 1) - 1) + \ldots \\
&\qquad +(\ell_K + (\ell_{K-1} + (\ldots(\ell_2 + \ell_1 - 1)\ldots) - 1) - 1) \\
&\overset{\text{Zusammenfassen}}{=} \quad K \cdot \ell_1 + (K-1) \cdot \ell_2 + \ldots \\
&\qquad +3 \cdot \ell_{K-2} + 2 \cdot \ell_{K-1} + \ell_K - 1 \cdot \sum_{k=2}^{K} (k-1) \\
&\overset{\text{Indextransformation}}{=} \quad K \cdot \ell_1 + (K-1) \cdot \ell_2 + \ldots \\
&\qquad +3 \cdot \ell_{K-2} + 2 \cdot \ell_{K-1} + \ell_K - 1 \cdot \sum_{k=1}^{K-1} k \\
&\overset{\text{Gaußsche Summenformel}}{=} \quad K \cdot \ell_1 + (K-1) \cdot \ell_2 + \ldots \\
&\qquad +3 \cdot \ell_{K-2} + 2 \cdot \ell_{K-1} + \ell_K - \frac{K \cdot (K-1)}{2}
\end{aligned}
$$

□

Abbildung 4.10 zeigt Beispiele, bei denen die Gleichheit für die Gesamtlaufzeit angenommen wird.

Es wurde eine Gleichung für die maximale Laufzeit in Abhängigkeit von den Listenlängen hergeleitet. Nun wird ausgehend von dieser Formel gezeigt, dass die Laufzeit des Algorithmus mit der Annahme $K = const$ linear von der Summe den Listenlängen abhängt.

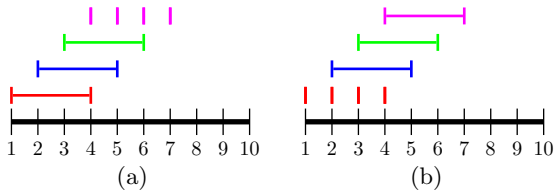


Abbildung 4.10. Für die Beispiele seien $K = 4$ Keyframes vorgegeben. Es gelte Stiffness $\sigma = 1.0$, $\delta_i = 1$. Die Laufzeit beträgt nach Gleichung (4.16): $\Gamma \leq 4 \cdot \ell_1 + 3 \cdot \ell_2 + 2 \cdot \ell_3 + \ell_4 - 6$. In **(a)** beträgt die Laufzeit, also die Anzahl der rekursiven Aufrufe plus die Anzahl der gefundenen Matches genau 7. Mit $\ell_1 = 1, \ell_2 = 1, \ell_3 = 1$ und $\ell_4 = 4$ beträgt die vorhergesagte Laufzeit Γebenfalls 7. In **(b)** beträgt die Laufzeit 16. Die berechnete Laufzeit ist ebenfalls $\Gamma = 16$. In diesen konstruierten Beispielen ist die Gleichheit gegeben, in den meisten anderen Beispielen liegt die tatsächliche Laufzeit unter der Schranke Γ.

Folgerung 4.1. *Unter der Annahme, dass die Anzahl der Keyframes beschränkt ist, also $K \leq K_C$, ist die Laufzeit linear in der Summe der Listenlängen:*

$$\Gamma \in O(\sum_{k=1}^{K_C} \ell_k)$$

Beweis 4.4. Die Aussage ergibt sich als direkte Folgerung aus Gleichung (4.16), Seite 53. Sei $K \leq K_C$. Dann gilt:

$$\begin{aligned}
\Gamma &\leq K \cdot \ell_1 + (K-1) \cdot \ell_2 + \ldots \\
&\quad + 3 \cdot \ell_{K-2} + 2 \cdot \ell_{K-1} + \ell_K - \frac{K \cdot (K-1)}{2} \\
&\leq K_C \cdot \ell_1 + K_C \cdot \ell_2 + \ldots \\
&\quad + K_C \cdot \ell_{K-2} + K_C \cdot \ell_{K-1} + K_C \cdot \ell_K - \frac{K_C \cdot (K_C-1)}{2} \\
&= K_C \cdot (\sum_{k=1}^{K_C} \ell_k) - C \\
&\in O(\sum_{k=1}^{K_C} \ell_k)
\end{aligned}$$

□

Bemerkung 4.2. Die Beispiele, in denen die Worst-Case-Laufzeit angenommen wird, sind insofern künstlich, als dass diese Beispiele in den Datenbanken, auf die der Algorithmus angewandt wird, gar nicht vorkommen können, sofern man Keyframes bezüglich einer einzigen Featurefunktion F benutzt. In allen Worst-Case-Beispielen überdeckt ein Segment einer Keyframeliste einen Bereich, in welchem in einer anderen Keyframeliste kürzere Segmente vorkommen. Beispielsweise überdeckt in Abbildung 4.8(a) das Segment der ersten Keyframeliste den Bereich 1 bis 4. In der zweiten Keyframeliste wird dieser Bereich nun durch mehrere Keyframes „zerstückelt“. Betrachtet man die in den Experimenten verwendete Motion-Capture-Datenbank bezüglich einer F-Segmentierung, so sind solche Konstruktionen nicht möglich. Das liegt daran, dass die Motion-Capture-Datenbank in Segmente unterteilt ist, in denen sich die Featurevektoren nicht ändern. Ein Keyframe ist eine Menge von Featurevektoren.

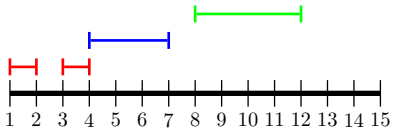


Abbildung 4.11. Für dieses Beispiel seien $K = 3$ Keyframes vorgegeben. Es gelte Stiffness $\sigma = 0.5$, $\delta_i = 2$. Die Laufzeit beträgt nach Formel 4.16: $\Gamma \leq 3 \cdot \ell_1 + 2 \cdot \ell_2 + 1 \cdot \ell_3 - 3 = 6$. Die tatsächliche Laufzeit beträgt ebenfalls genau 6.

Im Algorithmus 4.1 auf Seite 42 wird in Zeile 2 die invertierten Liste eines Keyframes gebildet. Diese wird durch Vereinigung der invertierten Listen der Featurevektoren des Keyframes gebildet. Zwei Segmente, die direkt nebeneinander liegen, werden nicht durch ein größeres Segment ersetzt. Ebenso kann es wegen der Natur der invertierten Listen keine sich überlappenden Segmente geben. Also sind in zwei verschiedenen Keyframelisten Segmente entweder identisch oder sie überlappen sich nicht.

Nichtsdestotrotz können solche Worst-Case-Beispiele bei Experimenten auftreten. Mit der Wahl einer Stiffness $0 \leq \sigma < 1.0$ und Keyframeabständen $\delta_i > 1$ lassen sich ebenfalls Worst-Case-Beispiele konstruieren, in denen auch mit der verwendeten Datenbank die entsprechenden Laufzeiten auftreten. Um zu demonstrieren, dass die Konstruktion solcher Beispiele möglich ist, wird in Abbildung 4.11 ein Worst-Case-Szenario mit $K = 3$ Keyframes, $\sigma = 0.5$ und $\delta_1 = \delta_2 = 2$ gegeben. Aus der bei dieser Datenbank verwendeten Aufrufreihenfolge von Algorithmus 4.1, dargestellt in Auflistung 4.5 auf Seite 58, geht hervor, dass die tatsächliche Laufzeit und die berechnete Laufzeit auf dem Wert $\Gamma = 6$ übereinstimmen: Es erfolgen genau 4 Aufrufe der rekursiven Funktion und es werden zwei Matches gefunden.

In der Praxis werden in einer Anfrage Keyframes bezüglich unterschiedlicher Featuremengen benutzt. Die unterschiedlichen Featuremengen führen zu verschiedenen F-Segmentierungen, man betrachtet hier die Datenbank also bezüglich verschiedener F-Segmentierungen. Damit können Überlappungen der Segmente auftreten Mit der Verwendung unterschiedlicher Featuremengen wird jedoch die Ausdrucksfähigkeit der Keyframes stark verbessert und in der Praxis treten kürzere Laufzeiten auf.

4.2.4 Verbesserung der praktischen Laufzeit

Durch die F-Segmentierung wird die Datenbank in einzelne Segmente zerstückelt. In vielen Fällen kann es dazu kommen, dass die invertierten Listen der Keyframes mehrere in der Datenbank nebeneinander liegende Segmente enthalten, wie Abbildung 4.12 zeigt. Dies ist insofern ineffizient, als dass Algorithmus 4.1 für jedes einzelne Segment eine Berechnungskette starten könnte. Man kann den zerstückelten Bereich durch ein einzelnes Segment ersetzen und dadurch die Längen der invertierten Listen deutlich reduzieren. Dies kann durch einen einfachen Algorithmus der Laufzeit $O(M)$ geschehen, indem in jeder invertierten Liste diejenigen Segmente vereinigt werden, deren Framepositionen direkt nebeneinander liegen. Dieser Vorgang werde „Nachsegmentierung“ genannt.

Zu dieser Zerstückelung kann es dadurch kommen, dass als Keyframes Mengen von Featurevektoren zugelassen sind. Angenommen, es befinden sich vier Segmente $S_1 = (1, 2)$, $S_2 =$

Beispiel 4.5: Befehlsabfolge von Algorithmus 4.1 bei Verwendung der Datenbank aus Abbildung 4.11

$p_1 = 1$
allowedRange $= \mu_{0.5,2}(1,2) = (1 + \lceil 0.5 \cdot 2 \rceil, 2 + \lfloor \frac{1}{0.5} \cdot 2 \rfloor) = (1+1, 2+4) = (2,6)$
$\hookrightarrow$ (2, allowedRange)
    pointerIncremented = false; intersection $= (4,7) \cap (2,6) = (4,6)$
    newAllowedRange $= \mu_{0.5,2}(4,6) = (4 + \lceil 0.5 \cdot 2 \rceil, 6 + \lfloor \frac{1}{0.5} \cdot 2 \rfloor) = (4+1, 6+4) = (5,10)$
    $\hookrightarrow$ (3, newAllowedRange)
        pointerIncremented = false; intersection $= (8,12) \cap (5,10) = (8,10)$
        matches = matches $\cup\, p$ = matches $\cup\, (1,1,1) = \{(1,1,1)\}$
        $p_3 = 2$; pointerIncremented = true; intersection $= \emptyset$
        $p_3 = 2 - 1 = 1$;
        $\hookleftarrow$
    $p_2 = 2$; pointerIncremented = true; intersection $= \emptyset$
    $p_2 = 2 - 1 = 1$;
    $\hookleftarrow$
$p_1 = 2$
allowedRange $= \mu_{0.5,2}(3,4) = (3 + \lceil 0.5 \cdot 2 \rceil, 4 + \lfloor \frac{1}{0.5} \cdot 2 \rfloor) = (3+1, 4+4) = (4,8)$
$\hookrightarrow$ (2, allowedRange)
    pointerIncremented = false; intersection $= (4,7) \cap (4,8) = (4,7)$
    newAllowedRange $= \mu_{0.5,2}(4,7) = (4 + \lceil 0.5 \cdot 2 \rceil, 7 + \lfloor \frac{1}{0.5} \cdot 2 \rfloor) = (4+1, 7+4) = (5,11)$
    $\hookrightarrow$ (3, newAllowedRange)
        pointerIncremented = false; intersection $= (8,12) \cap (5,11) = (8,11)$
        matches = matches $\cup\, p$ = matches $\cup\, (1,1,2) = \{(1,1,1),(1,1,2)\}$
        $p_3 = 3$; pointerIncremented = true; intersection $= \emptyset$
        $p_3 = 3 - 1 = 2$;
        $\hookleftarrow$
    $\hookleftarrow$
    $p_2 = 2$; pointerIncremented = true; intersection $= \emptyset$
    $p_2 = 2 - 1 = 1$;
    $\hookleftarrow$
Ende der ersten Liste erreicht.

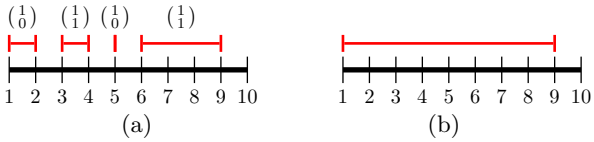


Abbildung 4.12. Abbildung **(a)** zeigt einen Ausschnitt der Datenbank, in dem die invertierte Liste eines Keyframes mehrere direkt nebeneinander liegende Segmente enthält. Effizienter ließen sich diese Segmente der invertierten Liste durch ein einziges Segment wie in **(b)** ausdrücken.

$(3,4)$, $S_3 = (5,5)$ und $S_4 = (6,9)$ in der F-Segmentierung der Datenbank, die den Featurevektoren $v_1 = \binom{1}{0}$ und $v_2 = \binom{1}{1}$ wie in Abbildung 4.12(a) dargestellt zugeordnet werden. Es sei ein Keyframe als $V = \{\binom{1}{0}, \binom{1}{1}\}$ gegeben. Dann ist die invertierte Liste des Keyframes Λ_V durch

$$\Lambda_V = L(\tbinom{1}{0}) \cup L(\tbinom{1}{1})$$

gegeben und Λ_V enthält damit die Segmente S_1, S_2, S_3 und S_4, welche direkt nebeneinander liegen. Diese Segmente werden besser durch ein einzelnes Segment wie in Abbildung 4.12(b) repräsentiert, was durch eine Nachsegmentierung erreicht werden kann.

Man kann sich leicht vorstellen, dass die potentielle Anzahl von solchen redundanten Darstellungen mit der Anzahl der Featurevektoren in einem Keyframe wächst. Je unspezifischer, je allgemeiner ein Keyframe also beschrieben wird, desto stärker ist die Verbesserung, die durch die Nachsegmentierung erreicht wird. In allen Experimenten in dieser Arbeit wird die Nachsegmentierung der invertierten Listen angewendet.

4.3 Vorhergehender Ansatz

In [Mül07] und [Röd06] wurde bereits ein Algorithmus vorgeschlagen, wie mit Hilfe von Keyframes eine Vorauswahl von Treffern bestimmt werden kann. Die vorgeschlagene Technik sucht nach Abschnitten in der Datenbank, welche die Keyframes in der vorgegebenen Reihenfolge enthalten. Des Weiteren wird eine Zeitschranke eingehalten, welche bestimmt, wie viele Frames der erste und der letzte Keyframe in der Datenbank auseinander liegen dürfen. Bei gegebenen Keyframes $V_1, \ldots, V_K$ und Zeitschranke Θ wurde ein *Keyframe-Treffer* definiert als ein Tupel $(r, s) \in [1 : M]^2$, so dass eine aufsteigende Folge von Frames $r = t_1 < t_2 < \ldots < t_K = s$ existiert, welche die Bedingung

$$\forall k \in [1 : K] : F(D(t_k)) \in V_k \quad \textit{und} \quad t_K - t_1 + 1 \leq \theta$$

erfüllt. Da es eine große Zahl von Keyframe-Treffern geben kann, die sich stark überlappen und alle den gleichen Bereich beschreiben, wird das Konzept des *reduzierten Keyframe-Treffers* eingeführt. Ein Keyframe-Treffer (r, s) wird reduziert genannt, wenn $F(D(r+1)) \notin V_1$ und s minimal bezüglich aller mit r beginnenden Keyframe-Treffern ist. Weiterhin wird eine technische Voraussetzung an die Keyframes gemacht: Die Keyframes müssen *zulässig* sein:

$$\forall k \in [1 : K-1] : V_k \cap V_{k+1} = \emptyset \tag{4.17}$$

Der Algorithmus benutzt Zeiger in die invertierten Listen, die die Segmente der Features der Datenbank enthalten. Es wird ein linearer Sweep mit den Zeigern durch die Listen beschrieben, in dem garantiert wird, dass alle reduzierten Keyframe-Treffer gefunden werden. Die Laufzeit- und Speicheranforderungen hängen linear von der Summer der Längen der invertierten Listen $\sum_{k=1}^{K} \ell_k$ ab. Ein Vergleich zwischen den Eigenschaften dieses und dem hier vorgestellten Algorithmus findet in Abschnitt 4.4 statt.

4.4 Vergleich mit vorherigem Ansatz

Im Folgenden wird der in Abschnitt 4.3 erwähnte Algorithmus mit A bezeichnet. Der in dieser Diplomarbeit entwickelte Algorithmus wird B genannt.

Algorithmus B der keyframebasierten Suche stellt eine Erweiterung des Algorithmus A dar. Anstatt einer globalen Zeitschranke für den Abstand des ersten und letzten Keyframes werden die gewünschten Abstände zwischen zwei aufeinander folgenden Keyframes angegeben (siehe Abbildung 4.13): Bei 4 Keyframes V_1, V_2, V_3 und V_4 sind das die Abstände δ_k (Abstand zwischen V_k und V_{k+1}) für $k \in [1 : 3]$ (siehe Abbildung 4.13 (b)). Algorithmus B findet aufsteigende Sequenzen von Frames $t_1 < t_2 < \ldots < t_K$, so dass $\forall k \in [1 : K] : F(D(t_k)) \in V_k$

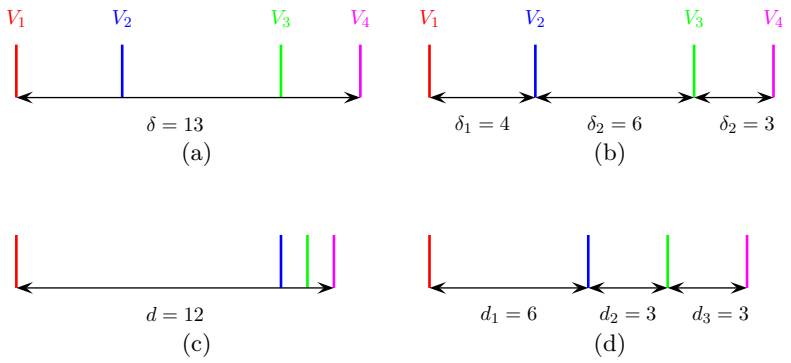


Abbildung 4.13. In **(a)** ist dargestellt, dass im Algorithmus A zur Bildung der globalen Zeitschranke Θ nur der Abstand zwischen dem ersten und dem letzten Keyframe berücksichtigt wird: $\Theta = \Theta(\delta)$. Abbildung **(b)** zeigt, dass im Algorithmus B beim Suchen der Matches die Abstände zwischen je zwei aufeinander folgenden Keyframes berücksichtigt werden. Die Abstände d_1, d_2 und d_3 der Frames bei den in der Datenbank gefundenen Hits weichen dann von δ_1, δ_2 und δ_3 nur um einen bestimmten Faktor ab.
In den Abbildungen **(c)** und **(d)** ist jeweils dargestellt, welche Abstände die Keyframes bei einem möglichen Match haben können. Abbildung **(c)** illustriert, dass mit Algorithmus A degenerierte Matches, bei denen die Keyframes an semantisch nicht sinnvollen Positionen auftreten, nicht vermieden werden können. Setzt man bei Algorithmus B fest, dass die Abstände d_1, d_2 und d_3 nur um den Faktor 0.5 von δ_1, δ_2 und δ_3 abweichen dürfen, so ist ein Match wie in **(c)** dargestellt, nicht möglich. Es werden dann nur noch Matches wie in **(d)** gezeigt gefunden: Hier weicht d_2 maximal um den Faktor 0.5 von δ_2 ab.

und die Abstände $d_k = t_{k+1} - t_k$ nur um einen Faktor σ von δ_k abweichen. Als Nachteil von Algorithmus B kann man angeben, dass der Faktor eine weitere Eingabe des Algorithmus darstellt und somit zusätzlichen Aufwand für den Benutzer beim Einstellen des „richtigen“ Parameters für seine Berechnungen bedeutet. Dabei hängt die Anzahl der gefundenen Matches und die Qualität der Ergebnisse von der Wahl dieses Faktors ab. Man kann diesen Parameter in einer Abwägung zwischen Anzahl und Qualität der Ergebnisse auf einem konstanten Wert festlegen, eine manuelle Einstellung dieses Faktors kann jedoch die Retrieval-Ergebnisse stark verbessern. Im Kapitel 6 wird ein automatisches Verfahren vorgestellt, wie dieser Parameter für eine Menge von Trainingsbewegungen optimiert werden kann.

Die Berücksichtigung der Abstände zwischen den einzelnen Keyframes hat den Vorteil gegenüber Algorithmus A, dass die Keyframes eine stärkere Semantik erhalten. Es ist ebenfalls möglich, mit einer Reihe von Keyframes Bewegungssequenzen zu beschreiben. Sind beispielsweise Sequenzen aus drei Bewegungen gesucht, die einen Radschlag am Anfang und Ende der Sequenz und in der Mitte eine beliebige Bewegung enthalten, so lassen sich für die Radschläge Keyframes angeben. Durch die Definition der Abstände zwischen den Keyframes lässt sich beschreiben, dass die Radschläge eine gewisse Zeit auseinander liegen sollen, in der dann eine beliebige andere Bewegung stattfinden kann. Mit Algorithmus A ließe sich nicht vermeiden, dass auch Treffer gefunden werden, bei denen die zwei Radschläge direkt aufeinander folgen.

Ein weiterer Vorteil von Algorithmus B ist, dass keine Voraussetzungen an die Keyframes gemacht werden. Die Bedingung, dass die Keyframes zulässig sein müssen (Gleichung (4.17)), fällt weg. Trotzdem kann die Korrektheit des Algorithmus bewiesen werden (Abschnitt 4.2.2).

Sind bei K Keyframes die Längen der invertierten Listen mit $\ell_1, \ldots, \ell_K$ bezeichnet, so kann Algorithmus A eine garantierte Laufzeit von

$$\Gamma_A \leq \sum_{k=1}^{K} \ell_k$$

vorweisen. Dagegen besteht im Algorithmus B noch eine Abhängigkeit der Worst-Case-Laufzeit von der Anzahl der Keyframes. Es gilt

$$\Gamma_B \leq K \cdot \sum_{k=1}^{K} \ell_k,$$

wie in Abschnitt 4.2.3 diskutiert wurde.

Bei genauer Betrachtung stellt man fest, dass die Zulässigkeitsbedingung an die Keyframes eine recht starke Bedingung ist, die in der Praxis oft nicht eingehalten werden kann. Es sollen beispielsweise mit $f = 5$ Features zwei Keyframes ausgewählt werden. Bei V_1 sei der vierte Featurewert nicht relevant, bei V_2 sei der erste Featurewert nicht relevant. Alle anderen Featurewerte seien festgelegt. Es ergeben sich die beiden Keyframes

$$V_1 = \left\{ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\}, \; V_2 = \left\{ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 0 \end{pmatrix} \right\}.$$

Man stellt fest, dass diese beiden Keyframes einen nichtleeren Schnitt haben und daher die Zulässigkeitsbedingung nicht erfüllen. Es ist leicht einsehbar, dass je „schwammiger“ die Keyframes definiert werden, desto schwerer die Zulässigkeitsbedingung einhaltbar ist. Man kann allerdings manchmal durch manuelle „Nachjustierung“ der Anfragen die Zulässigkeitsbedingung erzwingen, sofern die Keyframes nur bezüglich einer festen Featurefunktion definiert wurden. In der Praxis ist es ein Vorteil von Algorithmus B, dass diese Bedingung nicht für die Korrektheit eingehalten werden muss.

Experimentelle Vergleiche zwischen den beiden Algorithmen werden hier nicht vorgestellt, da die in dieser Arbeit verwendeten Keyframes die Zulässigkeitsbedingung nicht erfüllen und somit nicht mit Algorithmus A verwendet werden können. Um die Ausdrucksfähigkeit der Keyframes zu erhöhen und mehr Flexibilität beim Erstellen der Keyframes zu haben, wurde darauf verzichtet, die Zulässigkeitsbedingung einzuhalten.

Kapitel 5

Experimente zur keyframebasierten Bewegungssuche

Um zu überprüfen, wie sich der in dieser Arbeit vorgestellte Algorithmus (Algorithmus 4.1, Seite 42), in der Praxis verhält, werden ausgedehnte Experimente durchgeführt. Alle Experimente werden auf einem Laptop mit Intel® Pentium®M Prozessor mit 1500 Mhz und 1.25 GB RAM durchgeführt. Der Algorithmus 4.1 wurde in der Programmiersprache Matlab® 7 implementiert, wobei Teile des Algorithmus aus Effizienzgründen als C-DLL in Matlab® eingebunden worden sind.

Zuerst wird die in den Experimenten benutzte Menge von relationalen Features in Abschnitt 5.1 beschrieben. Abschnitt 5.2 geht auf die Gewinnung der für den Algorithmus benötigten Menge von Keyframes ein. Schließlich werden die benutzten Keyframes in Abschnitt 5.3 systematisch auf ihre Qualität untersucht. Den Abschluss des Kapitels bildet der Abschnitt 5.4 mit einigen Anwendungsszenarien auf realistischen Datensätzen.

5.1 Entwurf von Features

Die Experimente in dieser Diplomarbeit werden mit einer Featuremenge aus $f = 39$ relationalen Features durchgeführt. Diese Features haben sich in der Praxis als geeignete Merkmale zur Beschreibung von Ganzkörperbewegungen bewährt. Die Features wurden in der Arbeit [MR06] entwickelt und beschrieben. Für die technischen Details des Featuredesigns wird auf die Arbeit [MRC05], [Mül07] und [Röd06] verwiesen. Es wurden generische Features gebildet, die mit Gelenknamen und Daten wie Winkeln und Abständen parametrisiert werden können.

Als repräsentierendes Beispiel aus der Menge der Features soll hier die Semantik des generischen Features $F_{\theta,\text{plane}}^{(j_1,j_2,j_3;j_4)}$ vorgestellt werden: Dieses Feature testet den Abstand eines Punktes zu einer Ebene. Ist der Abstand größer als der definierte Schwellwert Θ, so nimmt das Feature den Wert 1 an, ansonsten ist der Featurewert 0. Für die Parameter j_i können Gelenke des Standardskelettes angegeben werden. Drei Parameter j_1, j_2 und j_3 spannen eine Ebene $\langle p_1, p_2, p_3 \rangle$ auf. Es wird ein gerichteter Abstand des Gelenkes j_4 zu dieser Ebene berechnet. Mit dem Schwellwert Θ kann nun eine Entscheidungsschwelle festgelegt werden, wann das Feature den Wert 1 und wann es den Wert 0 annimmt. Durch das Einsetzen verschiedener Gelenkwinkel als Parameter können aus diesem generischen Feature verschiedene konkrete Features konstruiert werden. Für j_1 = „root“, j_2 = „lhip“ und j_3 = „lankle“ wird

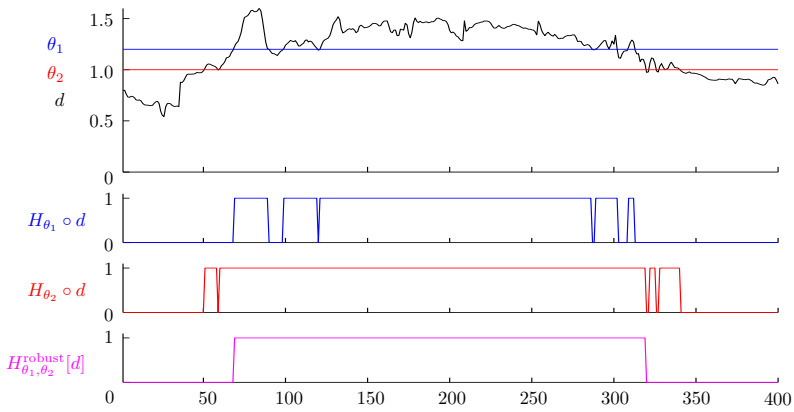


Abbildung 5.1. Oben: Distanzverlauf d eines Abstandfeatures. Die blaue Linie zeigt den größeren ($\theta_1 = 1.2$), die rote Linie den kleineren Schwellwert ($\theta_2 = 1$). **Mitte**: Schwellwertbildung des Distanzverlaufes mit dem kleinen und separat mit dem größeren Schwellwert. **Unten**: Robuste Schwellwertbildung unter Berücksichtigung beider Schwellwerte (Magenta). Abbildung entnommen aus [Röd06].

beispielsweise eine Ebene aufgespannt, in welcher das linke Bein liegt. Für $j_4 =$ „rankle“, und $\Theta = 0$ kann nun getestet werden, ob der rechte Fuß sich hinter dem linken Bein befindet oder nicht (siehe Abbildung 2.4). Wählt man andere Parameter, so lässt sich beispielsweise herausfinden, ob eine Hand nach vorne ausgestreckt ist, oder nicht.

Bei dieser Art von Features kann ein Problem auftreten, wenn der Abstand zu der Ebene nahe am Schwellwert liegt. In diesem Fall kann der Wert des Features sehr schnell zwischen den Werten 0 und 1 hin- und herspringen, so dass ein fast zufälliges Wechseln des Featurewertes möglich ist. Um diese Artefakte zu minimieren, kann man eine robuste Schwellwert-Strategie verwenden, die zwei Schwellwerte Θ_1 und Θ_2 mit $\Theta_1 > \Theta_2$ benutzt. Ist der Featurewert 0, so wechselt der Wert erst bei Überschreiten von Θ_2 auf 1. Ist der Featurewert 1, so wechselt der Featurewert erst bei Unterschreiten von Θ_1 auf 0. So wird erreicht, dass wenn sich der Abstand zwischen den beiden Schwellwerten befindet, der vorhergehende Featurewert beibehalten wird. Dieser Prozess ist auch in Abbildung 5.1 veranschaulicht. Dort wird gezeigt, wie durch robuste Schwellwertbildung eine mehr oder weniger zufällige Fluktuation der Featurewerte vermieden werden kann.

Die verwendeten Features und ihre anschaulichen Beschreibungen sind in Tabelle 5.1 aufgelistet. Aus technischen Gründen wird innerhalb diese Featuremenge eine weitere Unterteilung zwischen Features in der unteren Körperregion („lower, ℓ“), Features am Oberkörper („upper“, „u“) und gemischten Features für den ganzen Körper („mix“, „m“) vorgenommen. Eine Unterteilung ist notwendig, da eine Featuremenge von $f = 39$ Features beim Indexierungsschritt der Datenbank eine Anzahl von $2^{39} \approx 5 \cdot 10^{11}$ invertierten Listen induzieren würde. Diese Anzahl von Listen ist auch mit modernen Rechnern nicht mehr handhabbar. Für die Angabe von Keyframes ist es ebenfalls von Vorteil, Featuremengen mit moderater Größe zu benutzen. Will man nämlich in einer Featuremenge der Größe $f = 39$ nur 5 Features einen Wert 0 oder 1 zuweisen und die restlichen Features unspezifiziert lassen, so müssen, um die invertierte Liste dieses Keyframes zu bilden, $2^{39-5} \approx 10^{10}$ invertiere Listen vereinigt werden.

Für eine Featuremenge der Größe $f = 12$ müssen 4096 invertierte Listen verwaltet werden.

ID	Set	Beschreibung
F_1/F_2	ℓ	Ist der rechte/linke Fuß hinter dem linken/rechten Bein?
F_3/F_4	ℓ	Ist der rechte/linke Fuß angehoben?
F_5	ℓ	Sind die Füße seitwärts weit auseinander?
F_6/F_7	ℓ	Ist das rechte/linke Knie gebeugt?
F_8	ℓ	Sind die Füße gekreuzt?
F_9	ℓ	Bewegen sich die Füße seitwärts aufeinander zu?
F_{10}	ℓ	Bewegen sich die Füße seitwärts voneinander weg?
F_{11}/F_{12}	ℓ	Ist der rechte/linke Fuß schnell
F_{13}/F_{14}	u	Bewegt sich die Rechte/linke Hand nach vorne?
F_{15}/F_{16}	u	Ist die recht/linke Hand über dem Nacken?
F_{17}/F_{18}	u	Bewegt sich die rechte/linke Hand nach oben?
F_{19}/F_{20}	u	Ist der reche/linke Ellenbogen gebeugt?
F_{21}	u	Sind die Hände seitwärts weit auseinander?
F_{22}	u	Bewegen sich die Hände aufeinander zu?
F_{23}/F_{24}	u	Bewegt sich die rechte/linke Hand vom Körper weg?
F_{25}/F_{26}	u	Bewegt sich die rechte/linke Hand schnell?
F_{27}/F_{28}	m	Ist der rechte/linke Oberarm angelegt?
F_{29}/F_{30}	m	Ist der rechte/linke Oberschenkel angezogen?
F_{31}	m	Ist der Körperschwerpunkt nach hinten verlagert?
F_{32}	m	Ist die Wirbelsäule horizontal?
F_{33}/F_{34}	m	Ist die rechte/linke Hand abgesenkt?
F_{35}/F_{36}	m	Sind die Schultern in Bezug auf die Hüfte nach rechts/links gedreht?
F_{37}	m	Liegt der höchste Punkt des Körpers nahe beim tiefsten Punkt?
F_{38}	m	Ist der rechteste Punkt des Körpers weit entfernt vom linkesten Punkt?
F_{39}	m	Bewegt sich der Körperschwerpunkt schnell?

Tabelle 5.1. Die benutze Featuremenge F besteht aus $f = 39$ relationalen Features. Die technische Realisation der Features ist in [MR06] beschrieben. Hier sollen nur die geometrisch anschaulichen Beschreibungen der Features wiedergegeben werden. Die Spalte „Set“ gibt an, in welcher Teil-Featuremenge sich das Feature befindet: „ℓ“ („lower“) steht für Bewegungsmerkmale in der unteren Körperhälfte, „u“ („upper“) steht für Bewegungsmermale am Oberkörper und „m“ („mix“) beschreibt verschiedene, gemischte Merkmale am ganzen Körper.

Werden in einem Keyframe nur 5 Werte spezifiziert und 7 nicht angegeben (oder gleichwertig auf den Wert 0.5 festgelegt), so müssen lediglich 128 invertierte Listen vereinigt werden, um die invertierte Liste dieses Keyframes zu bilden. Die Größe der Featuremenge ist $f_{\text{lower}} = 12$, $f_{\text{upper}} = 14$ und $f_{\text{mix}} = 13$.

Es macht jedoch keinen Sinn, die Featuremengen noch stärker zu unterteilen, denn je kleiner eine Featuremenge wird, desto länger werden die zu verwaltenden invertierten Listen. Im Extremfall, in dem nur ein einzelnes Feature in einer Featuremenge vorhanden ist, enthalten die zwei invertierten Listen jeweils ungefähr die Hälfte der Segmente der gesamten Datenbank. Wird eine Menge von $f = 39$ Features auf diese Weise unterteilt, so sind lediglich $2 \cdot 39 = 78$ invertierte Listen zu verwalten, die aber jeweils die Hälfte aller Segmente der Datenbank enthalten. Möchte man nun einen Keyframe definieren, der 5 Features auf einen bestimmten Wert setzt, so muss der Schnitt von 5 sehr langen invertierten Listen gebildet werden. Dieser Vorgang kostet aufgrund der Länge der Listen und der Tatsache, dass sich die Segmente überlappen können, sehr viel Rechenzeit.

Mit einer Anzahl der Features von $12-14$ ist eine gute Größe gefunden, die akzeptable Rechenzeiten bei Durchschnitt und Vereinigung der Listen erlauben, sowie den Verwaltungsaufwand für die invertierten Listen in einem vernünftigen Rahmen hält.

5.2 Entwurf von Keyframes

Die für die Experimente verwendeten Keyframes werden in einem semi-automatischen Verfahren generiert. Es sei darauf hingewiesen, dass die Gewinnung der Keyframes, wie sie in dieser Arbeit geschehen ist, keineswegs ein Rezept für die automatische Gewinnung von Keyframes sein kann: In dem Gewinnungsprozess werden sowohl die Trainings- als auch die Evaluationsdatenbank betrachtet, um die Keyframes möglichst aussagekräftig für die Gesamtdatenbank zu gestalten und somit eine Menge von gut funktionierenden Keyframes zu gewinnen. In diesem Schritt ging es darum, die Mächtigkeit des Algorithmus 4.1 zu untersuchen. Um Aussagen über die Qualität des Algorithmus treffen zu können, ist es notwendig, für die Anfragen an die Datenbank „gute“, also aussagekräftige Keyframes für die Bewegungsklassen zu haben. Daher wird in dem manuellen Optimierungsschritt der Keyframes die Evaluationsdatenbank mit einbezogen. Ein erster Ansatz zum automatischen Erstellen von Keyframes anhand von Beispielbewegungen wird in Kapitel 6 vorgestellt.

Wie in Abschnitt 4.1.2 beschrieben wurde, ist ein Keyframe modelliert als eine Menge von Featurevektoren. Im Folgenden ist diese Menge von Featurevektoren durch eine implizite Notation gegeben. Es sei $y \in \{0,1\}^f$ ein Featurevektor und $v \in \{0, 0.5, 1\}^f$ eine implizite Repräsentation eines Keyframes. Es sei $I(v) := \{i \in [1:f] | v_i \neq 0.5\}$ die Menge der Positionen in v, die nicht den Wert 0.5 enthalten. Dann kann zwischen y und v eine Beziehung hergestellt werden durch

$$y \Subset v \ :\Leftrightarrow \ y_{I(v)} = v_{I(v)}.$$

Der Keyframe V, der zu der impliziten Repräsentation v korrespondiert, ist dann durch

$$V_j := \left\{ y \in \{0,1\}^f \mid y \Subset v \right\}$$

gegeben. Enthält die implizite Repräsentation des Keyframes also drei 0.5-Werte, so enthält der Keyframe V genau $2^3 = 8$ Featurevektoren. Ist beispielsweise $v = (0, 0, 0.5, 0, 1, 0.5)^T$, so ist der entsprechende Keyframe gegeben durch

$$V = \left\{ \begin{pmatrix} 0\\0\\0\\0\\1\\0 \end{pmatrix}, \begin{pmatrix} 0\\0\\1\\0\\1\\0 \end{pmatrix}, \begin{pmatrix} 0\\0\\0\\0\\1\\1 \end{pmatrix}, \begin{pmatrix} 0\\0\\1\\0\\1\\1 \end{pmatrix} \right\}.$$

Zwischen einem Keyframe und der impliziten Repräsentation wird im Folgenden nicht unterschieden, da klar ist, welche Repräsentation gemeint ist. Ist ein Keyframe als Menge von boolschen Featurevektoren gegeben, so ist die explizite Repräsentation gewählt. Wird für einen Keyframe ein einzelner Vektor $\in \{0, 0.5, 1\}^f$ angegeben, so ist die implizite Repräsentation gemeint.

Der Prozess der semi-automatischen Auswahl läuft wie folgt ab: Es werden für jede Bewegungsklasse der D^{57} aus dem harten MT Keyframes ausgewählt. Die harten MTs wurden mit der Trainingsmenge der Bewegungsdatenbank $\mathcal{D}^{57}$ berechnet. Diejenigen Frames werden als Keyframes ausgewählt, die möglichst konsistent unter der Trainingsmenge sind und welche die Klasse möglichst gut von anderen Klassen separieren. Die Keyframes werden dann so lange vergröbert (d.h. Einsen oder Nullen werden mit 0.5-Einträgen überschrieben), bis keine Bewegung der Evaluationsdatenbank durch die Keyframes verloren geht. Dadurch wird aber in den meisten Fällen die Anzahl der zurückgelieferten Treffer, die nicht in der Klasse liegen,

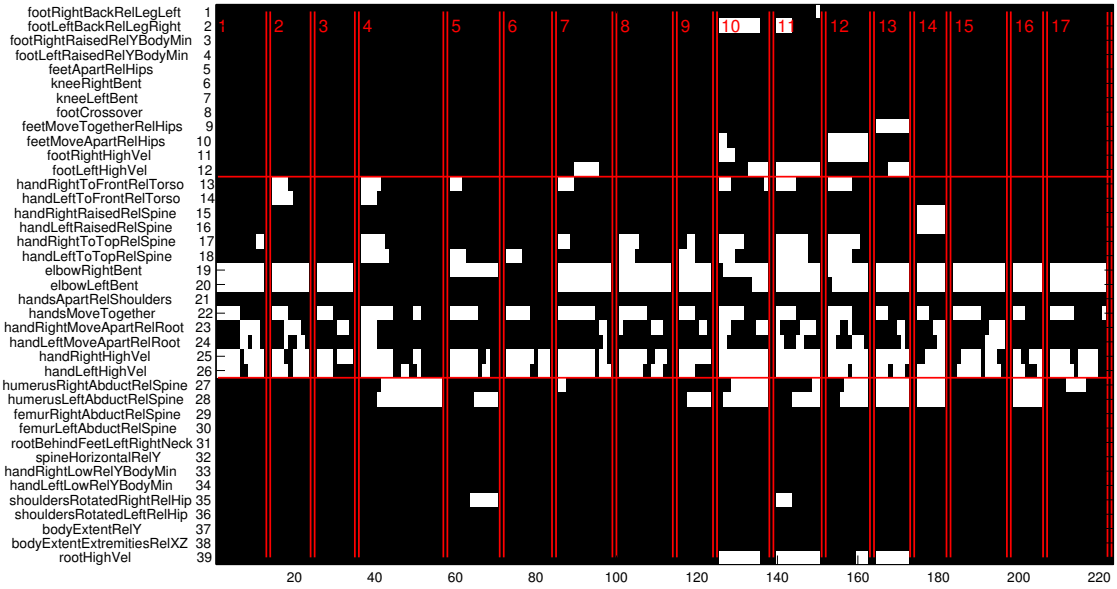


Abbildung 5.2. Die Featurematrizen aller `clap1Reps`-Bewegungen sind in dieser Abbildung dargestellt. Durch 2 senkrechte rote Striche ist das Ende einer Bewegung angedeutet. Die Trennung zwischen den Featuremengen „lower“, „upper“ und „mix“ ist durch zwei waagerechte, rote Linien angedeutet. Die Beschriftung der Y-Achse ist mit den Featurebeschreibungen in Tabelle 5.1 konform.

erhöht. Anschließend werden Keyframes hinzugefügt, um die Gesamtanzahl der gefundenen Treffer zu verringern. Dabei wird teilweise in Kauf genommen, dass die Anzahl der relevanten Treffer wieder sinkt, wenn durch den zusätzlichen Keyframe eine starke Verkleinerung der Treffermenge erreicht wird. Dieser Prozess wird so lange iteriert, bis möglichst viele Treffer der beabsichtigten Bewegungsklasse gefunden werden und möglichst wenig Treffer in andere Klassen fallen. Dabei wird insgesamt darauf geachtet, dass nicht zu viele 0.5-Einträge und möglichst viele 1-Einträge in einem Keyframe vorliegen. Wie in der Bemerkung 4.1 angedeutet kann der Stiffness-Parameter abhängig vom Keyframe gewählt werden. Bei der Gewinnung der Keyframes wurde der Stiffness-Parameter für je zwei aufeinander folgende Keyframes so angepasst, dass die Charakteristik der Bewegung möglichst gut repräsentiert wird.

Je weniger 0.5-Einträge ein Keyframe enthält, desto weniger invertierte Listen müssen im Algorithmus 4.1 vereinigt werden und desto größer ist die Chance, dass die invertierte Liste eines Keyframes kurz ist und somit zu kurzen Laufzeiten beiträgt. Ein 1-Eintrag in einem Keyframes beschreibt, dass das entsprechende Feature 1 sein muss. Die Features sind so konstruiert, dass sie in der Standpose (Körper steht senkrecht mit geschlossenen Füßen, Arme sind waagerecht seitlich ausgestreckt) alle den Wert 0 annehmen. Die 1-Einträge unterscheiden somit einen Keyframe von der Standpose und leisten dadurch einen wesentlichen Beitrag zur Separierung einer Klasse.

Die Features mancher Bewegungsklassen sind schon innerhalb einer Klasse sehr unterschiedlich. Deutlich wird dies z.B. an der Klasse clap1Reps. Alle Bewegungen dieser Klasse in $\mathcal{D}^{57}$ sind in Abbildung 5.2 dargestellt. Welche Bewegung die untere Körperhälfte beim Klatschen ausführt, ist in der Beschreibung nicht definiert. Somit ist es klar, dass die Features der „lower“-Featuremenge inkonsistent innerhalb der Klasse sein können. Für die meisten Bewegungen der Klasse sind die „lower“-Features durchgehend 0. Die Features der Bewegungen 10

bis 13 enthalten jedoch die Featurewerte 1 in den Features F_1, F_2, F_9, F_{10}, F_{11} und F_{12}. Durch diese hohe Inkonsistenz innerhalb der Featuremenge müssen die Keyframes viele 0.5-Werte enthalten. Es fällt sofort auf, dass unter den Bewegungen in der „lower“-Featuremenge kein Feature konsistent unter allen Bewegungen eine 1 annimmt. Dadurch können in einem Keyframe auch keine Einsen vorkommen. Der Keyframe würde also keine charakteristische Pose beschreiben. Insgesamt sind so nur Keyframes wählbar, die nicht besonders charakteristisch für die Bewegung sind. Im Extremfall, in welchem in keinem der drei Featuremengen gemeinsame Einsen für die Wahl eines Keyframes gefunden werden können, kann die beschriebene Bewegung mit den Keyframes nicht von der Standpose unterschieden werden. Somit ist bei einer Anfrage mit solchen Keyframes eine Vielzahl von False Positives zu erwarten.

Die Klatsch-Bewegung kann schnell oder langsam ausgeführt werden, sie kann mit gebeugten oder gestreckten Ellenbogen ausgeführt werden und eventuell bewegen sich die Hände beim Klatschen nach oben. Auch kann man weit ausholen oder nur eine kleine Bewegung ausführen. Die wenigen Gemeinsamkeiten in der „upper“-Featuremenge (z.B. dass alle Klatsch-Bewegungen die Hände aufeinander zu bewegen) können zwar mit Keyframes ausgedrückt werden, nur müssen diese Keyframes dann durch die starken Inkonsistenzen in den anderen Features sehr viele 0.5-Werte enthalten, um keine False Negatives zu verursachen. Damit wird die Verwendung dieser Keyframes sehr ineffizient und sie verlieren außerdem durch die vielen 0.5-Werte an Charakteristik für die Bewegungsklasse. Die Ursache für die Probleme der Keyframeauswahl bei dieser Bewegung liegt in den verwendeten Features. Sie bieten nicht genug Charakteristik für Klatsch-Bewegungen. Dies ist auch daran zu erkennen, dass im Retrieval mit Motion Templates (siehe Tabelle 3.1) für die clap1Reps-Klasse über 500 Treffer zurückgegeben werden, wobei nur 6 der 8 relevanten Dokumente erkannt werden.

Um Bewegungen wie die Klatsch-Bewegungen clap1Reps beschreiben zu können, kann die Veränderung der Featuremenge (Hinzunahme neuer Features, Entfernen von Features, Verändern der Schwellwerte) einen wesentlichen Vorteil bringen. Die Veränderung der Featuremenge geht über den Rahmen dieser Diplomarbeit hinaus und soll hier nicht geschehen. Stattdessen können Bewegungen dieser Art als Beispiele dienen, wie sich der Algorithmus mit „schlechten“ und uncharakteristischen Keyframes verhält.

5.3 Experimente auf Bewegungsklassen

In diesem Abschnitt soll die Qualität der erstellten Keyframes untersucht werden, welche mit ihren Abständen und Stiffness-Parametern im Anhang B für jede Klasse abgebildet sind. Es ist wichtig, beurteilen zu können, ob die Keyframes eine Bewegungsklasse von anderen Bewegungen gut trennen können. Grenzen die Keyframes die Klasse gegenüber anderen Klassen nicht gut ab, so sind beim Retrieval viele False Positives zu erwarten. Außerdem wird dann durch die Keyframes die Größe der Datenbank eventuell nicht stark genug reduziert, so dass ein anschließender Ranking-Schritt auf der reduzierten Datenbank mit dem DTW-Ranking sehr lange dauern kann. Sind die Keyframes für eine Klasse zu restriktiv gewählt, so kann nicht erwartet werden, dass alle relevanten Treffer gefunden werden.

Das Retrieval-Verfahren ist in zwei Schritte unterteilt:

1. Mit der keyframebasierten Suche wird aus der Datenbank eine Vorauswahl von Treffern bestimmt und damit die Größe der Datenbank reduziert. Die reduzierte Datenbank ergibt sich dadurch, dass mit dem Operator Ω^Z (Definition 4.8) die vom Algorithmus 4.1 zurückgegebene Menge von Matches M in Zusammenhangskomponenten aufgespalten und auf die von den einzelnen Zusammenhangskomponenten überspannten Hitrelevanten Bereiche abgebildet wird. Mit der Bildung der Zusammenhangskomponenten wird vermieden, dass bei sich überlappenden Matches die Überlappungen mehrfach betrachtet werden müssen. In Tabelle 5.2 gibt die Spalte $\%(D)$ den Anteil der reduzierten Datenbank an der Gesamtdatenbank in Prozent wieder. Wie in Abschnitt 4.1.2 beschrieben wurde, benötigt die keyframebasierte Suche invertierte Listen, welche die Framepositionen der Segmente in der Datenbank enthalten müssen. Die benutzten invertierten Listen liegen in einer anderen Form vor und müssen erst konvertiert werden. Die Gesamtlaufzeit der keyframebasierten Suche ist in der Spalte t^K in Sekunden angegeben. Darin enthalten ist die Konvertierung der invertierten Listen, die Zeit des Algorithmus 4.1 und die Zeit für die Anwendung Operators Ω^Z. In Abschnitt 5.4.1 wird gezeigt, dass die Laufzeit für die keyframebasierte Suche von den Vorverarbeitungsschritten dominiert wird. Die Ausführung der rekursiven Funktion und Anwendung des Operators Ω^Z dauert nur wenige Millisekunden, die Vorverarbeitung der invertierten Listen liegt im Bereich von einer Sekunde. Die Entwicklung einer effizienteren Indexstruktur soll im Rahmen dieser Diplomarbeit nicht betrachtet werden.

2. Anschließend wird auf der reduzierten Datenbank eine Suche mit weichen MTs (wie in Abschnitt 3.4 beschrieben) und dem Schwellwert $\tau = 0.1$ durchgeführt. Das bedeutet, dass nur Treffer mit kleineren Kosten als τ gefunden werden. Schon beim Retrieval mit Motion Templates in Abschnitt 3.5, Tabelle 3.1 hat sich gezeigt, dass bei diesem Schwellwert nur in Ausnahmefällen relevante Dokumente nicht gefunden werden. Weil jedem Treffer Kosten in Bezug auf das weiche MT zugewiesen werden, entspricht dieser Schritt dem Herstellen einer Rangordnung der Treffer in der reduzierten Datenbank. Die dafür benötigte Zeit in Sekunden ist in der Spalte t^R aufgeführt.

Anhand der Tabelle 5.2 (Seite 70) kann nachvollzogen werden, wie gut die Keyframes für jede Klasse sind. Der Wert γ gibt an, wie viele relevanten Treffer die Evaluationsdatenbank für jede Klasse enthält. Um einen direkten Vergleich der Laufzeiten des keyframebasierten Retrievals und des rein motiontemplatebasierten Retrievals zu haben, sind hier mit t^T die Zeiten in Sekunden aufgelistet, die das Retrieval mit den Motion Templates (auch einzusehen in Tabelle 3.1) benötigt. Am wichtigsten für die Beurteilung der Trefferqualität ist die Angabe der False Negatives (#FN) und False Positives (#FP). Mit #FN wird gezeigt, wie viele relevante Treffer nach der Keyframesuche und dem Ranken nicht gefunden wurden. Dabei spielt das Ranken in diesem Fall nur die Rolle, dass Treffer, deren Kosten den Schwellwert $\tau = 0.1$ übersteigen, nicht berücksichtigt werden. Die Angabe der False Positives gibt einen Hinweis auf die Präzision der Ergebnisse. Sind viele Treffer fälschlicherweise als relevant erkannt worden, so ist die Menge der vorausgewählten Treffer groß (und damit die Präzision klein) und das Herstellen einer Rangordnung dieser Vorauswahl nimmt viel Zeit in Anspruch. Eine große Menge von False Positives tritt bei Verwendung von Keyframes, die nicht beschreibend genug für die Klasse der Bewegung sind, auf.

Bei der Beurteilung der Keyframes für jede Klasse stellt man mit Hilfe von Tabelle 5.2 fest, dass einige Klassen mit Keyframes nicht gut beschrieben sind, da sie eine hohe An-

Nr.	Bewegungsklasse	γ	$t^T[s]$	$t^K[s]$	$t^R[s]$	$\%(D)$	#FN	#FP
1	cartwheelLHandStart1Reps	10	4.59	0.01	0.10	2.68	0	0
2	clap1Reps	8	5.04	0.21	0.17	6.05	2	174
3	clapAboveHead1Reps	8	3.64	0.08	0.02	1.08	1	8
4	depositFloorR	16	5.79	0.11	0.29	10.00	1	12
5	depositHighR	14	7.02	0.14	0.11	3.67	2	10
6	elbowToKnee1RepsLelbowStart	13	3.04	0.01	0.02	1.27	0	0
7	elbowToKnee1RepsRelbowStart	13	2.93	0.02	0.02	1.36	0	0
8	grabFloorR	8	4.80	0.11	0.09	3.73	2	4
9	grabHighR	14	6.98	0.11	0.11	3.61	2	10
10	hopBothLegs1hops	18	5.70	0.12	0.03	1.72	1	4
11	hopLLeg1hops	20	5.37	0.04	0.02	1.11	0	0
12	hopRLeg1hops	21	5.28	0.08	0.03	1.44	1	3
13	jogLeftCircle4StepsRstart	8	5.47	0.02	0.03	1.48	0	1
14	jogOnPlaceStartFloor2StepsRStart	7	7.10	0.18	0.53	11.08	1	164
15	jogRightCircle4StepsRstart	8	4.79	0.01	0.02	1.14	1	1
16	jumpDown	7	6.75	0.05	0.05	1.66	0	0
17	jumpingJack1Reps	26	1.98	0.01	0.04	2.63	0	0
18	kickLFront1Reps	16	9.14	0.17	0.11	4.35	0	16
19	kickLSide1Reps	11	7.31	0.02	0.06	2.38	0	3
20	kickRFront1Reps	15	9.50	0.05	0.07	2.69	1	3
21	kickRSide1Reps	14	8.46	0.07	0.08	3.22	1	6
22	lieDownFloor	10	8.54	0.03	0.27	4.33	0	0
23	punchLFront1Reps	15	8.68	0.08	0.08	3.46	0	8
24	punchLSide1Reps	15	7.88	0.16	0.10	4.24	1	19
25	punchRFront1Reps	15	8.86	0.12	0.08	3.12	3	5
26	punchRSide1Reps	14	7.88	0.24	0.07	2.82	1	8
27	rotateArmsBothBackward1Reps	8	2.39	0.01	0.02	0.61	0	0
28	rotateArmsBothForward1Reps	8	2.57	0.04	0.01	0.87	0	2
29	rotateArmsLBackward1Reps	8	7.60	0.05	0.00	0.60	0	0
30	rotateArmsLForward1Reps	8	7.90	0.01	0.01	0.59	0	0
31	rotateArmsRBackward1Reps	8	6.92	0.02	0.02	0.70	0	0
32	rotateArmsRForward1Reps	8	7.47	0.01	0.00	0.58	0	0
33	runOnPlaceStartFloor2StepsRStart	7	6.26	0.15	0.02	0.75	0	4
34	shuffle2StepsRStart	6	8.55	0.15	0.03	1.27	0	3
35	sitDownChair	10	7.42	0.05	0.13	3.57	0	5
36	sitDownFloor	10	7.69	0.04	0.32	7.12	0	10
37	sitDownKneelTieShoes	8	8.68	0.01	0.20	3.40	0	0
38	sitDownTable	10	9.55	0.11	0.40	9.32	0	36
39	skier1RepsLstart	15	4.09	0.04	0.04	2.02	0	2
40	sneak2StepsLStart	8	9.23	0.04	0.05	1.31	0	1
41	sneak2StepsRStart	8	8.67	0.08	0.07	2.36	0	9
42	squat1Reps	26	2.55	0.02	0.08	3.48	0	0
43	staircaseDown3Rstart	7	7.93	0.07	0.09	3.84	0	15
44	staircaseUp3Rstart	14	6.48	0.03	0.09	2.68	0	0
45	standUpLieFloor	10	7.01	0.02	0.17	3.54	0	0
46	standUpSitFloor	10	5.71	0.07	0.29	7.16	0	16
47	throwBasketball	7	6.24	0.04	0.09	2.41	0	0
48	turnLeft	15	10.62	0.18	0.06	2.52	0	11
49	turnRight	14	11.06	0.30	0.07	2.51	2	9
50	walk2StepsLstart	15	8.92	0.07	0.07	3.47	0	16
51	walk2StepsRstart	15	8.78	0.08	0.11	4.90	1	27
52	walkBackwards2StepsRstart	7	7.98	0.06	0.03	0.99	0	0
53	walkLeft2Steps	8	6.11	0.01	0.06	1.77	0	1
54	walkLeftCircle4StepsRstart	9	5.77	0.06	0.08	2.42	0	2
55	walkOnPlace2StepsLStart	7	9.63	0.11	0.30	9.23	0	90
56	walkRightCircle4StepsRstart	7	5.76	0.06	0.07	1.77	0	1
57	walkRightCrossFront2Steps	8	6.83	0.04	0.06	1.82	0	0

Tabelle 5.2. Diese Tabelle zeigt die Retrieval-Zeiten auf der Datenbank $\mathcal{D}^{57}$ (Länge: ca. 20 Minuten, 30 Hz Abtastrate) zusammen mit der Auswertung der False Positives und False Negatives. γ: Anzahl der relevanten Dokumente. t^T: Zeit in Sekunden für das Retrieval mit weichen Motion Templates. t^K: Zeit in Sekunden für die keyframebasierte Suche. t^R: Zeit in Sekunden für das Ranken der reduzierten Datenbank mit $\tau = 0.1$ als Schwellwert. $\%(D)$: Anteil der reduzierten Datenbank an der Gesamtdatenbank in Prozent. #FN und #FP: False Negatives und False Positives bei der keyframebasierten Suche.

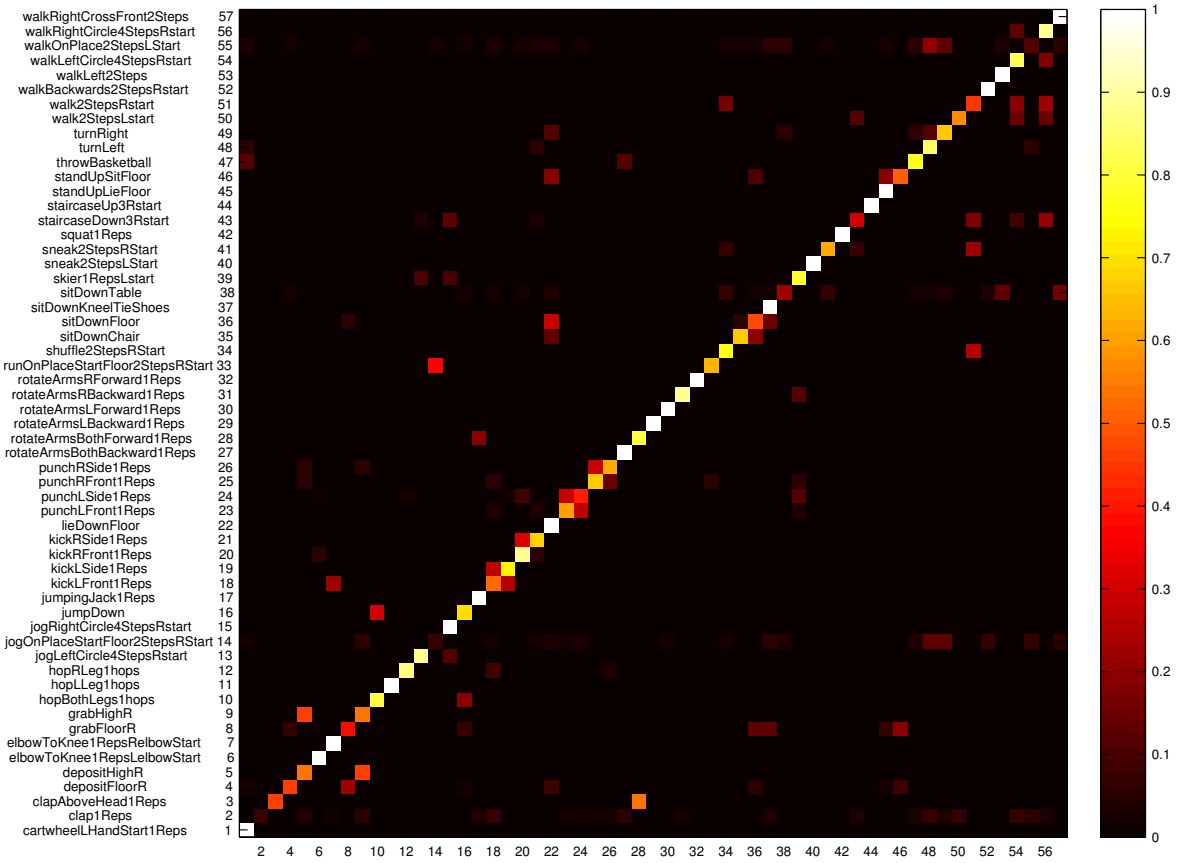


Abbildung 5.3. Confusion-Matrix für die Datenbank D^{57}. Für jede Anfrageklasse ist dargestellt, welcher Anteil der insgesamt zurückgegebenen Treffer in welcher Klasse liegt.

zahl von False Positives verursachen. Dies sind die Klassen clap1Reps, jogOnPlaceStartFloor2StepsRStart, sitDownTable und walkOnPlace2StepsLStart. Bei der Durchsicht der für diese Klassen verwendeten Keyframes (siehe Anhang B) fällt auf, dass diese sehr wenige Einsen enthalten. Die Keyframes der Klassen jogOnPlaceStartFloor2StepsRStart, sitDownTable und walkOnPlace2StepsLStart enthalten keine Eins. Bei diesen Klassen war es nicht möglich, gemeinsame charakteristische Eigenschaften in den Featurematrizen zu finden, ohne signifikante Einbußen bezüglich der Präzision machen zu müssen. In der Klasse clap1Reps sind zwei Keyframes mit jeweils drei Einsen vorhanden. Die beiden Keyframes sind jedoch identisch und liegen genau einen Frame auseinander: Sie kommen an Position 3 und 4 vor. Das bedeutet nur, dass die definierte Pose mehr als einen Frame anhalten muss. Dadurch, dass die Keyframes dieselbe Pose in einem zeitlich sehr nahen Abstand enthalten, verlieren sie an Ausdruckskraft. Zusätzlich ist die Pose nicht sehr klar spezifiziert: Durch hohe Inkonsistenzen in der „upper“-Featuremenge mussten 7 der 14 Features auf 0.5 gesetzt werden, um nicht zu viele relevante Treffer zu verlieren. Wie man in Tabelle 5.2 nachlesen kann, werden die Keyframes bei dieser Klasse so definiert, dass zwei relevante Treffer der Evaluationsdatenbank durch die Keyframes verloren gehen. Diese Einbuße in den Recall-Wert wurde eingegangen, damit nicht noch mehr Einträge in den Keyframes unspezifiziert gelassen werden müssen und die Precision weiter sinkt.

Zur weiteren Beurteilung der Keyframes wird ein Diagramm (Abbildung 5.3) erstellt, welches für jede Anfrage mit Keyframes zeigt, wie groß der Anteil derjenigen Klassen an den insgesamt zurückgegebenen Treffern ist, der in einer bestimmten Klasse liegt. Dieser Typ von Diagramm wird auch Confusion-Matrix genannt. Das Diagramm ist wie folgt definiert: Es sei $\mathcal{C} = \{C_1, \ldots, C_N\}$ eine Menge von Bewegungsklassen. Dabei ist $N = |\mathcal{C}|$ die Anzahl der Bewegungsklassen. In der Datenbank D^{57} gilt also $N = 57$. Mit $H(C)$ wird die Menge der gefundenen Dokumente und mit $H^+(C_n)$ die Menge der relevanten Dokumente zu der Anfrageklasse C bezeichnet. Das Diagramm ist eine $N \times N$-Matrix M, wobei jeder Eintrag $M(n, m)$ (Zeile n, Spalte m) mit $n, m \in [1 : N]$ durch

$$M(n, m) := \frac{|H(C_n) \cap H^+(C_m)|}{|H(C_n)|}$$

definiert ist. Alle Einträge der Matrix liegen zwischen 0 und 1. In dem Diagramm werden sie farblich anhand der Farbtabelle rechts vom Diagramm codiert.

Anhand dieser Abbildung kann man die Präzision der Keyframes für jede Klasse einschätzen. Für eine ideale Präzision erwartet man den Wert 1 auf der Diagonalen und den Wert 0 in dem Rest der Matrix. Eine 1 auf der Diagonalen impliziert, dass alle gefundenen Treffer in der richtigen Klasse liegen. Ist der Diagonaleintrag kleiner als 1, so werden auch Treffer zurückgeliefert, die in anderen Klassen liegen. In der ersten Zeile sind die Ergebnisse zu der Klasse cartwheelLHandStart1Reps dargestellt. Hier ist klar zu erkennen, dass der Diagonaleintrag 1 ist, der Rest der Zeile ist 0. Somit liegen alle gefundenen Treffer in der korrekten Klasse. In Tabelle 5.2 ist ebenfalls zu erkennen, dass es in dieser Klasse keine False Positives gibt. In Zeile zwei, welche die Ergebnisse zur Anfrageklasse clap1Reps enthält, ist der Diagonaleintrag sehr dunkel. Hier sind sehr dunkle Einträge über die ganze Zeile verteilt. Das bedeutet, dass die zurückgelieferten Treffer gleichmäßig über sehr viele Klassen verteilt waren. Tatsächlich werden, wie in Tabelle 5.2 nachzulesen ist, für diese Klasse insgesamt 174 Treffer zurückgegeben, von denen nur 6 in der richtigen Klasse liegen. Es ist klar, dass die restlichen Treffer dann über die anderen Klassen verstreut sein müssen, da die Evaluationsdatenbank im Durchschnitt nur ca. 12 Bewegungen pro Klasse enthält.

In dieser Darstellung lässt sich jedoch der Recall nicht abschätzen, d.h. man kann nicht erkennen, wie viele der relevanten Dokumente für jede Klasse zurückgeliefert werden. Mit Abbildung 5.4 kann der Recall für jede Klasse abgeschätzt werden. Es seien die Variablen wie oben definiert. Dann ist jeder Eintrag $M'(n, m)$ der Recall-Matrix M' definiert durch

$$M'(n, m) := \frac{|H(C_n) \cap H^+(C_m)|}{|H^+(C_m)|}.$$

Auch hier liegen die Einträge zwischen 0 und 1 und sie werden farblich codiert.

In der ersten Zeile kann man erkennen, dass zu der Anfrage cartwheelLHandStart1Reps nur der Diagonaleintrag eine 1 enthält, alle anderen Einträge dieser Zeile sind 0. Dies bedeutet, dass alle Radschlag-Bewegungen gefunden werden und keine weiteren Treffer zurückgegeben werden. In Zeile 2 kann man für die Bewegungsklasse clap1Reps auf der Diagonalen erkennen, dass nur 75% der relevanten clap-Bewegungen gefunden werden. Tatsächlich werden, wie an Tabelle 5.2 zu erkennen ist, zwei der 8 relevanten Dokumente nicht gefunden. Daran, dass in der ganzen Zeile Einträge ungleich 0 vorhanden sind, ist leicht zu erkennen, dass die Keyframes

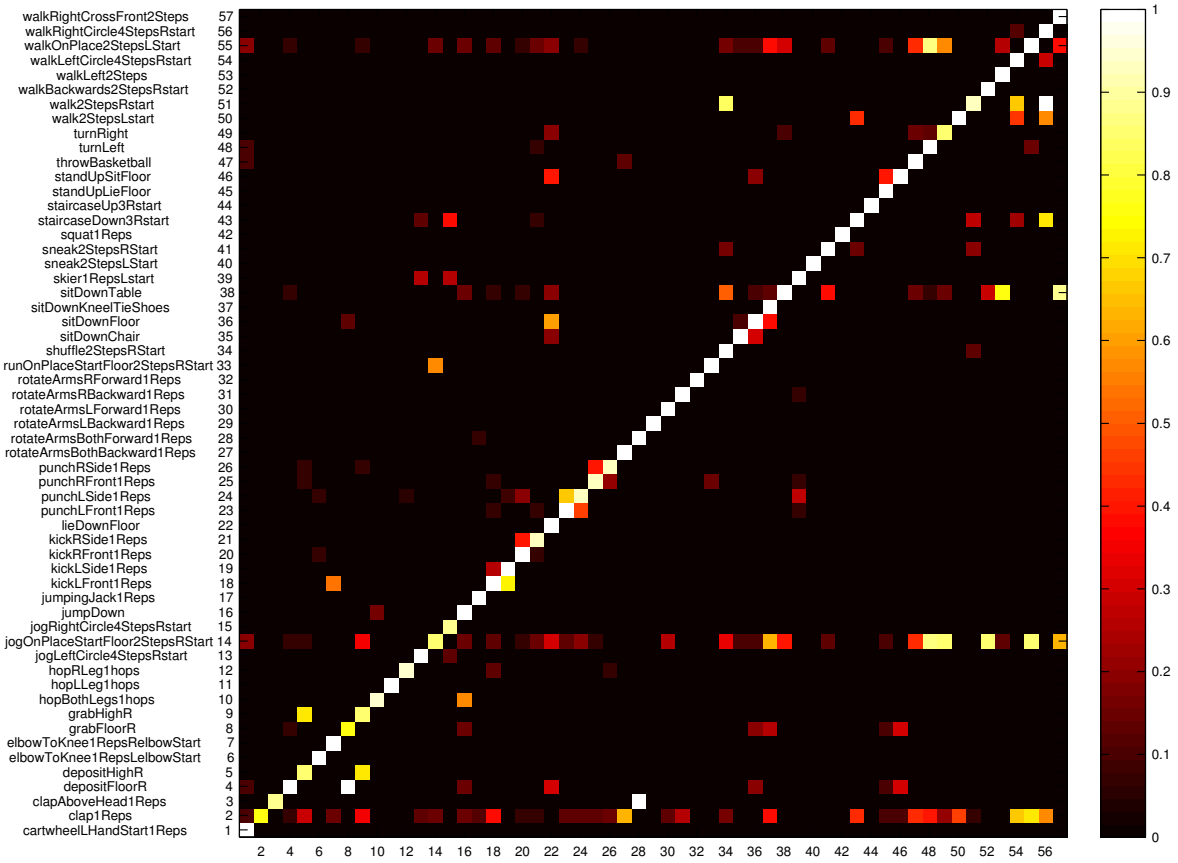
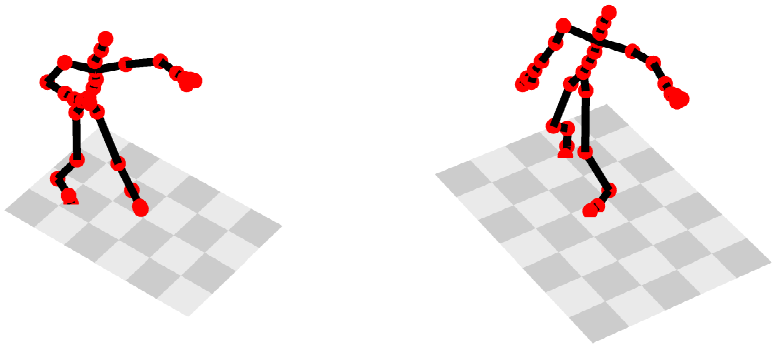


Abbildung 5.4. Diese Abbildung zeigt die Recall-Matrix, die für jede Anfrageklasse zeigt, welchen Anteil der relevanten Dokumente jeder Klasse gefunden werden.

für die clap1Reps-Bewegung sehr unspezifisch sind und die Klasse nicht genau beschrieben ist, wie es auch schon in Abschnitt 5.2 diskutiert wurde.

Anhand dieser beiden Darstellung erkennt man schnell, welche Klassen mit den Keyframes gut beschrieben sind, und welche nicht. Eine geringe Präzision ist bei den Klassen clap1Reps, jogOnPlaceStartFloor2StepsRStart, sitDownTable und walkOnPlace2StepsLStart zu erkennen. Man kann ebenfalls gut erkennen, dass die Bewegungen kickLFront1Reps und kickLSide1Reps durch die Keyframes nicht getrennt werden können: ca. 50% der Treffer bei kickLFront1Reps liegen in der korrekten Klasse, weitere 50% der Treffer liegen in der Klasse kickLSide1Reps. Das gleiche Problem tritt bei den Bewegungsklassen kickRFront1Reps, kickRSide1Reps, bei den Klassen punchLFront1Reps, punchLSide1Reps und punchRFront1Reps, punchRSide1Reps auf. Die Ursache hierfür liegt in den verwendeten Features. Als Repräsentant für Artefakte dieser Art sollen hier kurz die Bewegungen punchLFront1Reps und punchLSide1Reps untersucht werden. Abbildung 5.6 zeigt die „upper“-Features für jede der 30 Bewegungen in der punchLFront1Reps- und punchLSide1Reps-Klasse. Die beiden Bewegungen unterscheiden sich im Wesentlichen dadurch, dass die Box-Bewegung in der einen nach vorne, in der anderen Klasse zur Seite ausgeführt wird (siehe auch Abbildung 5.5). Das zweite Feature der „upper“-Featuremenge, welches beschreibt, ob sich die linke Hand nach vorne bewegt und welches die Bewegungen prinzipiell unterscheiden könnte, nimmt in der Klasse



(a) (b)

Abbildung 5.5. Abbildung (a) zeigt ein Standbild aus einer punchLFront Bewegung. In (b) ist die Bewegung punchLSide veranschaulicht.

punchLFront wie erwartet konsistent den Wert 1 an. In der Klasse punchLSide reagiert dieses Feature jedoch auch. Um zu verstehen, warum dies geschieht, kann man sich das Design des Features ansehen (siehe auch [MR06]): Es wird eine Ebene durch die drei Punkte „neck“, „rhip“ und „lhip“ des Standardskelettes (Abbildung 2.1) gelegt. Nun wird getestet, ob sich das Gelenk „lwrist“ in Richtung der Normalen zu dieser Ebene bewegt. Liegt die Geschwindigkeit der Bewegung über einem gewissen Schwellwert so nimmt das Feature den Wert 1 an. Nun ist auch bei einem Schlag, der seitwärts ausgeführt wird, eine kleine Geschwindigkeitskomponente in Richtung der Normalen der beschriebenen Ebene vorhanden. Wird die Bewegung nun schnell genug ausgeführt, wie es bei einem Schlag typischerweise der Fall ist, so übersteigt die Geschwindigkeitskomponente in Richtung der Normalen den gewählten Schwellwert. Somit ist dieses Feature nicht mehr geeignet, um die beiden Bewegungen zu unterscheiden und es kommt zu hoher Verwechslung zwischen den beiden Klassen.

5.4 Experimente auf längeren Bewegungssequenzen

Die vorhergehenden Untersuchungen zur Qualität der Keyframes geschahen auf der Datenbank $\mathcal{D}^{57}$. Diese Datenbank enthält geschnittene Einzelbewegungen von jeder Bewegungsklasse. So können automatisiert False Positives und False Negatives bestimmt werden. Damit ist diese Datenbank zum systematischen Überprüfen der Aussagekräftigkeit der Keyframes gut geeignet, jedoch nicht, um ein realistisches Retrieval-Szenario zu erhalten. Dadurch, dass die Daten geschnitten sind, verhalten sich die Retrieval-Laufzeiten besser als auf ungeschnittenen, längeren Bewegungssequenzen.

Realistische Retrieval-Szenarien werden in diesem Kapitel untersucht. Dabei wird die Datenbank HDM05_amc (siehe Anhang A) mit Hilfe von Keyframes durchsucht. Jedes Dokument der Datenbank ist eine vollständige Ausführung einer Szene aus dem Drehbuch der Datenbank. Das Drehbuch ist im Anhang A einzusehen. Die Datenbank enthält ca 3.5 Stunden Motion Capture-Material. Alle Retrieval-Tests werden bei einer Framerate von 30 Hz durchgeführt. Höhere Frameraten sind für die Aufgabe, Bewegungsklassen zu erkennen, nicht erforderlich. Die Datenbank hat bei 30 Hz eine Länge von 380838 Frames und enthält 324 Dokumente.

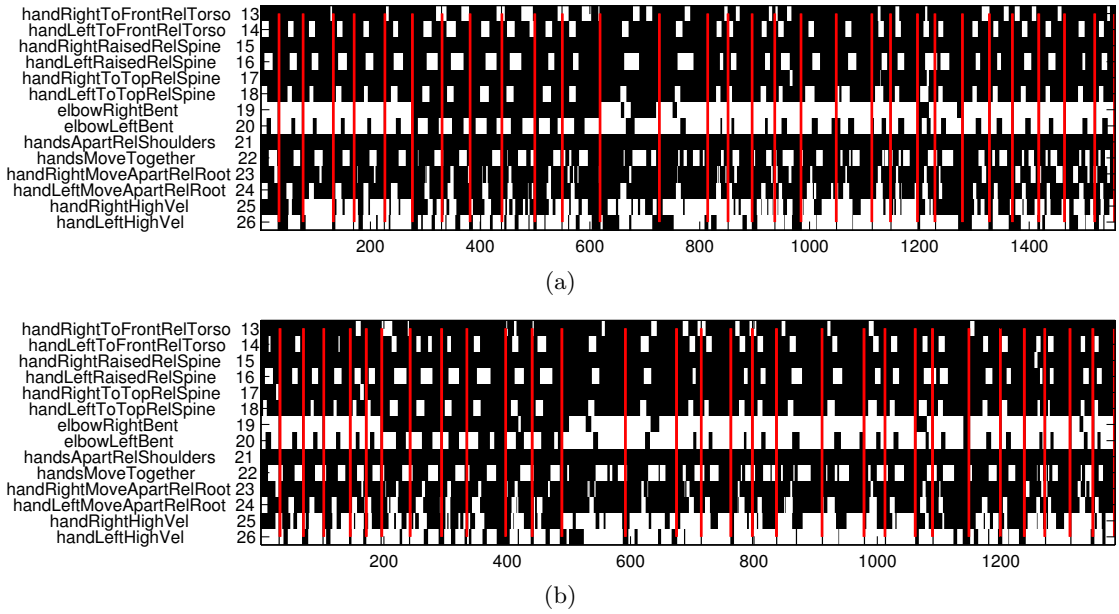


Abbildung 5.6. Diese Abbildung zeigt die „upper“-Featuremenge für alle Bewegungen der Klassen punchLFront1Reps in (a) und punchLSide1Reps in (b). Es ist zu erkennen, dass die Features der Bewegungen sehr ähnlich sind und im Speziellen das Feature handLeftToFrontRelTorso, welches die Bewegungen unterscheiden könnte, bei beiden Klassen den Wert 1 annimmt. Somit sind diese Klassen mit den verwendeten Features schwer zu unterscheiden.

Mit der verwendeten Entwicklungs-Software Matlab® 7.5 schlagen Anfragen an Datenbanken dieser Länge bei Verwendung eines DTW-basieren Ansatzes fehl. Die Kostenmatrix C passt hier nicht mehr in den verfügbaren Speicher. Abgesehen davon würde das Suchen von Bewegungsklassen bei dieser Datenbanklänge mehrere Minuten in Anspruch nehmen. In vielen Anwendungsszenarien sind Antwortzeiten in diesem Bereich nicht mehr akzeptabel.

Zuerst werden in Abschnitt 5.4.1 Analysen der Laufzeit des Algorithmus 4.1 auf längeren Bewegungssequenzen durchgeführt und auch das Verhalten der Laufzeit für wachsende Datenbank-Größen untersucht. In Abschnitt 5.4.2 wird dann untersucht, wie eine Retrieval-Aufgabe, in der nach einem komplexen Bewegungsablauf gesucht werden soll, mit Keyframes gelöst werden kann. Schließlich wird in Abschnitt 5.4.3 analysiert, welche Auswirkung die Variation des Stiffness-Parameters auf die Qualität der Retrieval-Ergebnisse hat.

5.4.1 Laufzeituntersuchung

Dieser Abschnitt untersucht die Laufzeiten von Algorithmus 4.1 in der Praxis auf ungeschnittenen, längeren Bewegungssequenzen. Das benutzte Retrieval-Verfahren ist in Abschnitt 5.3 zusammengefasst. Tabelle 5.3 zeigt die Laufzeit für jede der Bewegungsklassen der Datenbank $\mathcal{D}^{57}$ im Überblick. Die Spalte # KF gibt die Anzahl der Keyframes wieder, die bei der Anfrage benutzt werden. Die konkret benutzten Keyframes können in Anhang B für jede Klasse eingesehen werden. $\sum \ell_v$ gibt die Summe der Länge der invertierten Listen der Keyframes an. Die Länge einer invertierten Liste ist die Anzahl der enthaltenen Segmente. Mit der Angabe der

Laufzeit t^K (in Sekunden) der keyframebasierten Suche kann nun verglichen werden, für welche Listenlängen die Suche wie lange benötigt. Die Zeiten für die Ausführung der rekursiven Funktion (Algorithmus 4.2) sind in der Spalte t^K_{Rec} (in Millisekunden) angegeben. Hier kann man gut erkennen, dass die eigentliche Suche, die im Bereich weniger Millisekunden liegt, von den Vorverarbeitungsschritten (Vereinigung der invertierten Listen, Konvertierung der Listen) dominiert wird. Hier lässt sich die Effizienz der Suche noch enorm steigern, in dem die invertierten Listen in einer anderen Struktur gespeichert werden. Dies geht jedoch über den Rahmen dieser Diplomarbeit hinaus. Diskussionen zu dem Verhältnis von Listenlängen und Laufzeit werden später in diesem Abschnitt durchgeführt. Nun gibt $\%(D)$ Auskunft darüber, wie stark die Datenbankgröße durch die Keyframesuche reduziert wird. Es wird das Verhältnis der Frames in der Datenbank HDM05_amc und der Frames in der durch die Keyframesuche reduzieren Datenbank in Prozent angegeben. Schließlich gibt die Spalte t^R an, welche Zeit in Sekunden das Herstellen einer Rangordnung der Treffer auf der reduzieren Datenbank mit dem Teilfolgen-DTW-Algorithmus (Kapitel 3.2) benötigt.

Bei der Analyse der Zeiten des Algorithmus 4.1 fällt auf, dass die meisten Zeiten für die Keyframesuche im Bereich von Zehntelsekunden liegen. Dabei ist es klar, dass die Klassen, deren Keyframes wenig aussagekräftig sind, also wenig 1-Einträge enthalten, eine längere Laufzeit haben, da die invertierten Listen der Keyframes dann typischerweise mehr Elemente enthalten. Für die Bewegungsklassen walk2StepsLstart und walk2StepsRstart liegen die Zeiten für die Keyframesuche bei 0.5 Sekunden und die Datenbank wird jeweils nur auf ca. 10% heruntergeschnitten. Bei der manuellen Durchsicht der reduzierten Datenbank fällt hier auf, dass tatsächlich nur Geh-Bewegungen enthalten sind. Dadurch, dass in den einzelnen Szenen des Drehbuches beispielsweise zwischen zwei konkreten Bewegungen zum Ausgangspunkt zurückgegangen werden musste, enthält die Datenbank sehr viele Ausschnitte mit Gehbewegungen. Auch können Gehbewegungen im Kreis nicht gut von solchen, die gerade ausgeführt werden, getrennt werden. Dies ist auch in Abbildung 5.3 zu erkennen. Insgesamt ergibt sich eine immens große Treffermenge von korrekten Treffern, wodurch das Herstellen einer Rangordnung unter den Treffern auch ca. 10 Sekunden dauert.

Die Klassen clap1Reps, jogOnPlaceStartFloor2StepsRStart, sitDownTable, turnLeft, turnRight und walkOnPlace2StepsLStart sind dabei Klassen, deren Laufzeiten der Keyframesuche bei einer Sekunden und deren Laufzeiten des Ranking-Schrittes bei 10 Sekunden oder mehr liegen. Schon bei der Analyse auf Bewegungsklassen (Abschnitt 5.3) sind diese Klassen aufgrund ihrer schlechten Präzision hervorgetreten und es wurde festgestellt, dass die Keyframes für diese Klassen sehr unspezifisch sind und wenig charakteristische Einsen enthalten. Somit ist klar, dass die invertierten Listen entsprechend lang werden. Des Weiteren fällt bei diesen Klassen auf, dass die reduzierte Datenbank im Vergleich zur Datenbank HDM05_amc immer noch relativ groß ist: Es bleiben $10\% - 36\%$ der Datenbank übrig und dementsprechend liegen die Zeiten t^R für das Herstellen einer Rangordnung im Bereich von $10 - 45$ Sekunden.

Für die anderen Klassen sind die Retrieval-Zeiten wesentlich besser. Hier werden in den meisten Fällen nur 2% oder weniger der Datenbank HDM05_amc für das Ranking übrig gelassen. Die Zeiten für die Keyframesuche liegen bei einer Sekunde und auch die Ranking-Zeiten liegen im Bereich von einer Sekunde. Die Klasse cartwheelLHandStart1Reps ist hier ein Beispiel mit extrem kurzen invertierten Listen: Die Summe der Längen beträgt hier nur 96 Elemente. Der Grund hierfür ist, dass diese Klasse charakteristische Posen enthält, die in keiner anderen Klasse zu finden sind. So beschreibt der erste Keyframe dieser Klasse eine Pose, in welcher

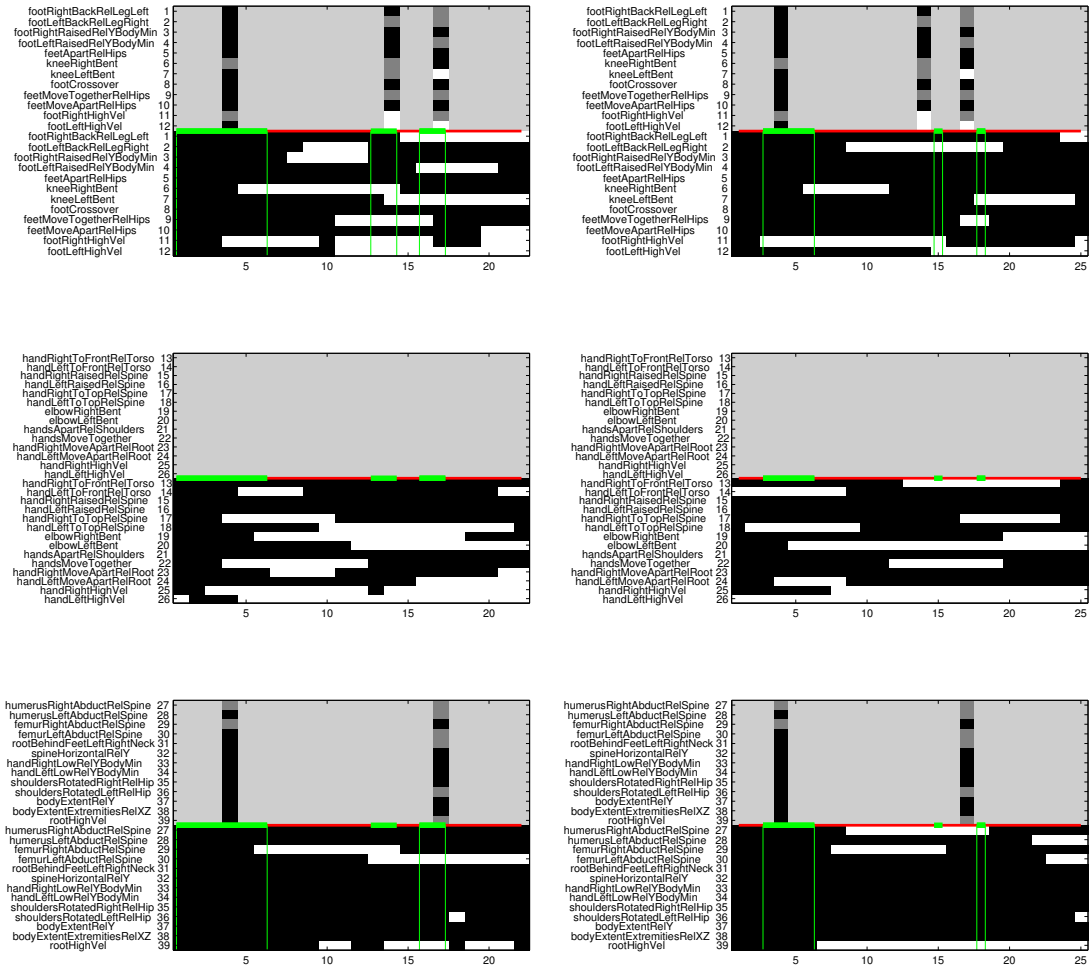
Nr.	Bewegungsklasse	$\#KF$	$\sum \ell_v$	$t^K[s]$	$t^K_{\text{Rec}}[ms]$	$\%(D)$	$t^R[s]$
1	cartwheelLHandStart1Reps	3	96	0.01	0.10	1.25	0.52
2	clap1Reps	5	32121	1.40	5.34	14.03	13.92
3	clapAboveHead1Reps	5	17697	0.46	0.92	1.73	1.05
4	depositFloorR	5	29671	0.71	1.16	5.96	3.41
5	depositHighR	7	27760	0.79	1.29	2.22	1.25
6	elbowToKnee1RepsLelbowStart	3	405	0.02	0.12	0.63	0.15
7	elbowToKnee1RepsRelbowStart	4	2436	0.08	0.27	0.59	0.14
8	grabFloorR	7	19944	0.75	0.72	2.50	1.02
9	grabHighR	6	16661	0.67	1.28	2.93	2.15
10	hopBothLegs1hops	5	26002	0.87	1.04	1.00	0.34
11	hopLLeg1hops	3	5396	0.17	0.38	0.54	0.13
12	hopRLeg1hops	4	19715	0.64	0.99	1.11	0.26
13	jogLeftCircle4StepsRstart	4	3477	0.09	0.36	1.42	0.57
14	jogOnPlaceStartFloor2StepsRStart	5	30014	1.03	6.14	30.35	20.36
15	jogRightCircle4StepsRstart	3	2594	0.07	0.32	1.33	0.48
16	jumpDown	4	9305	0.22	0.42	1.24	0.46
17	jumpingJack1Reps	3	1349	0.04	0.30	0.89	0.25
18	kickLFront1Reps	5	11344	1.03	0.74	1.97	1.02
19	kickLSide1Reps	4	1563	0.06	0.27	1.12	0.41
20	kickRFront1Reps	5	4882	0.17	0.40	1.13	0.43
21	kickRSide1Reps	4	6049	0.41	0.36	1.17	0.39
22	lieDownFloor	3	1325	0.07	0.24	2.68	2.23
23	punchLFront1Reps	5	9594	0.42	0.57	1.43	0.61
24	punchLSide1Reps	6	19894	1.02	1.36	2.61	1.42
25	punchRFront1Reps	6	16380	0.62	0.98	2.50	1.34
26	punchRSide1Reps	6	20382	1.52	1.07	1.67	0.78
27	rotateArmsBothBackward1Reps	3	1309	0.04	0.22	1.50	0.56
28	rotateArmsBothForward1Reps	6	10109	0.26	0.47	0.72	0.20
29	rotateArmsLBackward1Reps	4	11058	0.24	0.45	0.58	0.17
30	rotateArmsLForward1Reps	3	836	0.03	0.21	0.65	0.20
31	rotateArmsRBackward1Reps	3	2488	0.07	0.31	0.61	0.17
32	rotateArmsRForward1Reps	3	475	0.02	0.17	0.74	0.22
33	runOnPlaceStartFloor2StepsRStart	5	21753	0.88	1.63	0.40	0.09
34	shuffle2StepsRStart	8	39025	1.17	2.42	2.03	1.20
35	sitDownChair	6	4076	0.12	0.59	2.37	1.36
36	sitDownFloor	5	5749	0.16	0.56	4.69	3.75
37	sitDownKneelTieShoes	3	590	0.02	0.16	2.18	1.70
38	sitDownTable	6	29826	0.72	4.20	29.27	34.62
39	skier1RepsLstart	5	4600	0.17	0.56	0.79	0.21
40	sneak2StepsLStart	4	9139	0.21	0.93	1.54	0.77
41	sneak2StepsRStart	6	23508	0.69	1.62	1.96	1.07
42	squat1Reps	4	1524	0.04	0.23	0.97	0.31
43	staircaseDown3Rstart	5	15470	0.39	1.86	6.54	4.88
44	staircaseUp3Rstart	5	4647	0.12	0.39	1.67	0.87
45	standUpLieFloor	3	529	0.03	0.12	1.74	1.12
46	standUpSitFloor	6	11279	0.31	0.57	3.56	2.35
47	throwBasketball	5	7014	0.23	0.71	2.95	1.82
48	turnLeft	7	37642	1.26	3.25	10.63	12.50
49	turnRight	9	61584	2.29	9.73	20.09	25.10
50	walk2StepsLstart	6	21801	0.52	2.53	10.54	9.15
51	walk2StepsRstart	5	21607	0.71	2.78	13.23	9.03
52	walkBackwards2StepsRstart	5	16586	0.35	1.11	1.13	0.45
53	walkLeft2Steps	3	625	0.02	0.15	0.95	0.37
54	walkLeftCircle4StepsRstart	6	19645	0.47	2.28	13.53	13.66
55	walkOnPlace2StepsLStart	5	27383	0.59	5.13	35.91	44.09
56	walkRightCircle4StepsRstart	7	19124	0.43	2.60	11.48	10.76
57	walkRightCrossFront2Steps	3	7642	0.26	0.46	4.05	2.96

Tabelle 5.3. Diese Tabelle zeigt die Retrieval-Zeiten für jede Bewegungsklasse der $\mathcal{D}^{57}$ auf der Datenbank HDM05_amc. Diese beinhaltet 380838 Frames, dies entspricht bei 30 Hz einer Länge von ca. 3.5 Stunden Motion-Capture-Daten. $\#KF$: Anzahl der benutzten Keyframes. $\sum \ell_v$: Summe der Längen der invertierten Keyframelisten. t^K: Zeit der keyframebasierten Suche in Sekunden. t^K_{Rec}: Zeit des rekursiven Algorithmus 4.2 in Millisekunden. $\%(D)$: Anteil der reduzierten Datenbank an der Gesamtdatenbank in Prozent. t^R: Zeit für das Ranken der reduzieren Datenbank in Sekunden.

die Wirbelsäule horizontal steht und die linke Hand abgesenkt ist ($F32, F34$). Gleichzeitig sind die Extremitäten ausgestreckt ($F38$). Diese Pose ist in keiner der anderen Bewegungen vorhanden. In der darauf folgenden Pose, beschrieben mit dem zweiten Keyframe, ist die Wirbelsäule nicht mehr horizontal, zusätzlich ist noch die rechte Hand abgesenkt. Auch diese Pose ist dadurch, dass $F37$ den Wert 0 und $F38$ den Wert 1 hat, also der Körper in alle Richtungen ausgestreckt ist, gut von einer grabFloor - Bewegung unterscheidbar und einzigartig unter den Bewegungsklassen. Somit ist klar, dass nur ca. 1% der Datenbank nach der Keyframesuche übrig bleiben. Durch die charakteristischen Keyframes ist es nicht verwunderlich, dass die manuelle Analyse der Treffer bei dieser Klasse ein perfektes Retrieval-Ergebnis ergibt, bei dem weder False Positives noch False Negatives auftreten.

Doch auch bei nicht sehr charakteristischen Klassen kann die Keyframesuche erfolgreich sein: Die Klasse runOnPlaceStartFloor2StepsRStart beispielsweise ist mit 5 Keyframes und insgesamt nur 4 Einsen in den Keyframes nicht sehr stark spezifiziert. Dadurch sind die invertierten Listen in der Summe sehr lang. Doch die Kombination der Keyframes ist so einzigartig innerhalb der Datenbank, dass weniger als 1% der Datenbank nach der Keyframesuche übrig bleibt und so das Ranken der reduzierten Datenbank nur noch 90ms in Anspruch nimmt, so dass die Gesamtlaufzeit für das Retrieval unter einer Sekunde liegt. Unter den ersten 34 Treffern dieser Klasse befinden sich nur 2 False Positives. Es zeigt sich, dass diese beiden False Positives Jogging-Bewegungen sind, bei denen die Füße stark angezogen werden. Abbildung 5.7 zeigt die Keyframes der Anfrage und die Featurematrix eines korrekten Treffers, sowie eines False Positives: Eine Jogging-Bewegung in einer Linkskurve. In den Abbildungen sieht man im oberen Drittel die „lower“-Featuremenge, im mittleren Drittel die „upper“-Featuremenge und unten ist die „mix“-Featuremenge zu sehen. Jedes Drittel ist nun mit einer horizontalen, roten Linie in zwei Hälften unterteilt. In der oberen Hälfte sind die Keyframes der jeweiligen Featuremenge zu sehen. Die untere Hälfte zeigt die Feturematrix der entsprechenden Featuremenge des Treffers. In jedem Drittel der Abbildung sind mit waagerechten, grünen Linien die Bereiche eingezeichnet, die von einem Keyframe getroffen werden. So zeigt die erste grüne waagerechte Linie an den Frames 1-6 an, dass dieser Bereich von einem Keyframe getroffen wird. Will man wissen, in welcher Featuremenge der Keyframe liegt, der diesen Bereich getroffen hat, so muss man die senkrechten grünen Linien betrachten. Sie zeigen beispielsweise (Abbildung 5.7(a)) im oberen Drittel an, dass der Bereich $1-6$ vom ersten Keyframe getroffen wird. Im unteren Drittel wird ebenfalls der Bereich $1-6$ getroffen, jedoch von dem zweiten Keyframe, der in diesem Fall dieselbe Position (4) wie der erste Keyframe hat. Dabei sind mit den senkrechten grünen Linien die größtmöglichen Bereiche markiert, die von Keyframes getroffen werden können. Das bedeutet, dass man in jedem Falle Fehl schlägt, wenn man einen Hit nach Definition 4.4 konstruieren möchte und für einen Keyframe einen Frame auswählen möchte, der außerhalb des jeweiligen mit senkrechten grünen Linien markierten Bereiches liegt. Andererseits bedeutet dies auch, dass wenn man für einen Keyframe (z.B. den ersten Keyframe) einen beliebigen Frame aus dem grün markierten Bereich (Frames $1-6$) auswählen kann, und ausgehend von diesem Frame in jedem Fall einen Hit nach Definition 4.4 finden kann.

Dies ist an dem Beispiel Abbildung 5.7(a) nachzuvollziehen. Die Keyframes für die Klasse 33: runOnPlaceStartFloor2StepsRStart treten nach Anhang B an den Positionen (4, 4, 14, 17, 17) auf. Sie befinden sich in den Featuremengen (l, m, l, l, m). Die Stiffness zum jeweils nächsten Keyframe ist (1, 0.8, 0.8, 1). Wie in Bemerkung 4.1 erwähnt können die Stiffness-Parameter zwischen je zwei aufeinander folgenden Keyframes unterschiedliche gewählt werden. Möchte



(a) (b)

Abbildung 5.7. Diese Abbildung zeigt zwei Treffer bei der Keyframesuche nach der Klasse runOnPlaceStartFloor2StepsRStart zusammen mit den verwendeten Keyframes. Die Bereiche, in welchen die gefundenen Keyframes den Abstandsbedingungen eines Hits genügen, sind mit senkrechten grünen Linien markiert. Die drei Featurematrizen zu den Featuremengen „lower“, „upper“, „mix“ (von oben nach unten) eines korrekten Treffers in (a) und eines False Positives in (b) sind gezeigt.

man nun prüfen, ob mit den in Abbildung 5.7(a) gegebenen Bereichen für die Keyframes ein Hit möglich ist, so kann man wie folgt vorgehen: Man wählt für den ersten Keyframe einen beliebigen Frame in dem mit senkrechten grünen Linien markierten Bereich. Wählt man nun beispielsweise für den ersten Keyframe den Frame 1, so bleibt für den zweiten Keyframe keine Wahl, da der Abstand der ersten beiden Keyframes 0 betragen muss. Somit wird für den zweiten Keyframe ebenfalls die Position 1 gewählt. Der Abstand zum nächsten Keyframe beträgt $14-4=10$. Mit der Stiffness von 0.8 ergibt sich ein Abstand zum nächsten Keyframe, der in dem Bereich $[\lceil 10 \cdot 0.8 \rceil : \lfloor 10/0.8 \rfloor] = [8 : 12]$ liegen muss. Wählt man für den dritten Keyframe den Frame 13, so beträgt der Abstand zum zweiten Keyframe genau 12 und liegt somit in dem erlaubten Bereich. Der Abstand zum vierten Keyframe beträgt $17 - 13 = 3$. Der Abstand zum nächsten Keyframe muss also im Bereich $[\lceil 3 \cdot 0.8 \rceil : \lfloor 3/0.8 \rfloor] = [3 : 3]$

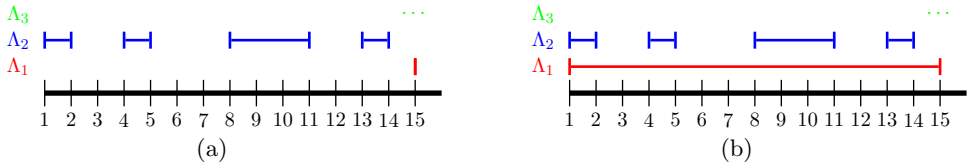


Abbildung 5.8. In (a) ist ein Szenario dargestellt, in dem die lange invertierte Liste des zweiten Keyframes kaum etwas zu der praktischen Laufzeit beiträgt, da die Elemente von Λ_2 übersprungen werden. In (b) kann mit jedem Element von Λ_2 eine Rekursion gestartet werden. Daher kann die beobachtete Laufzeit wesentlich größer sein als in (a), obwohl in beiden Szenarien die Anzahl der Segmente in den invertierten Listen gleich ist.

liegen. Nun muss man also den letzten Keyframe an Position 16 wählen, und er liegt auch tatsächlich innerhalb des grün markierten Bereiches. Ebenso prüft man leicht nach, dass alle grün markierten Bereiche zu den entsprechenden Keyframes passen: Alle Bereiche, die schwarz und weiß in einem Keyframe markiert sind, also einen Wert 0 oder 1 haben, verfügen über denselben Wert in den grün markierten Bereichen. Die Werte, die in einem Keyframe mit 0.5 (grau) markiert sind, können in der Featurematrix sowohl den Wert 0 als auch den Wert 1 annehmen.

Im Vergleich des korrekten Treffers (a) und der fälschlicherweise als Treffer erkannten Jogging-Bewegung (b) fällt auf, dass der korrekte Treffer eine längere Sequenz enthält, in der beide Füße eine hohe Geschwindigkeit haben ($F11, F12$). Diese Phase fällt in der Jogging-Bewegung (b) nur sehr kurz aus. Außerdem existiert in (a) eine Phase, in der beide Knie gebeugt sind ($F6$, $F7$) und beide Oberschenkel angezogen sind ($F29$, $F30$). Diese Posen sind in (b) nicht vorhanden. Da die eben beschriebenen Posen jedoch nicht mit den Keyframes beschrieben werden, wird dieser Treffer trotz struktureller Unterschiede in den Featurematrizen gefunden.

Die Laufzeit der keyframebasierten Suche für diese und fast alle anderen Klassen liegen im Bereich von einer Sekunde. Hier trägt die Ausführung der rekursiven Funktion (Algorithmus 4.2) nur einen geringen Anteil von wenigen Millisekunden (siehe Spalte t^K_{Rec} in Tabelle 5.3). In der theoretischen Betrachtung des Algorithmus 4.1 wurde eine Laufzeituntersuchung in Abschnitt 4.2.3 durchgeführt, in der bewiesen wurde, dass die Worst-Case-Laufzeit von Algorithmus 4.1 linear in der Summe der Längen der invertierten Listen der Keyframes ist. Diese Aussage soll im Folgenden überprüft werden. Dafür wird nur die Laufzeit der rekursiven Funktion gemessen und alle Vorverarbeitungsschritte ignoriert, um ein möglichst klares Bild von der Laufzeit der Rekursion zu erhalten. In der Praxis hängt die Laufzeit des rekursiven Algorithmus nicht nur von der Länge, sondern auch von der Struktur der Listen untereinander ab. Wie in Abbildung 5.8 dargestellt wird, kann die Länge einer invertierten Liste irrelevant für die Laufzeit sein. In 5.8(a) beginnt das erste Element der ersten invertierten Liste Λ_1 nach dem Ende der zweiten Liste Λ_2. Dadurch werden bei einem Start des Algorithmus 4.1 mit diesen invertierten Listen die Elemente der zweiten invertierten Liste übersprungen, ohne dass mit ihnen eine Rekursion gestartet wird. In der Praxis benötigt das Überspringen von Listenelementen kaum Zeit. Wird mit einem Listenelement eine Rekursion gestartet, so kann dies sehr viel mehr Zeit kosten, als wenn das Element übersprungen wird. Bei einer Listenstruktur wie in 5.8(b) kann – abhängig von dem Abstand der Keyframes und dem Stiffness-Parameter – für jedes Element der Liste Λ_2 eine Rekursion gestartet werden. Obwohl in beiden Beispielen die Summe

der Listenlängen gleich ist, wird die beobachtete Laufzeit in (b) sehr viel größer sein als in (a). Dieser Sachverhalt zeigt sich in der Praxis auch in Tabelle 5.3. Hier sind die praktischen Laufzeiten trotz ähnlichen Listenlängen oft sehr unterschiedlich. Betrachte beispielsweise die Klassen 4:depositFloorR und 14:jogOnPlaceStartFloor2StepsRStart. Die Summe der Listenlängen beträgt bei beiden Klassen ungefähr 30000. Die Laufzeiten für die rekursive Funktion sind jedoch mit 1.16 und 6.14 Millisekunden sehr unterschiedlich.

Um zu überprüfen, ob sich die praktische Laufzeit für wachsende Listenlängen linear verhält, ist es also nicht sinnvoll, beliebige Strukturen von invertierten Listen zuzulassen. Stattdessen ist es angebracht, diese Vermutung jeweils für ähnliche Listenstrukturen zu untersuchen. Dies kann erreicht werden, indem bei der Untersuchung die Anfrage und damit die Keyframes konstant gelassen werden und die Größe der Datenbank variiert wird. Dadurch, dass die Keyframes nicht variieren, ist auch zu erwarten, dass die Struktur der invertierten Listen der Keyframes untereinander gleich bleibt. In der Untersuchung wird die Datenbank HDM05_amc in 10 Abschnitte mit ungefähr gleicher Größe unterteilt. Aus diesen 10 Abschnitten $D_1, \ldots, D_{10}$ werden nun insgesamt 10 verschiedene Datenbanken mit ansteigender Größe gebildet: $\mathcal{D}_1 = (D_1)$, $\mathcal{D}_2 = (D_1, D_2)$, $\ldots$, $\mathcal{D}_{10} = (D_1, D_2, \ldots, D_{10})$. Nun werden jede dieser 10 Datenbanken mit der Anfrage durchsucht und die Summe der Längen der invertierten Listen, sowie die auftretende Laufzeit protokolliert. Dabei ist die protokollierte Zeit nur die Zeit, die zu der Ausführung der rekursiven Funktion (Algorithmus 4.2) benötigt wird. Die Zeit zum Vereinigen der invertierten Listen (Algorithmus 4.1 Zeile 2) wird dabei nicht berücksichtigt, um ein möglichst klares Bild von den Laufzeiten der rekursiven Funktion zu erhalten. Abbildung 5.9 zeigt dabei die Laufzeiten (auf der senkrechten Achse, aufgetragen in Millisekunden) in Abhängigkeit von der Länge der invertierten Listen. Dabei sind die Laufzeiten für jede der 57 Bewegungsklassen der $\mathcal{D}^{57}$ eingetragen. Die einzelnen Messwerte einer Messreihe werden hier mit geraden Linien verbunden, um einen besseren visuellen Eindruck zu erhalten, welche Messpunkte zusammen gehören. In dem Überblick fällt dabei auf, dass sich die Laufzeiten für die verschiedenen Anfragen alle linear in den Listenlängen verhalten. Die starken Schwankungen mancher Messkurven lassen sich dadurch erklären, dass in dem Messbereich von wenigen Millisekunden die Messwerte mit Matlab® oft nicht exakt bestimmt werden können. Im Einzelnen treten für bestimmte Listenlängen sehr unterschiedliche Laufzeiten auf. So streckt sich der Bereich der Laufzeiten bei einer Listenlänge von 20000 Segmenten von ca. $0.5 - 3.5$ Millisekunden. Hier zeigt sich, dass wie in Abbildung 5.8 gezeigt die Laufzeit bei gleichen Listenlängen sehr unterschiedlich sein kann. Das bedeutet, dass in der Praxis die Summe der Listenlängen nur ein grobes Maß ist, um die Laufzeit vorherzusagen. Mit der Summer der Listenlängen lässt sich die Worst-Case-Laufzeit angeben, in der Praxis sind die auftretenden Laufzeiten jedoch meistens viel geringer.

Am klarsten ist die Linie der Klasse turnRight mit einer Länge von über 61000 Segmenten zu sehen. Da in der Abbildung die Klassen mit relativ geringen Listenlängen schlecht zu erkennen sind, wird in Abbildung 5.10 das Laufzeitverhalten für die beiden Klassen cartwheelLHandStart1Reps und skier1RepsLstart exemplarisch aufgezeichnet. Auch hier verhalten sich die Laufzeiten linear in der Anzahl der Listenelemente.

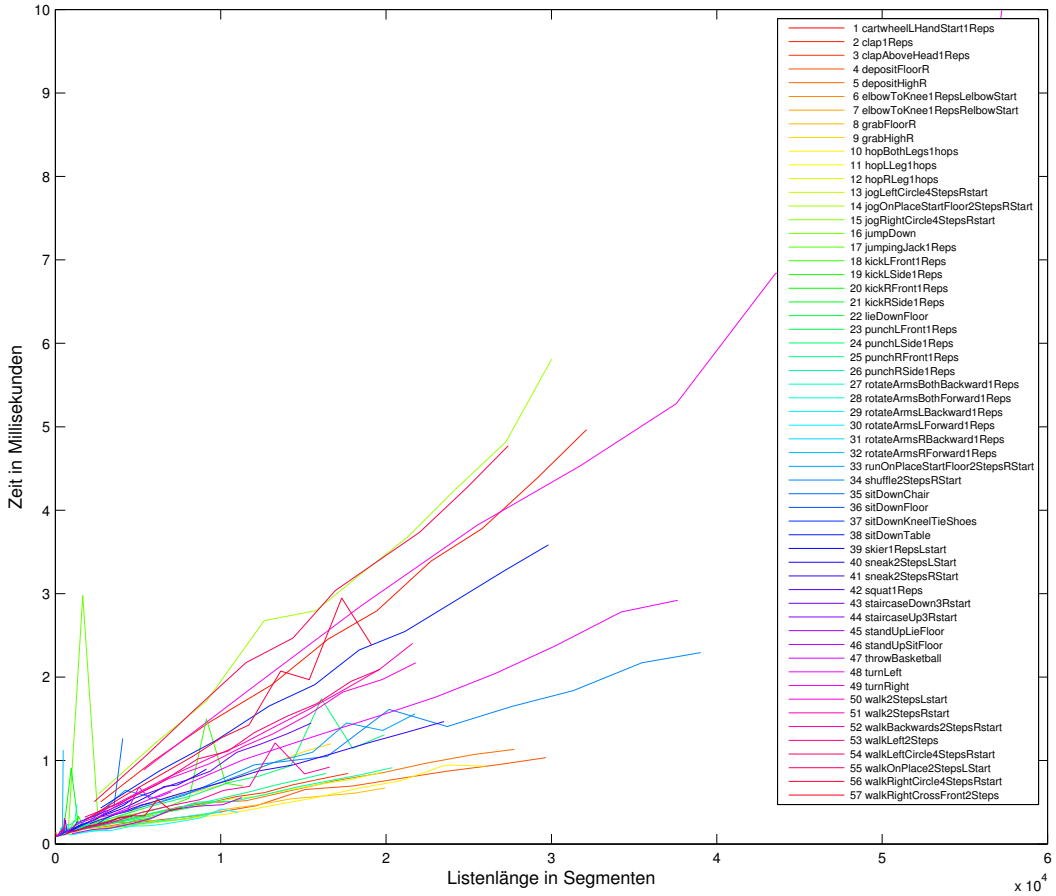
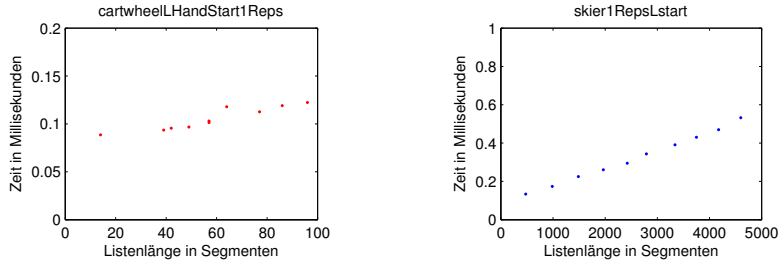


Abbildung 5.9. Laufzeiten der Anfragen bei variabler Datenbankgröße. Es wurden die Laufzeiten der die rekursive Funktion (Algorithmus 4.2) gemessen. Die Anfragen wurden mit 10 verschiedenen Datenbankgrößen durchgeführt. Die längste Datenbank (HDM05_amc, Anhang A) enthält 380838 Frames bei 30 Hz.



(a) (b)

Abbildung 5.10. Die Laufzeiten der rekursiven Funktion in Abhängigkeit von der Länge der invertierten Listen wachsen sowohl bei sehr kurzen, als auch bei langen invertierten Listen linear.

5.4.2 Suche nach komplexen Bewegungsabläufen

Neben der Suche nach einzelnen Bewegungsklassen in einer Datenbank ist mit Hilfe von Keyframes auch eine effiziente Suche nach komplexen Bewegungsabläufen möglich. Um bei-

spielsweise nach der Sequenz zu suchen, in der zuerst eine Ski-Bewegung und danach eine Ellenbogen-zum-Knie-Bewegung stattfindet, können die entsprechenden Keyframes der Bewegungsklasse in einer Anfrage kombiniert werden. Dies sei an dem Beispiel der Ski- und Ellenbogen-zum-Knie-Sequenz erklärt. Die Keyframes der Ski-Bewegung (Klasse 39: skier1RepsLstart) kommen wie in Anhang B beschrieben an den Positionen $(5, 10, 13, 15, 24)$ vor. Wenn man nun diese Keyframes an die Ellenbogen-zum-Knie-Bewegung anhängen möchte, so muss man die Positionen der Keyframes der Ellenbogen-zum-Knie-Bewegung translatieren. Dazu wird auf die Positionen der Keyframes der Ellenbogen-zum-Knie-Bewegung $(12, 17, 24)$ die Länge der Ski-Bewegung und eine gewünschte Distanz zwischen den beiden Bewegungen aufaddiert. Die Ski-Bewegung hat eine Länge von 36 Frames. Der Abstand zwischen den beiden Bewegungen soll 3 Sekunden, also 90 Frames betragen. Somit ergeben sich die Positionen der Keyframes der kombinierten Anfrage zu $(5, 10, 13, 15, 24, 12+36+90, 17+36+90, 24+36+90) = (5, 10, 13, 15, 24, 138, 143, 150)$. Die Abstände zwischen den Keyframes sind damit $(5, 3, 2, 9, 114, 5, 7)$. Nun kann man dem Abstand 114 einen Stiffness-Faktor zuweisen, der die zeitliche Variation der Abstände zwischen den Bewegungen kontrolliert. Nach der Keyframesuche wird auf der reduzierten Datenbank ein Ranking der Treffer durchgeführt. Dazu wird ein MT gebildet, welches aus den MTs der beiden Klassen Ski und Ellenbogen-zum-Knie zusammen gesetzt wird. Zwischen den beiden MTs wird eine Matrix $0.5^{f \times 90}$ eingefügt, wenn der Abstand der beiden Bewegungen 90 Frames betragen soll. Das bewirkt beim Herstellen der Rangordnung unter den Treffern, dass die Bewegungsdaten zwischen den geforderten Bewegungen keine Kosten verursachen.

Auf diese Weise werden 9 Anfragen von komplexen Bewegungsabläufen erstellt, die $6-16$ Keyframes enthalten. Diese Anfragen werden so gewählt, dass die gewählten Sequenzen eindeutig in einer Szene des Drehbuches vorkommen. Da jede Szene des Drehbuches in einem eigenen Dokument der Datenbank vorhanden ist, kann so automatisiert überprüft werden, welche gefundenen Dokumente korrekte Treffer sind und welche nicht. So werden die in Abbildung 5.11 gezeigten Precision-Recall-Diagramme erstellt. Über dem jeweiligen Diagramm sind Informationen über die Anfrage enthalten. In der ersten Zeile sind die angefragten Bewegungen gedruckt, wobei die verschiedenen Bewegungen einer Sequenz mit einem Unterstrich („_“) getrennt sind. Die jeweils zweite Zeile zeigt die Anzahl der gefundenen Treffer („# Hits“), die Anzahl der korrekten gefundenen Treffer („TP“) und die Anzahl der nicht gefundenen relevanten Dokumente („FN“). In der jeweils dritten Zeile sind die Zeiten für die Keyframesuche („t^K“) und die Zeit für das Ranken der reduzierten Datenbank („t^R“) in Sekunden abzulesen.

Precision-Recall-Diagramme eignen sich dazu, auf einen Blick die Qualität von Suchergebnissen einzuschätzen. Dazu ist es hilfreich, eine nach Relevanz sortierte Liste von Treffern als Grundlage zu haben. Eine ausführliche Einführung zu Precision-Recall-Diagrammen findet man in [CM07]. Liegt zu einer Anfrage q nach der keyframebasierten Suche und dem MT-basierten Ranking der reduzierten Datenbank eine Trefferliste T_q vor und ist die Menge der relevanten Treffer R_q zur Anfrage bekannt, so können *Precision* p_q und *Recall* r_q berechnet werden:

$$p_q = \frac{|T_q \cap R_q|}{|T_q|} \tag{5.1}$$

$$r_q = \frac{|T_q \cap R_q|}{|R_q|} \tag{5.2}$$

Dabei gibt die Precision das Verhältnis der Anzahl der gefundenen relevanten Treffer zur Gesamtzahl der erhaltenen Treffer an, während der Recall das Verhältnis der Anzahl der gefundenen relevanten Treffer zur Zahl der gewünschten Treffer beschreibt. Die Precision stellt also die Güte der Trefferliste in Bezug auf die Anzahl fehlerhafter Einträge dar und der Recall die Güte in Bezug auf Vollständigkeit der Trefferliste. Betrachtet man nur die ersten $i \in [1 : |T_q|]$ Einträge der Trefferliste, so definiert man die Precision- und Recallwerte entsprechend für die eingeschränkte Treffermenge $T_{q,i}$:

$$p_{q,i} = \frac{|T_{q,i} \cap R_q|}{|T_{q,i}|} = \frac{|T_{q,i} \cap R_q|}{i} \tag{5.3}$$

$$r_{q,i} = \frac{|T_{q,i} \cap R_q|}{|R_q|} \tag{5.4}$$

Bei schrittweiser Vergrößerung der betrachteten Trefferliste ergeben sich Folgen von Precision- und Recallwerten. Trägt man diese Folgen gegeneinander auf, erhält man das Precision-Recall-Diagramm:

$$\{(r_{q,i}, p_{q,i}) \mid i \in [1 : |T_q|]\} \tag{5.5}$$

Die Diagramme erlauben es, den Verlauf der Precision bei steigendem Recall zu untersuchen. Steigenden Recall erhält man, wenn die betrachtete Liste verlängert wird und damit mehr Treffer berücksichtigt werden. Im Allgemeinen enthält eine längere Trefferliste jedoch auch mehr falsche Einträge, sodass die Precision in den meisten Fällen bei steigendem Recall sinkt.

Im Überblick fällt auf, dass die Precision in jedem Diagramm sehr hoch ist. Bis auf eine Ausnahme in der Anfrage kickLSide1Reps_punchRFront1Reps_punchRSide1Reps sind alle relevanten unter den gefundenen Treffern vorne in der Trefferliste zu finden. Die False Positives befinden sich am Schluss der Trefferliste. Zusätzlich ist der Recall bei den meisten Anfragen sehr hoch. Die Zeiten für die Keyframesuche bewegen sich im Rahmen von 0.06 bis 2.22 Sekunden. Es ist nicht überraschend, dass die kürzeste Zeit für die Keyframesuche bei der Sequenz elbowToKnee1RepsLelbowStart_squat1Reps auftritt: Beide Bewegungen werden mit Keyframes beschrieben, deren invertierte Listen kurz sind, da die Keyframes zu wenig anderen Bewegungen passen. Klar ist auch, dass die Zeit für die Suche nach der Bewegungssequenz kickLSide1Reps_punchRFront1Reps_punchRSide1Reps lange dauert: Hier wird mit insgesamt 16 Keyframes angefragt, die 210 Einträge enthalten. 96 dieser Einträge (also fast die Hälfte) sind auf 0.5 gesetzt, wodurch diese bei diesen Keyframes viele invertierte Listen vereinigt werden müssen. Auch die Laufzeit der Anfrage walkOnPlace2StepsLStart_runOnPlaceStartFloor2StepsRStart ist mit 1.47 Sekunden relativ hoch. In den insgesamt 10 Keyframes sind nur 3 Einsen vorhanden. Damit sind die invertierten Listen lang und es besteht eine Tendenz zu langen Laufzeiten. Insgesamt befindet sich die Retrieval-Zeit als Summe aus den Zeiten für die keyframebasierte Suche und das Ranken der reduzierten Datenbank mit Hilfe von MTs im Bereich von wenigen Sekunden.

Bei der Auswertung der Anfrageergebnisse treten einige Überraschungen auf. Die Anfrage *elbowToKnee1RepsLelbowStart_squat1Reps* führt zu einem False Negative. Dabei passt das Dokument, welches nicht gefunden wird (`HDM_mm_03-05_03_120.amc`), nicht zum Drehbuch: Die Ellenbogen-zum-Knie-Bewegungen und Kniebeugen sind in diesem Dokument fälschlicherweise nicht enthalten.

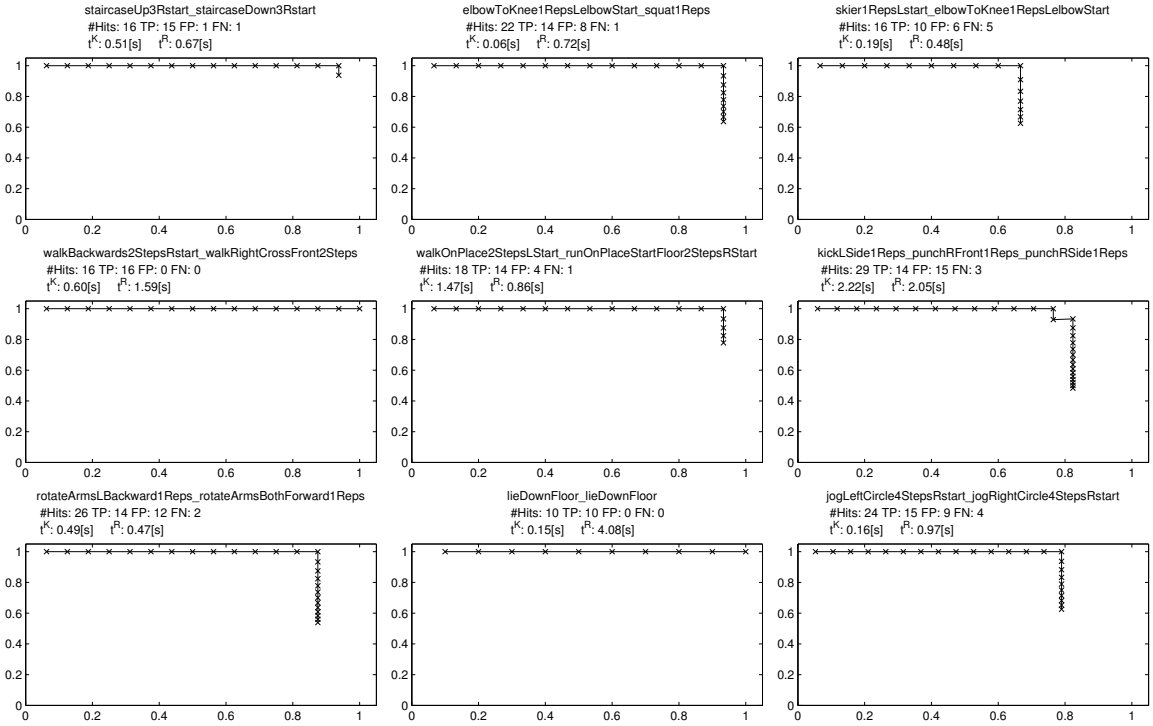
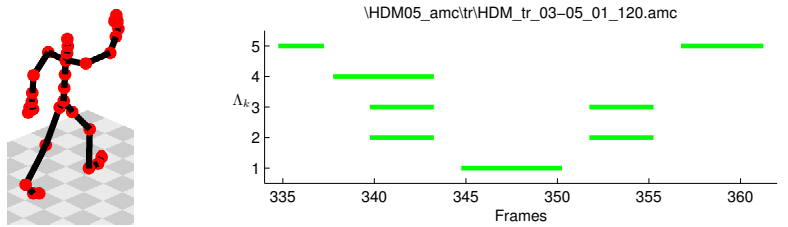


Abbildung 5.11. Precision-Recall-Diagramme verschiedener Anfragen nach komplexen Bewegungsabläufen, gestellt mit 6 bis 16 Keyframes an die Datenbank HDM05_amc (3.5 Stunden Motion-Capture-Material). #Hits: Anzahl der gefundenen Treffer. TP: Anzahl der korrekt gefundenen Treffer (True Positives). FP: Anzahl der False Positives. FN: Anzahl der False Negatives. t^K: Zeit für die keyframe-basierte Suche in Sekunden. t^R: Zeit für das Ranken der reduzierten Datenbank in Sekunden.

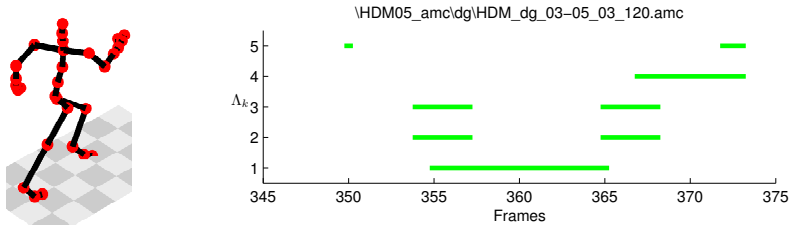
Die Anfrage *skier1RepsLstart_elbowToKnee1RepsLelbowStart* führt zu 5 False Negatives. Dies sind die folgenden Dokumente:

1. `HDM_bk_03-05_01_120.amc`
2. `HDM_mm_03-05_03_120.amc`
3. `HDM_tr_03-05_01_120.amc`
4. `HDM_tr_03-05_02_120.amc`
5. `HDM_tr_03-05_03_120.amc`

Dabei wird das 1. Dokument nicht gefunden, weil die beiden Bewegungen dort ca. 500 Frames auseinander liegen. Die Anfrage erlaubt mit einer Distanz von 90 Frames und zugehöriger Stiffness von 0.5 nur einen maximalen Abstand von 180 Frames. Das zweite Dokument ist schon bei der vorhergehenden Anfrage aufgefallen: Dort fehlen die Ellenbogen-zum-Knie-Bewegungen. Außerdem sind die Ski-Bewegungen in einer Variation ausgeführt, in welcher die Beine die Ski-Bewegung realisieren, die Arme jedoch eine Hampelmann-Bewegung gestalten. Dass die Dokumente 3 bis 5 nicht gefunden werden, bedarf einer genauen Untersuchung. Es stellt sich heraus, dass der ausführende Schauspieler die Bewegung in einer ungewöhnlichen Variante ausgeführt hat. Die Arme und Beine werden bei der Ski-Bewegung parallel und



(a)



(b)

Abbildung 5.12. Die ungewöhnliche Ausführung der Ski-Bewegung führt dazu, dass diese Ausführung mit den Keyframes nicht als Ski-Bewegung erkannt wird. Die ungewöhnliche Pose mit parallelen anstatt antiparallelen Armen und Beinen ist in (a) links dargestellt. Rechts daneben ist ein Ausschnitt der invertierten Listen an einer Position, an der diese Ski-Bewegung vorkommt, zu sehen. In (b) ist eine korrekte Ausführung der Bewegung mit Pose und invertierten Listen dargestellt.

nicht wie allgemein üblich gegenläufig bewegt. Dies ist auch in Abbildung 5.12(a) zu sehen. Dort ist eine Pose aus Dokument 3 abgebildet: Das rechte Bein und die rechte Hand befinden sich vorne. Die invertierten Listen der Keyframes für die skier1RepsLstart-Klasse und einen Ausschnitt des Dokumentes 3 sind dort zu sehen. Eine waagerechte, grüne Linie zeigt ein Segment einer invertierten Liste eines Keyframes an. Die Höhe in Bezug auf die senkrechte Achse gibt an, um welchen Keyframe es sich handelt: Unten befindet sich die Liste des ersten Keyframes, oben die Liste des 5. Keyframes. Die geforderten Abstände der Ski-Keyframes sind $5, 3, 2$ und 9. Die Stiffnesswerte für die Abstände betragen $0.7, 0.5, 0.5$ und 0.5. So kann der erste Keyframe zu Frame 347 passen, der zweite zu Frame 352 und der dritte zu Frame 354. Der dritte Keyframe beschreibt eine Pose, in der sich der rechte Fuß hinter dem linken Bein befindet und sich beide Füße schnell bewegen (siehe Anhang B). Es wird die Luft-Phase der Füße beschrieben, in welcher sich der rechte Fuß nach vorne bewegt. Bei einer korrekten Ausführung der Ski-Bewegung tritt gleichzeitig eine Pose auf, in welcher sich die linke Hand nach vorne und oben bewegt. Der vierte Keyframe beschreibt genau eine solche Pose. In der hier beschriebenen Ausführung der Ski-Bewegung tritt nun aber eine Pose auf, in welcher sich die rechte Hand nach oben bewegt, weshalb der vierte Keyframe nicht im direkten Anschluss an Frame 545 auftritt. Er tritt zeitversetzt bei der umgekehrten Fußbewegung auf. Somit wird hier kein Treffer gefunden. Abbildung 5.12(b) zeigt dagegen eine korrekte Ausführung der Ski-Bewegung und die entsprechende invertierte Liste. Hier ist gut zu sehen, dass der vierte Keyframe im Gegensatz zu 5.12(a) zeitversetzt auftritt und somit die Keyframes exakt zu den invertierten Listen passen. Dementsprechend wird hier ein Treffer gefunden.

Bei der Durchsicht der False Positives der Anfrage *kickLSide1Reps_punchRFront1Reps_-punchRSide1Reps* fällt ein Treffer im Dokument `HDM_tr_05-01_03_120.amc` dadurch auf,

dass dieses Dokument die komplette Szene 03-01 aus dem Drehbuch (Anhang A) enthält. Dieses Dokument wurde beim Aufbau der Datenbank fälschlicherweise als zu der Szene 05-01 dazugehörig eingeordnet.

Bei der Anfrage *jogLeftCircle4StepsRstart_jogRightCircle4StepsRstart* sind zwei der drei False Negatives auffällig. In den Dokumenten `HDM_bk_01-03_01_120.amc` und `HDM_bk_01-03_02_120.amc` werden keine Treffer gefunden, obwohl das Drehbuch für diese Szene 01-03 die angefragten Bewegungen vorschreibt. In der Tat sind in diesen beiden Dokumenten die Reihenfolge der Ausführung der beiden Anfragebewegungen vertauscht, wodurch kein Treffer gefunden werden konnte (und sollte).

Zusammenfassend sind die Retrieval-Ergebnisse also noch besser als in den automatisch generierten Precision-Recall - Diagrammen (Abbildung 5.11) dargestellt.

Mit der Anfrage *walkOnPlace2StepsLStart_runOnPlaceStartFloor2StepsRStart* wird ein Szenario gestellt, in welchem die Anfangs-Bewegung (walkOnPlace) und die End-Bewegung (runOnPlace) vorgegeben wird und durch einen hohen Abstand (5.5 bis 6.5 Sekunden) zwischen den Bewegungen definiert wird, dass zwischen den gegebenen Bewegungen eine beliebige Bewegung ausgeführt werden kann. Wie im Drehbuch zu der Datenbank (Anhang A) zu erkennen ist, treten die gegebenen Bewegungen in Szene 01-02 mit der Bewegung jogOnPlace als Zwischenbewegung auf. Die manuelle Durchsicht der Treffer zeigt, dass die als korrekt klassifizierten Treffer diese Bewegungssequenz enthalten. Ebenso wird in dem Szenario *lieDownFloor_lieDownFloor* mit einem hohen Abstand von 2.5 bis 7.5 Sekunden zwischen den Bewegungen definiert, dass eine Zwischenbewegung erfolgen kann. Es werden alle Bewegungen aus der Szene 04-01 gefunden, wobei zwischen den beiden geforderten Bewegungen die Person aufsteht und ein paar Schritte geht.

Insgesamt ist mit Keyframes und definierbaren Abständen mit Stiffness-Parametern ein intuitives Werkzeug gegeben, mit welchem effizient in einer großen Dokumentenkollektion von Motion Capture Dokumenten semantisch gesucht werden kann.

5.4.3 Diskussion des Stiffness-Parameters

In den bisherigen Experimenten wurden für Retrieval-Aufgaben die semi-automatisch erstellten Keyframes aus Anhang B benutzt. Bei diesen Keyframes wurde auch der Stiffness-Parameter, welcher die Flexibilität des Abstandes zwischen zwei Keyframes definiert, manuell gesetzt. In diesem Kapitel soll untersucht werden, wie sich Retrieval-Ergebnisse verhalten, wenn die manuell gesetzten Stiffness-Werte σ durch verschiedene Konstanten ersetzt werden. Beim Verändern der Stiffness-Werte der zusammengesetzten Anfragen werden dabei nur die Stiffness-Parameter innerhalb der einzelnen Bewegungen modifiziert und nicht der Stiffness-Wert, welcher die Variation des Abstandes zwischen den einzelnen Bewegungen festlegt. Damit wird erreicht, dass bei sehr hoch eingestelltem σ schlechte Recall-Werte auf zu hohe Stiffness innerhalb der einzelnen Bewegungen zurückzuführen sind und nicht darauf, dass der Abstand zwischen den einzelnen Bewegungen zu strikt festgelegt wurde.

Die Abbildungen 5.13, 5.14 und 5.15 enthalten jeweils 6 Precision-Recall-Diagramme für 6 verschiedene Stiffness-Werte. Die verwendeten Stiffness-Werte sind $\sigma = 0, 0.2, 0.4, 0.6, 0.8$ und

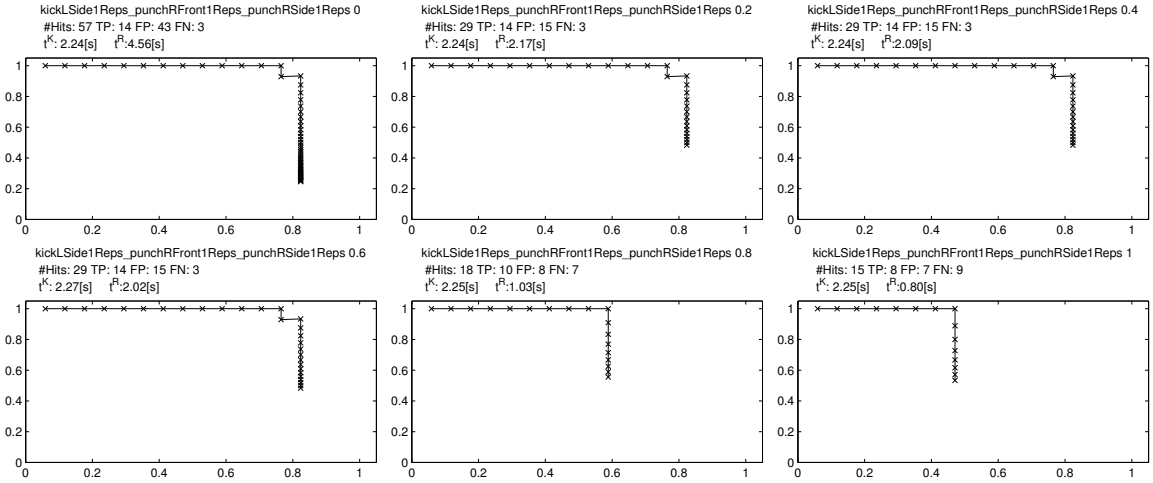


Abbildung 5.13. Precision-Recall-Diagramme der Anfrage kickLSide1Reps_punchRFront1Reps_-punchRSide1Reps an die Datenbank HDM05_amc (3.5 Stunden Motion Capture Material) mit verschiedenen Stiffness-Parametern ($\sigma = 0, 0.2, 0.4, 0.6, 0.8$, 1). #Hits: Anzahl der gefundenen Treffer. TP: Anzahl der korrekt gefundenen Treffer (True Positives). FP: Anzahl der False Positives. FN: Anzahl der False Negatives. t^K: Zeit für die keyframebasierte Suche in Sekunden. t^R: Zeit für das Ranken der reduzierten Datenbank in Sekunden.

1. In dem Titel jedes Diagrammes ist in der jeweils ersten Zeile rechts der verwendete Stiffness-Parameter eingetragen. Die jeweils zweite Zeile gibt Informationen über die gefundenen Treffer. Zuerst wird die Anzahl der insgesamt gefundenen Treffer angegeben. Danach bezeichnet TP, wie viele der gefundenen Treffer korrekte Treffer sind. FP nennt die Anzahl der falschen Treffer. Schließlich gibt FN die Anzahl der relevanten Dokumente an, in denen fälschlicherweise kein Treffer gefunden wird. Die jeweils dritte Zeile enthält Informationen über die Zeit, die für die Beantwortung der Anfragen benötigt wird. Der erste Wert gibt dabei die Zeit für die Keyframesuche, der zweite Wert die Zeit für das Ranken der reduzierten Datenbank in Sekunden an.

Für die Anfrage kickLSide1Reps_punchRFront1Reps_punchRSide1Reps ist in Abbildung 5.13 klar erkennbar, dass mit dem Anstieg der Stiffness die Anzahl der gefundenen Treffer sinkt: Werden bei $\sigma = 0$ insgesamt 57 Treffer gefunden, so bleiben bei $\sigma = 1$ noch 29 Treffer übrig. Gleichzeitig sinkt aber auch der Recall-Wert. Im Übergang von $\sigma = 0.6$ zu $\sigma = 0.8$ sinkt die Anzahl der gefundenen relevanten Dokumente von über 80% auf unter 60%. Bei $\sigma = 1.0$ beträgt sie sogar nur noch weniger als 50%. Im Stiffness-Bereich von $\sigma = 0.2$ bis 0.6 verändert sich der Recall-Wert nicht. Die Größe der reduzierten Datenbank sinkt jedoch leicht, was an den sinkenden Zeiten für das Ranken abgelesen werden kann. In diesem Beispiel kann also $\sigma = 0.6$ gewählt werden. Eine niedrigere Stiffness führt zu mehr False Positives und längeren Retrievalzeiten, eine höhere Stiffness führt zu mehr False Negatives. Die Retrieval-Ergebnisse sind für dieses σ mit den Ergebnissen vergleichbar, die mit manuell eingestellter Stiffness gebildet werden: Abbildung 5.11 zeigt für die beschriebene Anfrage das gleiche Diagramm, die Retrieval-Zeiten sind sehr ähnlich.

Die Anfrage lieDownFloor_lieDownFloor (Abbildung 5.14) weist einen ähnlichen Verlauf der Precision-Recall-Diagramme auf. Auch hier wird bei $\sigma = 0.6$ eine optimale Precision und

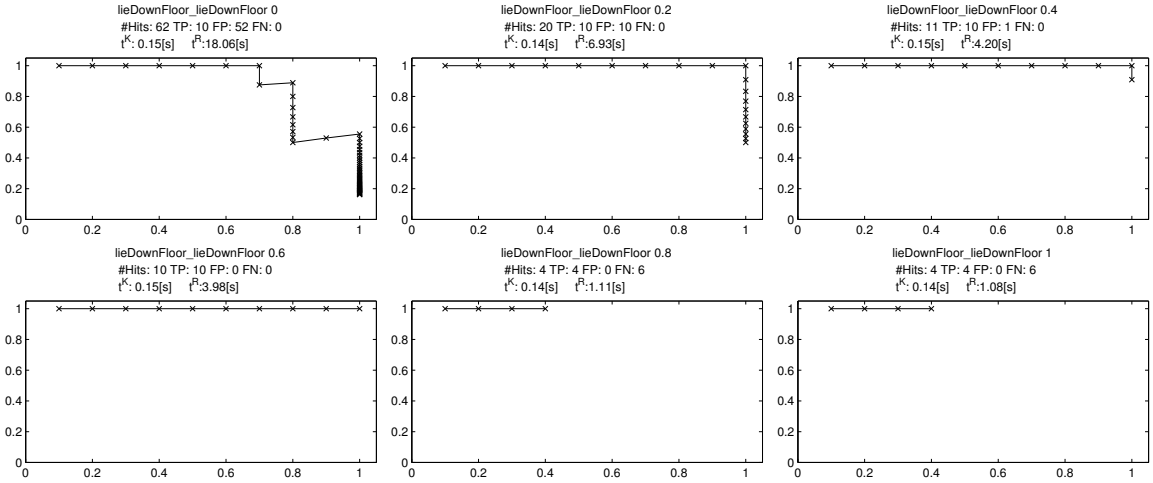


Abbildung 5.14. Precision-Recall-Diagramme der Anfrage lieDownFloor_lieDownFloor an die Datenbank HDM05_amc (3.5 Stunden Motion Capture Material) mit verschiedenen Stiffness-Parametern ($\sigma = 0, 0.2, 0.4, 0.6, 0.8$, 1). #Hits: Anzahl der gefundenen Treffer. TP: Anzahl der korrekt gefundenen Treffer (True Positives). FP: Anzahl der False Positives. FN: Anzahl der False Negatives. t^K: Zeit für die keyframebasierte Suche in Sekunden. t^R: Zeit für das Ranken der reduzierten Datenbank in Sekunden.

Recall erreicht. Auffällig ist, dass im Sprung von $\sigma = 0$ auf $\sigma = 0.2$ die Precision stark ansteigt. Hier werden durch die Stiffness 42 nicht relevante Dokumente aus der Trefferliste eliminiert. Im Vergleich mit den Ergebnissen bei manuell optimierter Stiffness fällt bei $\sigma = 0.6$ auf, dass genau wie bei den manuellen Stiffness-Werten alle relevanten Dokumente gefunden werden, jedoch die Zeit für das Ranken hier sogar geringer als bei den manuellen Stiffness-Werten ist. Die manuell eingestellte Stiffness bei der lieDownFloor-Bewegung ist $\sigma = 0.5$ für alle Keyframeabstände. Offensichtlich werden hier bessere Retrieval-Zeiten bei gleichbleibender Qualität der Ergebnisse durch die Wahl von $\sigma = 0.6$ erreicht.

Schließlich wird die Anfrage jogLeftCircle4StepsRstart_jogRightCircle4StepsRstart (Abbildung 5.15) untersucht. Auch hier ist wieder ein starker Rückgang der gefundenen Treffer mit wachsender Stiffness zu beobachten. Auffällig ist, dass schon beim Sprung von $\sigma = 0$ zu $\sigma = 0.2$ zwei relevante Dokumente verloren gehen. Erst bei Werten von $\sigma > 0.6$ steigt die Anzahl der False Negatives stark. Die Retrieval-Ergebnisse und -Zeiten sind bei $\sigma = 0.6$ mit den Ergebnissen aus Abbildung 5.11 vergleichbar.

Die Laufzeiten für die Keyframesuche der drei Anfragen und die Laufzeiten für das Ranken der reduzierten Datenbank sind für die gewählten drei Anfragen in Abbildung 5.16 (Seite 91) zu sehen. Bei der Darstellung werden die Messpunkte mit Linien verbunden, um die Zugehörigkeit der Messpunkte zu einer Messreihe anzudeuten. Man sieht in 5.16(a) deutlich, dass die Zeiten für die keyframebasierte Suche kaum von der Stiffness abhängen. Viel mehr Einfluss auf die Laufzeit hat hier, wie in Abschnitt 5.4.1 diskutiert, die Länge und die Struktur der invertierten Listen untereinander. Die Laufzeiten sinken nur sehr leicht mit wachsender Stiffness. Es ist klar ersichtlich, dass die Ranking-Zeiten (Abbildung 5.16(b)) mit wachsender Stiffness abfallen, da die Größe der reduzieren Datenbank mit wachsender Stiffness stärker eingeschränkt werden kann. Somit fällt der zeitintensive Ranking-Schritt mit wachsender Stiff-

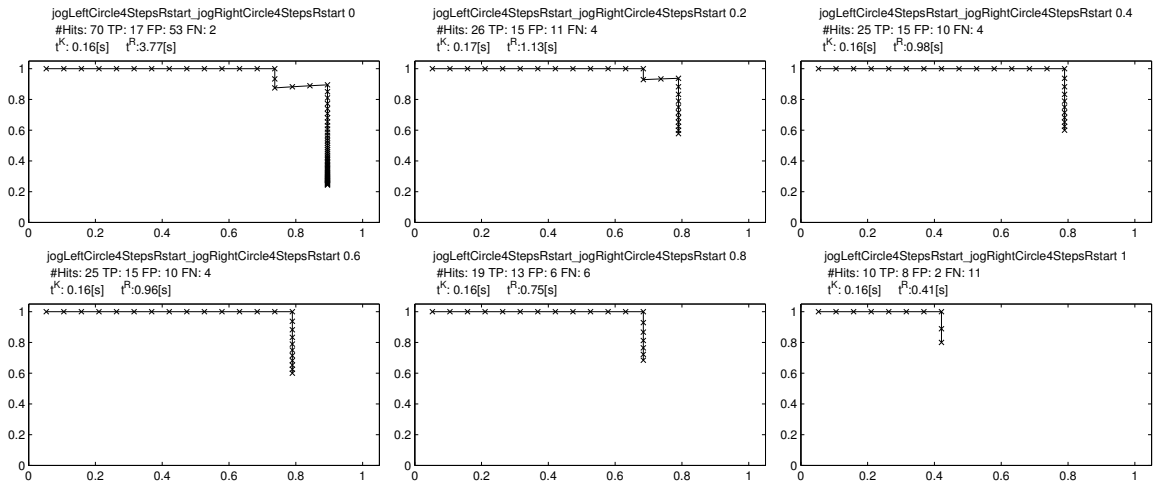
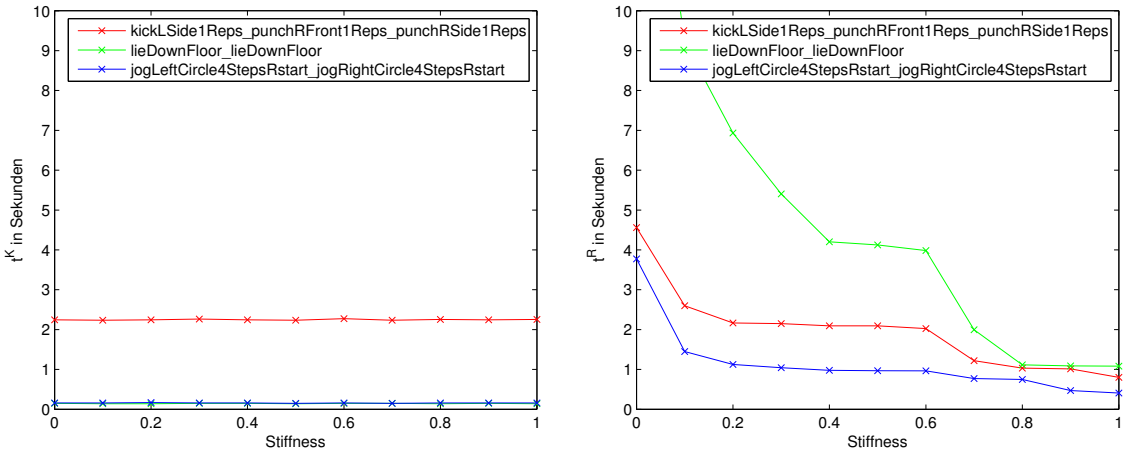


Abbildung 5.15. Precision-Recall-Diagramme der Anfrage jogLeftCircle4StepsRstart_jogRightCircle4StepsRstart an die Datenbank HDM05_amc (3.5 Stunden Motion Capture Material) mit verschiedenen Stiffness-Parametern ($\sigma = 0, 0.2, 0.4, 0.6, 0.8$, 1). #Hits: Anzahl der gefundenen Treffer. TP: Anzahl der korrekt gefundenen Treffer (True Positives). FP: Anzahl der False Positives. FN: Anzahl der False Negatives. t^K: Zeit für die keyframebasierte Suche in Sekunden. t^R: Zeit für das Ranken der reduzierten Datenbank in Sekunden.

ness immer weniger ins Gewicht. Die Auswirkung der Stiffness ist jedoch für die verschiedenen Anfragen sehr unterschiedlich. Der sehr starke Effekt auf die lieDownFloor-Anfrage steht der eher schwachen Wirkung auf die jogLeftCircle-Anfrage gegenüber. Dies kann mit der Länge der Anfrage erklärt werden. Für die lieDown-Anfrage werden Treffer von insgesamt 470 Frames Länge erwartet. Dies wird mit den Abständen der Keyframes definiert. Das MT, das zum Ranken der Treffer benutzt wird, hat ebenfalls diese Länge. Für die Kick-Anfrage beträgt sie 272, die Walk-Anfrage hat eine Länge von 211 Frames. Je länger die Anfrage ist, desto länger sind auch die erwarteten Ausschnitte der Datenbank nach der Keyframesuche und die lange Laufzeit des Nachverarbeitungsschrittes trägt den größten Teil der Gesamtlaufzeit. So zeigt es sich, dass gerade bei längeren Anfragen die Wahl einer nicht zu kleinen Stiffness wichtig für eine kurze Gesamtlaufzeit ist.

In den vorgestellten Beispielen sind die Retrieval-Ergebnisse und -Zeiten bei automatischer Wahl der Stiffness von $\sigma = 0.6$ mit den Ergebnissen bei manueller Wahl der Stiffness vergleichbar. Es ist klar, dass je nach zeitlicher Inhomogenität der Bewegungsklasse auch bei kleineren Stiffness-Werten viele False Negatives entstehen können. Wie die Beispiele jedoch zeigen, kann man für einen ersten Ansatz den Stiffness-Wert auf eine Konstante von 0.6 festlegen, ohne dabei die Retrieval-Ergebnisse wesentlich zu verschlechtern. Für die Gewinnung der bestmöglichen Retrieval-Ergebnisse ist es jedoch erforderlich, den Stiffness-Wert für den Abstand von je zwei Keyframes zu optimieren. Kapitel 6 schlägt einen ersten Ansatz zum automatischen Gewinnen von Keyframes vor. Darin wird auch eine Methode beschrieben, welche die Keyframe-Abstände und die Stiffness-Werte für eine Trainingsmenge optimiert.



(a) (b)

Abbildung 5.16. Für die Laufzeit der keyframebasierten Suche, aufgetragen in Sekunden, (gezeigt in (a)) spielt der Stiffness-Parameter praktisch keine Rolle. Betrachtet man die Laufzeit des Ranking-Schrittes, aufgetragen in Sekunden in (b), so sinken die Laufzeiten stark mit wachsender Stiffness. Gleichzeitig werden bei zu hoher Stiffness aber auch relevante Treffer nicht mehr gefunden, wie an den vorhergehenden Precision-Recall-Diagrammen abgelesen werden kann.

Kapitel 6

Automatische Bestimmung von Keyframes

Für gute Ergebnisse beim keyframebasierten Retrieval ist die Qualität der benutzten Keyframes entscheidend. Sind die Keyframes nicht aussagekräftig genug für die Bewegungsklasse, so sind die nach der Keyframesuche aus der Datenbank ausgewählten Bereiche groß und die hohe Laufzeit des Ranking-Schrittes führt zu Laufzeiten von mehreren Minuten bei einer Datenbankgröße von mehreren Stunden. Schränken die Keyframes die Klasse zu stark ein, so werden relevante Treffer eventuell nicht mehr gefunden. Um die Mächtigkeit der keyframebasierten Suche zu demonstrieren, wurden in dieser Arbeit möglichst gute Keyframes von Hand erzeugt. Eine Automatisierung der Keyframegenerierung mir Hilfe von Trainingsbewegungen würde diesen Schritt wesentlich einfacher gestalten.

Ein erster Ansatz zum automatischen Generieren von Keyframes kann mit Hilfe von harten MTs geschehen. Wie in Kapitel 3.3.2 gezeigt wurde, führt der DTW-Vergleich eines harten MTs mit der Trainingsmenge zu keinen Kosten. Dies bedeutet, dass alle Spalten des harten MTs in jeder der Trainingsbewegungen in der selben Reihenfolge wie im harten MT vorkommen. Dabei können für „graue“ Bereiche (Wert 0.5) in den Spalten der MTs die Featurematrizen der Trainingsmenge beliebige Werte annehmen. Wählt man nun Vektoren aus einem harten MT als Keyframes, so ist garantiert, dass man auf der Trainingsmenge durch die Wahl der Keyframes keine False Negatives erzeugt. Die Abstände der Keyframes zusammen mit den Stiffness-Parametern können nun ebenfalls automatisiert gewählt werden. Dazu müssen die Positionen, an denen die Keyframes in der Trainingsmenge auftreten, bestimmt werden. Nun kann aus dem größten auftretenden Abstand M und dem kleinsten auftretenden Abstand m von je zwei aufeinander folgenden Keyframes der Keyframeabstand δ mit Stiffness σ berechnet werden, so dass alle in den Trainingsbewegungen vorkommenden Abstände innerhalb des Bereiches $[\delta \cdot \sigma, \delta/\sigma]$ liegen. Dies kann durch eine einfache Rechnung geschehen:

$$\delta = \sqrt{m \cdot M}$$
$$\sigma = \sqrt{m/M}.$$

Ist beispielsweise $m = 4$, $M = 16$, so ist $\delta = \sqrt{4 \cdot 16} = \sqrt{64} = 8$ und $\sigma = \sqrt{4/16} = 0.5$ zu wählen. Mit dieser Wahl des Abstandes und der Stiffness werden mit Algorithmus 4.1 alle auftretenden Abstände der Keyframes in der Trainingsmenge berücksichtigt und die Wahl von δ und σ verursacht keine False Negatives in der Trainingsmenge. Dadurch ist die Chance groß, auch keine relevanten Treffer auf beliebigen Daten zu verlieren.

Abbildung 6.1(a) zeigt das harte MT der Klasse walkBackwards2StepsRstart. Die Featuremengen „lower“, „upper“ und „mix“ wurden dabei getrennt dargestellt. Nun können aus

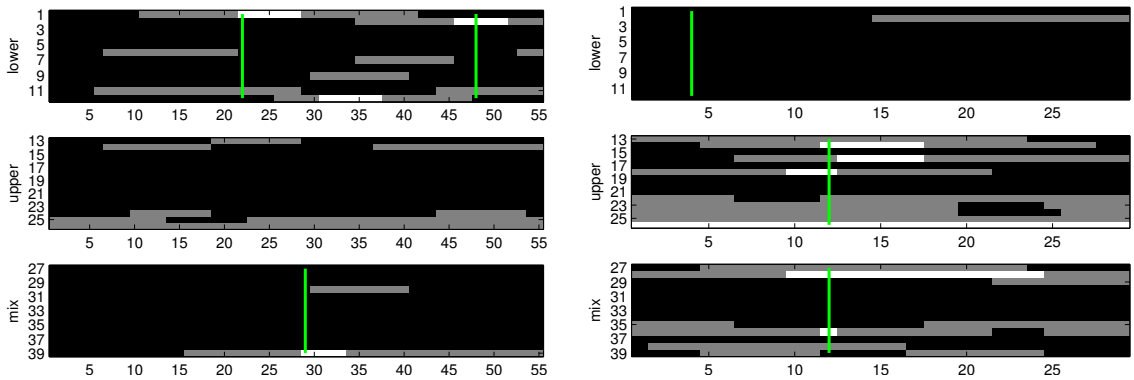


Abbildung 6.1. Die senkrechten grünen Linien zeigen an, welche Keyframes aus den harten MTs automatisch ausgewählt wurden. In (a) ist das harte MT der Klasse walkBackwards2StepsRstart gezeigt und (b) zeigt das harte MT der Klasse rotateArmsLForward1Reps.

diesem harten MT die Keyframes nach einer einfachen Heuristik ausgewählt werden. Es ist klar, dass alle Einsen (weiße Bereiche) in einem harten MT charakteristische Eigenschaften in einem Feature anzeigen, die konsistent über alle Trainingsbewegungen sind. Alle Nullen (schwarze Bereiche) zeigen, dass in diesem Feature die Bewegung nicht von der Standpose zu unterscheiden ist. Bereiche, die den Wert 0.5 annehmen, bedeuten eine Inkonsistenz bezüglich des entsprechenden Features. Mit diesem Wissen wird für jede Spalte des harten MTs eine Punktzahl vergeben, die sich nach der Anzahl der weißen und schwarzen Einträge richtet. Einem weißen Eintrag werden nach einer heuristischen Überlegung dreimal so viele Punkte wie einem schwarzen Eintrag zugewiesen, um Keyframes mit charakteristischen, weißen Einträgen zu bevorzugen. Graue Einträge tragen zu den Punkten keinen Wert bei. Werden nun diejenigen Spalten mit der höchsten Punktzahl als Keyframes ausgewählt, so erhält man die für die Bewegung charakteristischen Spalten aus dem harten MT als Keyframes. Eine Schwierigkeit besteht nun darin, zu bestimmen, wie viele Keyframes benötigt werden, um die Bewegung genau zu beschreiben. In diesem Beispiel werden 3 Keyframes ausgewählt.

Abbildung 6.1 zeigt zwei harte MTs und die für sie automatisch ausgewählten Keyframes. Wie man sieht, funktioniert die Auswahl in (a) sehr gut: Das Muster der Gehbewegung wird durch die Keyframes repräsentiert. In (b) sieht man, dass für die Klasse rotateArmsLForward1Reps die automatische Auswahl mit Hilfe von harten MTs fehlgeschlagen ist. Die Klasse ist in der „upper“-Featuremenge so inkonsistent, dass dort das harte MT einen sehr hohen Grauanteil hat und die ausgewählten Keyframes damit nicht aussagekräftig für diese Klasse sind. Retrieval-Tests mit so ausgewählten Keyframes zeigen, dass die Datenbank nicht gut verkleinert wird und im Ranking-Schritt immer noch ein Großteil der Datenbank betrachtet werden muss, wodurch die Laufzeiten für das Retrieval sehr groß werden.

Kapitel 7

Zusammenfassung und Ausblick

In dieser Diplomarbeit wurde ein Algorithmus vorgestellt, mit dem in einer großen Datenbank unter Benutzung von Keyframes effizient gesucht werden kann. Die Keyframes drücken Schlüsseleigenschaften einer Bewegungsklasse aus und bei der Suche werden die zeitlichen Abstände dieser berücksichtigt. Trotz der Einführung einer zeitlichen Variabilität der Keyframes ist die Laufzeitkomplexität des vorgestellten Algorithmus linear in der Summe der Längen aller benutzten Keyframelisten, wie in Abschnitt 4.2.3 bewiesen wurde.

Die Experimente, die in Abschnitt 5.4 beschrieben werden, basieren auf realistischen Anwendungsszenarien. Hier zeigt sich, dass der vorgestellte Algorithmus zur keyframebasierten Suche in der Praxis einen enormen Effizienzgewinn gegenüber klassischen DTW-basierten Methoden bietet. Bei der verwendeten Motion-Capture-Datenbank, die 3.5 Stunden Material enthält, bewegen sich die Suchzeiten im Rahmen weniger Sekunden. Der hier entwickelte rekursive Algorithmus trägt mit Laufzeiten im Millisekundenbereich nur einen geringen Prozentsatz an der Laufzeit der keyframebasierten Suche. Bisher wurde ein Index benutzt, der nicht für die keyframebasierte Suche optimiert ist. Verbessert man die Indexstruktur, so dass zeitintensive Konvertierungsoperationen nicht mehr durchgeführt werden müssen, so ist zu erwarten, dass sich die Laufzeit stark verkürzt. Mehr noch ist die Qualität der Suchergebnisse, verwendet man einen DTW-basierten Ranking-Schritt als Nachverarbeitung, vergleichbar mit rein DTW-basierten Verfahren. Die DTW-basierte Nachverarbeitung bewegt sich bei der verwendeten Datenbank in Größenordnungen von wenigen Sekunden. Wegen der Benutzung eines Indexes ist die praktische Laufzeit des vorgestellten Algorithmus im Wesentlichen davon abhängig, wie häufig die mit den Keyframes beschriebenen Posen in der Datenbank vorkommen. In dieser Arbeit wurden semi-automatisch generierte Keyframes benutzt.

Eine wichtige, noch offene Fragestellung ist, wie die benötigten Schlüsseleigenschaften einer Klasse von Bewegungen automatisch extrahiert werden können. Zu der Lösung dieser Fragestellung können harte MTs, wie sie in Abschnitt 3.3.1 vorgestellt werden, aufgrund ihrer in Abschnitt 3.3.2 bewiesenen Eigenschaften beitragen. In den bisher durchgeführten Experimenten mit so generierten Keyframes erwiesen sich diese als nicht einschränkend genug für eine Klasse, so dass die Laufzeiten in dem anschließenden Ranking-Schritt sehr groß wurden. Um die Suchergebnisse der keyframebasierten Suche weiter einzuschränken, können zusätzliche Zwangsbedingungen realisiert werden. An harten MTs lassen sich Eigenschaften wie „Dieses Merkmal nimmt während der gesamten Bewegung den Wert 0 an“ oder „Dieses Merkmal muss über einen längeren Zeitraum den Wert 1 annehmen“ leicht ablesen. Gelingt es, diese Zwangsbedingungen effizient zu modellieren, so könnten sie unterstützend zu automatisch generierten Keyframes in einem kombinierten Suchverfahren die Qualität der Suchergebnisse weiter verbessern.

Anhang A

Datenbanken der Bewegungsklassen

A.1 HDM05_cut_57 oder auch $\mathcal{D}^{57}$

In dieser Diplomarbeit wurden mehrere Datenbanken von Bewegungsdaten verwendet. Die Datenbank, die zum Lernen der Motion Templates und als Grundlage für die Keyframes dient, ist die Datenbank $\mathcal{D}^{57}$. Sie basiert auf der in [MRC+07] vorgestellten HDM05_cut_amc-Datenbank. Die HDM05_cut_amc-Datenbank enthält ca. 130 verschiedene Bewegungsklassen. Für die meisten dieser Klassen sind 10 bis 50 unterschiedliche Repräsentanten der Klasse vorhanden, jeder dieser Repräsentanten ist mit einer Framerate von 120 Hz vorhanden. In der Datenbank $\mathcal{D}^{57}$ wurde nun eine Auswahl unter den verschiedenen Klassen getroffen. Die HDM05_cut_amc-Daten sind für viele Klassen in mehreren Wiederholungen vorhanden. So gibt es beispielsweise die Klassen cartwheelLHandStart1Reps und cartwheelLHandStart2Reps. In der Datenbank $\mathcal{D}^{57}$ wurden systematisch nur die Bewegungen mit der jeweils geringsten Wiederholungszahl übernommen. Die Motivation hierfür ist, dass nach dem Lernen der Motion Templates für eine einzelne Ausführung der Bewegung ein Template für mehrere Ausführungen erstellt werden kann, indem das Motion Template wiederholt wird. Des Weiteren wurden in der Datenbank folgende kleine Fehler behoben:

kickLSide1Reps Die Bewegungen `HDM_mm_kickLSide1Reps_018_120.amc`, `HDM_mm_kickLSide1Reps_019_120.amc` und `HDM_mm_kickLSide1Reps_020_120.amc` sind kickLFront1Reps-Bewegungen. Diese wurden aus der Kategorie kickLSide1Reps gelöscht und in der Kategorie kickLFront1Reps als `HDM_mm_kickLFront1Reps_030_120.amc`, `HDM_mm_kickLFront1Reps_031_120.amc` und `HDM_mm_kickLFront1Reps_032_120.amc` eingefügt.

kickRSide1Reps Die Datei `HDM_bk_kickRSide1Reps_009_120.amc` enthält keine vollständige Bewegung und wurde gelöscht.

shuffle2StepsRStart Die Bewegung `HDM_dg_shuffle2StepsRStart_005_120.amc` ist eine shuffle4StepsRStart - Bewegung. Sie wurde aus der Kategorie shuffle2StepsRStart gelöscht und als `HDM_dg_shuffle4StepsRStart_014_120.amc` in die Kategorie shuffle4StepsRStart verschoben.

turnRight Die Bewegung `HDM_dg_turnRight_022_120.amc` ist eine turnLeft-Bewegung. Sie wurde aus dieser Klasse gelöscht und als `HDM_dg_turnLeft_031_120.amc` in die Klasse turnLeft verschoben.

Statistische Details dieser Datenbank sind in der Tabelle A.1, Seite 99 nachzulesen. Insgesamt sind 41 Minuten Motion-Capture-Material in der Datenbank enthalten, davon sind wird die Hälfte für die Trainingsdatenbank und die andere Hälfte für die Evaluationsdatenbank verwendet.

Die verwendeten Namen für die Bewegungsklassen enthalten bedeutungsvolle Abkürzungen, die an ein paar Beispielen erklärt werden:

cartwheelLHandStart1Reps „cartwheel“ gibt die Bewegungsklasse an. Es handelt sich um einen Radschlag. „LHandStart“ bedeutet, dass die Bewegung mit der linken Hand gestartet wird. Schließlich gibt „1Reps“ an, dass nur eine Wiederholung der Bewegung ausgeführt wird.

kickRFront1Reps Es handelt sich um eine „kick“-Bewegung, also einen Fußtritt. Der Tritt wird gemäß „RFront“ mit dem rechten Fuß ausgeführt und es wird nach vorne getreten. „1Reps“ gibt schließlich wieder an, dass nur eine Wiederholung eines Trittes ausgeführt wird.

runOnPlaceStartFloor2StepsRStart Diese Klasse enthält Bewegungen, die auf der Stelle rennen. Dabei werden zwei Renn-Schritte ausgeführt und mit dem rechten Fuß auf dem Boden gestartet.

A.2 HDM05_amc

Für die Experimente auf ungeschnittenen Daten wurde die Datenbank HDM05_amc aus [MRC+07] benutzt. Sie enthält ca. 3.5 Stunden Motion Capture-Material in 120 Hz-Framerate. Für die Experimente wurde die Abtastrate der Datenbank auf 30 Hz geändert. Damit enthält sie 380838 Frames. Die Daten wurden mit Hilfe eines Drehbuches aufgenommen, welches 5 Teile enthält. Jeder Teil ist wiederum aufgegliedert in mehrere Szenen. Das Drehbuch, nach dem die Schauspieler die Bewegungen ausgeführt haben, ist vollständig in [MRC+07] nachzulesen. Hier sind diejenigen Ausschnitte des Skriptes wiedergegeben, die für Experimente in Kapitel 5.4 benutzt wurden.

1 Walking, Running, Jumping

1-1 Walking: [1] walk 5 steps [2] turn around (right) [3] walk 5 steps (ducked) [4] walk 5 steps (backwards) [5] walk 5 steps (sideways, to the right, feet cross over alternately front/back) [6] 3 double steps (sideways, to the left, no cross over) [7] 3 double steps (sideways, to the right, cross over only front) [8] walk 5 steps (happily) [9] turn around (left) [10] walk 5 steps (sadly) [11] turn around (right) [12] walk 5 steps (creep) [13] turn around [14] walk 5 steps (shuffle)

1-2 Locomotion on the spot: [1] walk 5 steps on spot [2] jog 5 steps on spot [3] run 5 steps on spot [4] bend knees [5] walk 5 steps with bent knees

1-3 Locomotion: [1] walk 6 steps (semicircle left) [2] turn around [3] walk 6 steps (semicircle right), back to start [4] turn around [5] transition: walking to running [6] turn around [7] run 5 steps (semicircle left) [8] run 5 steps (semicircle right), back to start

1-6 Climbing stairs: [1] walk 5 steps [2] climb 4 stairs [3] turn around (right) [4] descend 4 stairs [5] walk 5 steps

Nr.	Bewegungsklasse	N	av $\#f$	N_E	av $\#f_E$	N_T	av $\#f_T$
1	cartwheelLHandStart1Reps	21	411	10	398	11	422
2	clap1Reps	17	43	8	42	9	44
3	clapAboveHead1Reps	17	94	8	93	9	95
4	depositFloorR	32	363	16	434	16	293
5	depositHighR	28	245	14	223	14	268
6	elbowToKnee1RepsLelbowStart	27	147	13	139	14	154
7	elbowToKnee1RepsRelbowStart	27	149	13	150	14	148
8	grabFloorR	16	269	8	291	8	247
9	grabHighR	29	259	14	242	15	274
10	hopBothLegs1hops	36	96	18	97	18	95
11	hopLLeg1hops	41	75	20	75	21	75
12	hopRLeg1hops	42	74	21	75	21	73
13	jogLeftCircle4StepsRstart	17	244	8	245	9	243
14	jogOnPlaceStartFloor2StepsRStart	14	115	7	114	7	116
15	jogRightCircle4StepsRstart	17	237	8	236	9	239
16	jumpDown	14	282	7	298	7	267
17	jumpingJack1Reps	52	144	26	145	26	144
18	kickLFront1Reps	32	213	16	221	16	205
19	kickLSide1Reps	23	238	11	234	12	241
20	kickRFront1Reps	30	221	15	227	15	215
21	kickRSide1Reps	29	229	14	251	15	209
22	lieDownFloor	20	655	10	649	10	661
23	punchLFront1Reps	30	198	15	199	15	197
24	punchLSide1Reps	30	175	15	185	15	165
25	punchRFront1Reps	30	215	15	215	15	215
26	punchRSide1Reps	28	180	14	195	14	166
27	rotateArmsBothBackward1Reps	16	109	8	106	8	112
28	rotateArmsBothForward1Reps	16	116	8	113	8	120
29	rotateArmsLBackward1Reps	16	108	8	104	8	112
30	rotateArmsLForward1Reps	16	108	8	102	8	113
31	rotateArmsRBackward1Reps	16	105	8	102	8	109
32	rotateArmsRForward1Reps	16	106	8	101	8	112
33	runOnPlaceStartFloor2StepsRStart	15	85	7	85	8	85
34	shuffle2StepsRStart	12	247	6	249	6	245
35	sitDownChair	20	319	10	310	10	328
36	sitDownFloor	20	408	10	394	10	422
37	sitDownKneelTieShoes	17	646	8	638	9	653
38	sitDownTable	20	271	10	268	10	273
39	skier1RepsLstart	30	141	15	140	15	143
40	sneak2StepsLStart	16	236	8	241	8	230
41	sneak2StepsRStart	16	279	8	298	8	259
42	squat1Reps	52	193	26	195	26	191
43	staircaseDown3Rstart	15	223	7	228	8	218
44	staircaseUp3Rstart	28	296	14	282	14	310
45	standUpLieFloor	20	525	10	530	10	520
46	standUpSitFloor	20	403	10	437	10	370
47	throwBasketball	14	408	7	435	7	380
48	turnLeft	31	195	15	192	16	198
49	turnRight	29	198	14	188	15	208
50	walk2StepsLstart	31	155	15	149	16	162
51	walk2StepsRstart	31	166	15	178	16	154
52	walkBackwards2StepsRstart	15	217	7	207	8	226
53	walkLeft2Steps	16	323	8	328	8	317
54	walkLeftCircle4StepsRstart	18	328	9	324	9	333
55	walkOnPlace2StepsLStart	15	153	7	151	8	156
56	walkRightCircle4StepsRstart	15	329	7	318	8	339
57	walkRightCrossFront2Steps	16	336	8	338	8	333
	Gesamtzahl	1327	294197	653	146205	674	147992

Tabelle A.1. Details über die Bewegungsklassen der Datenbank D^{57} können hier eingesehen werden. N: Gesamtzahl der Bewegungen. av #f: Durchschnittliche Anzahl Frames pro Bewegung (bei 120Hz) N_E: Anzahl der Bewegungen in der Evaluationsdatenbank. av $\#f_E$: Durchschnittliche Anzahl Frames pro Bewegung in der Evaluationsdatenbank (bei 120Hz). N_T: Anzahl der Bewegungen in der Trainingsdatenbank. av $\#f_T$: Durchschnittliche Anzahl Frames pro Bewegung in der Trainingsdatenbank (bei 120Hz).

3 Sports

3-2 Kicking and punching: [1] 2 kicks (right foot forwards) [2] 2 kicks (right foot sideways) [3] 2 kicks (left foot forwards) [4] 2 kicks (left foot sideways) [5] 2 punches (right hand forwards) [6] 2 punches (right hand sideways) [7] 2 punches (left hand forwards) [8] 2 punches (left hand sideways)

3-4 Rotating arms: [1] 4 forward rotations (right arm) [2] 4 backward rotations (right arm) [3] 4 forward rotations (left arm) [4] 4 backward rotations (left arm) [5] 4 forward rotations (both arms) [6] 4

backward rotations (both arms) [7] 4 swings in front of body (both arms) [8] 4 forward rotations while walking (both arms) [9] turn around [10] 4 backward rotations while walking (both arms)

3-5 Workout: [1] 4 jumping jacks [2] 4 times skiing exercise [3] 4 times elbow-to-knee exercise (start with right elbow to left knee) [4] 4 squats

4 Sitting and Lying Down:

4-1 Chair, table, floor: [1] walk 3 steps to chair [2] sit down on chair [3] stand up [4] walk 3 steps away from chair and turn around [5] walk 3 steps to chair [6] sit down on chair [7] stand up [8] walk 3 steps away from chair and turn around [9] walk 3 steps to table [10] sit down on table [11] stand up [12] walk 3 steps away from table and turn around [13] walk 3 steps to table [14] sit down on table [15] stand up [16] walk 3 steps away from table and turn around [17] walk 3 steps to lying position [18] sit down on floor [19] stand up [20] walk 3 steps away from lying position and turn around [21] walk 3 steps to lying position [22] sit down on floor [23] stand up [24] walk 3 steps away from lying position and turn around [25] walk 3 steps to lying position [26] lie down on floor [27] stand up [28] walk 3 steps away from lying position and turn around [29] walk 3 steps to lying position [30] lie down on floor [31] stand up [32] walk 3 steps away from lying position and turn around

5 Miscellaneous Motions:

5-3 Variations of locomotion: [1] stumbling [2] limping [3] running with acceleration and deceleration [4] cartwheel

Das Skript wurde von mehreren Schauspielern in mehreren Wiederholungen ausgeführt. Jede Wiederholung einer Szene von einem bestimmten Schauspieler (z.B. Szene 5-3, Schauspieler mm) wurde in einer Datei gespeichert. Die Namenskonvention für die Dateinamen für jede Aufnahme sind wie folgt:

`HDM_{actor}_{part}-{scene}_{take}_{framerate}.AMC`

Das Feld `actor` enthält das Namenskürzel der 5 verschiedenen Schauspieler. Die Felder `part` und `scene` geben an, welchen Teil und welche Szene in der Datei enthalten ist. Schließlich gibt das Feld `take` an, um welche Wiederholung des Teils und der Szene es sich handelt, da ein Schauspieler die selbe Szene aus dem Drehbuch mehrfach ausgeführt hat.

Im Folgenden wird wie in [MRC+07] die Liste der enthaltenen Szenen wiedergegeben. Sie ist zuerst nach dem Schauspieler und dann nach der Szene sortiert. In der Spalte „#(fr.)“ ist die Länge der Aufnahme in Frames bezüglich 120 Hz angegeben. Die Spalte „Beschreibung“ gibt an, um welche Szene es sich handelt. In der Spalte „Kommentare“ sind Kommentare zu Unregelmäßigkeiten in den einzelnen Aufnahmen vermerkt. Die Unregelmäßigkeiten in der Datenbank, die währen dieser Diplomarbeit gefunden wurden, sind hinzugefüg worden.

File Name Prefix	#(fr.)	Description	Comments
HDM_bd_01-01_01_120	9842	1-1 Walking	
HDM_bd_01-01_02_120	8091	·	
HDM_bd_01-01_03_120	7965	·	
HDM_bd_01-02_01_120	2900	1-2 Locomotion on the spot	
HDM_bd_01-02_02_120	2864	·	
HDM_bd_01-02_03_120	2923	·	
HDM_bd_01-03_01_120	4025	1-3 Locomotion	
HDM_bd_01-03_02_120	4089	·	
HDM_bd_01-03_03_120	4166	·	
HDM_bd_01-03_04_120	4086	·	
HDM_bd_01-04_01_120	6461	1-4 Locomotion with weights	
HDM_bd_01-04_02_120	7098	·	
HDM_bd_01-04_03_120	6454	·	

File Name Prefix	#(fr.)	Description	Comments
HDM_bd_01-04_04_120	6117	·	
HDM_bd_01-05_01_120	4457	1-5 Hopping and jumping	
HDM_bd_01-05_02_120	4655	·	
HDM_bd_01-05_03_120	4851	·	
HDM_bd_01-05_04_120	4633	·	
HDM_bd_01-05_05_120	4518	·	
HDM_bd_01-06_01_120	2219	1-6 Climbing stairs	
HDM_bd_01-06_02_120	1886	·	
HDM_bd_01-06_03_120	2046	·	
HDM_bd_02-01_01_120	3578	2-1 Table and floor	
HDM_bd_02-01_02_120	3670	·	
HDM_bd_02-01_03_120	3457	·	
HDM_bd_02-02_01_120	6324	2-2 Shelf (while walking)	
HDM_bd_02-02_02_120	6080	·	
HDM_bd_02-03_01_120	2684	2-3 Shelf (while standing)	
HDM_bd_02-03_02_120	2539	·	
HDM_bd_03-02_01_120	3958	3-2 Kicking and punching	
HDM_bd_03-02_02_120	4084	·	
HDM_bd_03-02_03_120	4046	·	
HDM_bd_03-03_01_120	3134	3-3 Throwing	
HDM_bd_03-03_02_120	2764	·	
HDM_bd_03-03_03_120	2765	·	
HDM_bd_03-04_01_120	6334	3-4 Rotating arms	
HDM_bd_03-04_02_120	6190	·	
HDM_bd_03-04_03_120	6126	·	
HDM_bd_03-04_04_120	6073	·	
HDM_bd_03-05_01_120	3316	3-5 workout	
HDM_bd_03-05_02_120	3707	·	
HDM_bd_03-05_03_120	3458	·	
HDM_bd_03-10_01_120	2686	3-10 Rope skipping	
HDM_bd_03-10_02_120	1056	·	
HDM_bd_03-10_03_120	939	·	
HDM_bd_03-11_01_120	4210	3-11 Badminton	
HDM_bd_03-11_02_120	2636	·	
HDM_bd_03-11_03_120	2258	·	
HDM_bd_04-01_01_120	11843	4-1 Chair, table, floor	
HDM_bd_04-01_02_120	10434	·	
HDM_bd_05-01_01_120	5920	5-1 Clapping and waving	
HDM_bd_05-01_02_120	4645	·	
HDM_bd_05-01_03_120	3862	·	
HDM_bd_05-02_01_120	1737	5-2 Shouting and tying shoes	
HDM_bd_05-02_02_120	1758	·	
HDM_bd_05-02_03_120	1828	·	
HDM_bd_05-03_01_120	3280	5-3 Variations of locomotion	
HDM_bd_05-03_02_120	3992	·	
HDM_bd_05-03_03_120	3501	·	
HDM_bd_06-01_01_120	3133	Different boxing and kicking motions	
HDM_bd_06-01_02_120	2490	Handstand	
HDM_bd_06-01_03_120	5659	Different clapping, cheering, and provoking motions	
HDM_bk_01-01_01_120	10282	1-1 Walking	
HDM_bk_01-01_02_120	8747	·	
HDM_bk_01-01_03_120	8969	·	
HDM_bk_01-02_01_120	3421	1-2 Locomotion on the spot	
HDM_bk_01-02_02_120	2834	·	
HDM_bk_01-02_03_120	3406	·	
HDM_bk_01-03_01_120	6707	1-3 Locomotion	Running sequence of semicircle left and semicircle right is interchanged
HDM_bk_01-03_02_120	6211	·	Running sequence of semicircle left and semicircle right is interchanged
HDM_bk_01-03_03_120	5115	·	
HDM_bk_01-03_04_120	4562	·	
HDM_bk_01-03_05_120	4467	·	
HDM_bk_01-04_01_120	7677	1-4 Locomotion with weights	
HDM_bk_01-04_02_120	7752	·	
HDM_bk_01-04_03_120	6722	·	
HDM_bk_01-05_01_120	5364	1-5 Hopping and jumping	
HDM_bk_01-05_02_120	5920	·	
HDM_bk_01-05_03_120	5171	·	

File Name Prefix	#(fr.)	Description	Comments
HDM_bk_01-06_01_120	2964	1-6 Climbing stairs	
HDM_bk_01-06_02_120	2345	·	
HDM_bk_01-06_03_120	2222	·	
HDM_bk_01-06_04_120	2117	·	
HDM_bk_02-01_01_120	3645	2-1 Table and floor	
HDM_bk_02-01_02_120	3714	·	
HDM_bk_02-01_03_120	3304	·	
HDM_bk_02-02_01_120	6680	2-2 Shelf (while walking)	
HDM_bk_02-02_02_120	7095	·	
HDM_bk_02-02_03_120	6449	·	
HDM_bk_02-03_01_120	2530	2-3 Shelf (while standing)	
HDM_bk_02-03_02_120	1867	·	
HDM_bk_02-03_03_120	2143	·	
HDM_bk_02-03_04_120	1915	·	
HDM_bk_03-01_01_120	9701	3-1 Dancing	
HDM_bk_03-01_02_120	8783	·	
HDM_bk_03-01_03_120	8289	·	
HDM_bk_03-02_01_120	4981	3-2 Kicking and punching	
HDM_bk_03-02_02_120	4141	·	
HDM_bk_03-02_03_120	4269	·	
HDM_bk_03-03_01_120	4412	3-3 Throwing	
HDM_bk_03-03_02_120	5018	·	
HDM_bk_03-03_03_120	4789	·	
HDM_bk_03-04_01_120	11398	3-4 Rotating arms	
HDM_bk_03-04_02_120	7422	·	
HDM_bk_03-04_03_120	7828	·	
HDM_bk_03-04_04_120	7330	·	
HDM_bk_03-05_01_120	6041	3-5 workout	
HDM_bk_03-05_02_120	4639	·	
HDM_bk_03-05_03_120	4612	·	
HDM_bk_03-08_01_120	4998	3-8 workout	
HDM_bk_03-11_01_120	3113	3-11 Badminton	
HDM_bk_03-11_02_120	3277	·	
HDM_bk_03-11_03_120	3008	·	
HDM_bk_04-01_01_120	13318	4-1 Chair, table, floor	
HDM_bk_04-01_02_120	12539	·	
HDM_bk_04-01_03_120	12649	·	
HDM_bk_05-01_01_120	5544	5-1 Clapping and waving	
HDM_bk_05-01_02_120	6154	·	
HDM_bk_05-01_03_120	5841	·	
HDM_bk_05-02_01_120	4192	·	
HDM_bk_05-02_02_120	4386	5-2 Shouting and tying shoes	
HDM_bk_05-02_03_120	4163	·	
HDM_bk_05-03_01_120	4128	5-3 Variations of locomotion	
HDM_bk_05-03_02_120	4594	·	
HDM_bk_05-03_03_120	3909	·	
HDM_dg_01-01_01_120	7542	1-1 Walking	
HDM_dg_01-01_02_120	7660	·	
HDM_dg_01-01_03_120	7687	·	
HDM_dg_01-01_04_120	7647	·	
HDM_dg_01-02_01_120	2867	1-2 Locomotion on the spot	
HDM_dg_01-02_02_120	2762	·	
HDM_dg_01-02_03_120	2786	·	
HDM_dg_01-03_01_120	3498	1-3 Locomotion	
HDM_dg_01-03_02_120	3794	·	
HDM_dg_01-03_03_120	3584	·	
HDM_dg_01-04_01_120	5509	1-4 Locomotion with weights	
HDM_dg_01-04_02_120	6394	·	
HDM_dg_01-04_03_120	6047	·	
HDM_dg_01-05_01_120	4489	1-5 Hopping and jumping	
HDM_dg_01-05_02_120	3947	·	
HDM_dg_01-05_03_120	4660	·	
HDM_dg_01-06_01_120	2348	1-6 Climbing stairs	
HDM_dg_01-06_02_120	2216	·	
HDM_dg_01-06_03_120	1909	·	
HDM_dg_02-01_01_120	3034	2-1 Table and floor	
HDM_dg_02-01_02_120	3526	·	

File Name Prefix	#(fr.)	Description	Comments
HDM_dg_02-01_03_120	3108	·	
HDM_dg_02-02_01_120	6087	2-2 Shelf (while walking)	
HDM_dg_02-02_02_120	4198	·	
HDM_dg_02-02_03_120	4750	·	
HDM_dg_02-02_04_120	5461	·	
HDM_dg_02-03_01_120	2436	2-3 Shelf (while standing)	
HDM_dg_02-03_02_120	2900	·	
HDM_dg_02-03_03_120	2225	·	
HDM_dg_03-01_01_120	8336	3-1 Dancing	
HDM_dg_03-01_02_120	7430	·	
HDM_dg_03-01_03_120	7951	·	
HDM_dg_03-02_01_120	6823	3-2 Kicking and punching	
HDM_dg_03-02_02_120	5861	·	
HDM_dg_03-02_03_120	3454	·	
HDM_dg_03-03_01_120	3219	3-3 Throwing	
HDM_dg_03-03_02_120	3312	·	
HDM_dg_03-03_03_120	3531	·	
HDM_dg_03-04_01_120	7101	3-4 Rotating arms	
HDM_dg_03-04_02_120	6307	·	ASF/AMC does not fit C3D
HDM_dg_03-04_03_120	6444	·	
HDM_dg_03-05_01_120	3839	3-5 workout	
HDM_dg_03-05_02_120	3866	·	
HDM_dg_03-05_03_120	3551	·	
HDM_dg_03-09_01_120	5314	sit-ups, push-ups, workout	flipping markers during push-ups
HDM_dg_03-09_02_120	4844	·	
HDM_dg_03-09_03_120	4052	·	
HDM_dg_03-11_01_120	2859	3-11 Badminton	
HDM_dg_03-11_02_120	3138	·	
HDM_dg_03-11_03_120	2711	·	
HDM_dg_03-11_04_120	6188	·	
HDM_dg_04-01_01_120	9088	4-1 Chair, table, floor	
HDM_dg_04-01_02_120	8316	·	Strong artifacts in ASF/AMC left leg data in frames 7224-7300
HDM_dg_05-01_01_120	5607	5-1 Clapping and waving	
HDM_dg_05-01_02_120	6865	·	
HDM_dg_05-01_03_120	6617	·	
HDM_dg_05-02_01_120	2468	5-2 Shouting and tying shoes	
HDM_dg_05-02_02_120	2063	·	
HDM_dg_05-02_03_120	1995	·	
HDM_dg_05-03_01_120	3276	5-3 Variations of locomotion	
HDM_dg_05-03_02_120	3425	·	
HDM_dg_06-01_01_120	9130	wobbling	
HDM_dg_06-02_01_120	8136	walking and jogging in circles	
HDM_dg_06-03_01_120	331	walking	
HDM_dg_06-03_02_120	285	·	
HDM_dg_06-03_03_120	202	jogging	
HDM_dg_06-03_04_120	99	running	
HDM_dg_06-04_01_120	3987	provoking	
HDM_dg_07-01_01_120	3436	inline skating in circles	
HDM_dg_07-01_02_120	279	inline skating straight	
HDM_dg_07-01_03_120	183	inline skating straight	
HDM_dg_07-01_04_120	234	inline skating backwards	
HDM_dg_07-01_05_120	309	inline skating straight	
HDM_dg_07-01_06_120	251	inline skating jumping	
HDM_dg_07-01_07_120	269	inline skating turning	
HDM_dg_07-01_08_120	285	inline skating backwards	
HDM_dg_08-01_01_120	2131	opening bottle and drinking	
HDM_mm_01-01_01_120	8361	1-1 Walking	
HDM_mm_01-01_02_120	8257	·	
HDM_mm_01-01_03_120	7487	·	
HDM_mm_01-02_01_120	3059	1-2 Locomotion on the spot	
HDM_mm_01-02_02_120	2712	·	
HDM_mm_01-02_03_120	3990	·	
HDM_mm_01-03_01_120	3928	1-3 Locomotion	

File Name Prefix	#(fr.)	Description	Comments
HDM_mm_01-03_02_120	4020	·	
HDM_mm_01-03_03_120	3674	·	
HDM_mm_01-04_01_120	6918	1-4 Locomotion with weights	
HDM_mm_01-04_02_120	6687	·	
HDM_mm_01-04_03_120	6806	·	
HDM_mm_01-05_01_120	4673	1-5 Hopping and jumping	
HDM_mm_01-06_01_120	2514	1-6 Climbing stairs	
HDM_mm_01-06_02_120	3188	·	
HDM_mm_01-06_03_120	1816	·	
HDM_mm_02-01_01_120	3673	2-1 Table and floor	
HDM_mm_02-01_02_120	2449	·	
HDM_mm_02-01_03_120	3576	·	
HDM_mm_02-01_04_120	1845	·	
HDM_mm_02-02_01_120	5810	2-2 Shelf (while walking)	
HDM_mm_02-02_02_120	8500	·	
HDM_mm_02-02_03_120	4237	·	
HDM_mm_02-03_01_120	3464	2-3 Shelf (while standing)	
HDM_mm_02-03_02_120	2205	·	
HDM_mm_02-03_03_120	2240	·	
HDM_mm_03-01_01_120	7767	3-1 Dancing	
HDM_mm_03-01_02_120	6995	·	
HDM_mm_03-01_03_120	8040	·	
HDM_mm_03-02_01_120	7671	3-2 Kicking and punching	
HDM_mm_03-02_02_120	4850	·	
HDM_mm_03-02_03_120	3313	·	
HDM_mm_03-02_04_120	8340	·	
HDM_mm_03-03_01_120	3900	3-3 Throwing	
HDM_mm_03-03_02_120	4299	·	
HDM_mm_03-04_01_120	10361	3-4 Rotating arms	
HDM_mm_03-04_02_120	7244	·	
HDM_mm_03-04_03_120	4543	·	
HDM_mm_03-05_01_120	1994	3-5 workout	The take doesn't start with a T-pose. The jumping Jacks are missing, only 3 skiers are contained.
HDM_mm_03-05_02_120	4290	·	
HDM_mm_03-05_03_120	3485	·	In the reslization of the jumping Jacks the hands move front-to-back instead of side-to-top. In the reslization of the skiing exercise the hands move side-to-top instead of front-to-back. The elbow-to-knee exercise and the squats are missing.
HDM_mm_03-10_01_120	5553	rope skipping (two-legged)	
HDM_mm_03-10_02_120	5270	rope skipping (one-legged)	
HDM_mm_03-10_03_120	6226	rope skipping (alternating legs)	
HDM_mm_03-10_04_120	2476	rope skipping (two turns per jump)	
HDM_mm_03-10_05_120	2468	rope skipping (very fast)	
HDM_mm_03-10_06_120	979	rope skipping (2 people turning rope)	
HDM_mm_03-10_07_120	1679	rope skipping (2 people turning rope)	
HDM_mm_04-01_01_120	12848	4-1 Chair, table, floor	
HDM_mm_04-01_02_120	7535	·	
HDM_mm_05-01_01_120	6361	5-1 Clapping and waving	
HDM_mm_05-01_02_120	6953	·	
HDM_mm_05-01_03_120	6218	·	
HDM_mm_05-02_01_120	2428	5-2 Shouting and tying shoes	
HDM_mm_05-02_02_120	2103	·	
HDM_mm_05-02_03_120	2556	·	
HDM_mm_05-03_01_120	4170	5-3 Variations of locomotion	
HDM_mm_05-03_02_120	4167	·	
HDM_mm_05-03_03_120	4328	·	
HDM_mm_06-04_01_120	5795	provoking	
HDM_mm_08-01_01_120	2450	opening bottle and drinking	
HDM_tr_01-01_01_120	8893	1-1 Walking	
HDM_tr_01-01_02_120	7967	·	
HDM_tr_01-01_03_120	7889	·	
HDM_tr_01-02_01_120	2952	1-2 Locomotion on the spot	
HDM_tr_01-02_02_120	2922	·	
HDM_tr_01-02_03_120	2972	·	
HDM_tr_01-03_01_120	3611	1-3 Locomotion	

File Name Prefix	#(fr.)	Description	Comments
HDM_tr_01-03_02_120	3959	·	
HDM_tr_01-03_03_120	3690	·	
HDM_tr_01-03_04_120	3735	·	
HDM_tr_01-04_01_120	6629	1-4 Locomotion with weights	
HDM_tr_01-05_01_120	4792	1-5 Hopping and jumping	
HDM_tr_01-05_02_120	4638	·	
HDM_tr_01-05_03_120	4548	·	
HDM_tr_01-06_01_120	1704	1-6 Climbing stairs	
HDM_tr_01-06_02_120	1351	·	
HDM_tr_01-06_03_120	2063	·	
HDM_tr_02-01_01_120	3402	2-1 Table and floor	
HDM_tr_02-01_02_120	3321	·	
HDM_tr_02-01_03_120	3307	·	
HDM_tr_02-02_01_120	5922	2-2 Shelf (while walking)	
HDM_tr_02-02_02_120	5883	·	
HDM_tr_02-02_03_120	5088	·	
HDM_tr_02-02_04_120	5573	·	
HDM_tr_02-03_01_120	2402	2-3 Shelf (while standing	
HDM_tr_02-03_02_120	2746	·	
HDM_tr_02-03_03_120	1720	·	
HDM_tr_03-01_01_120	9459	3-1 Dancing	
HDM_tr_03-01_02_120	8250	·	
HDM_tr_03-01_03_120	7516	·	
HDM_tr_03-01_04_120	8590	·	
HDM_tr_03-02_01_120	4761	3-2 Kicking and punching	
HDM_tr_03-02_02_120	4572	·	
HDM_tr_03-02_03_120	4452	·	
HDM_tr_03-02_04_120	4255	·	
HDM_tr_03-03_01_120	3721	3-3 Throwing	
HDM_tr_03-03_02_120	3723	·	
HDM_tr_03-03_03_120	3323	·	
HDM_tr_03-04_01_120	6239	3-4 Rotating arms	
HDM_tr_03-04_02_120	6988	·	
HDM_tr_03-04_03_120	7553	·	
HDM_tr_03-05_01_120	4999	3-5 workout	tr starts the skier with right hand and right foot in front instead of right hand and left foot in front.
HDM_tr_03-05_02_120	4500	·	tr starts the skier with right hand and right foot in front instead of right hand and left foot in front.
HDM_tr_03-05_03_120	4357	·	tr starts the skier with right hand and right foot in front instead of right hand and left foot in front.
HDM_tr_03-10_01_120	2855	rope jumping while spinning	
HDM_tr_03-10_02_120	2479	·	
HDM_tr_03-10_03_120	1578	·	
HDM_tr_03-11_01_120	3635	3-11 Badminton	
HDM_tr_03-11_02_120	2581	·	
HDM_tr_03-11_03_120	2209	·	
HDM_tr_04-01_01_120	11201	4-1 Chair, table, floor	
HDM_tr_05-01_01_120	5842	5-1 Clapping and waving	
HDM_tr_05-01_02_120	4942	·	
HDM_tr_05-01_03_120	4316	·	This file belongs to the scene 03-02 and not to the scene 05-01.
HDM_tr_05-02_01_120	1691	5-2 Shouting and tying shoes	
HDM_tr_05-02_02_120	1726	·	
HDM_tr_05-02_03_120	1705	·	
HDM_tr_05-03_01_120	3224	5-3 Variations of locomotion	
HDM_tr_05-03_02_120	3634	·	
HDM_tr_05-03_03_120	3338	·	
HDM_tr_05-03_04_120	6453	·	
HDM_tr_06-01_01_120	10767	various volleyball motions	
HDM_tr_06-01_02_120	16136	various weight lifting motions	

Anhang B

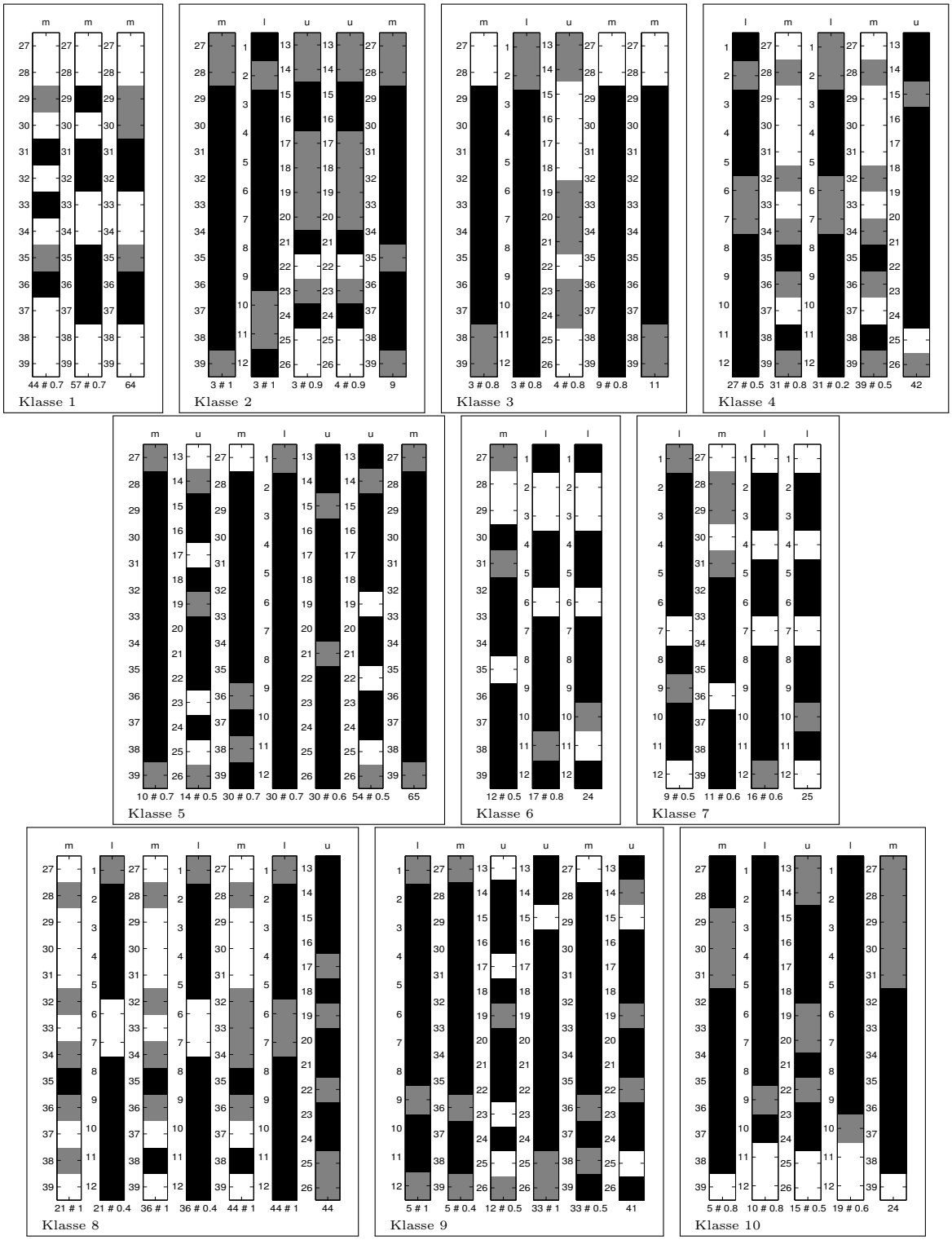
Bei Experimenten verwendete Keyframes

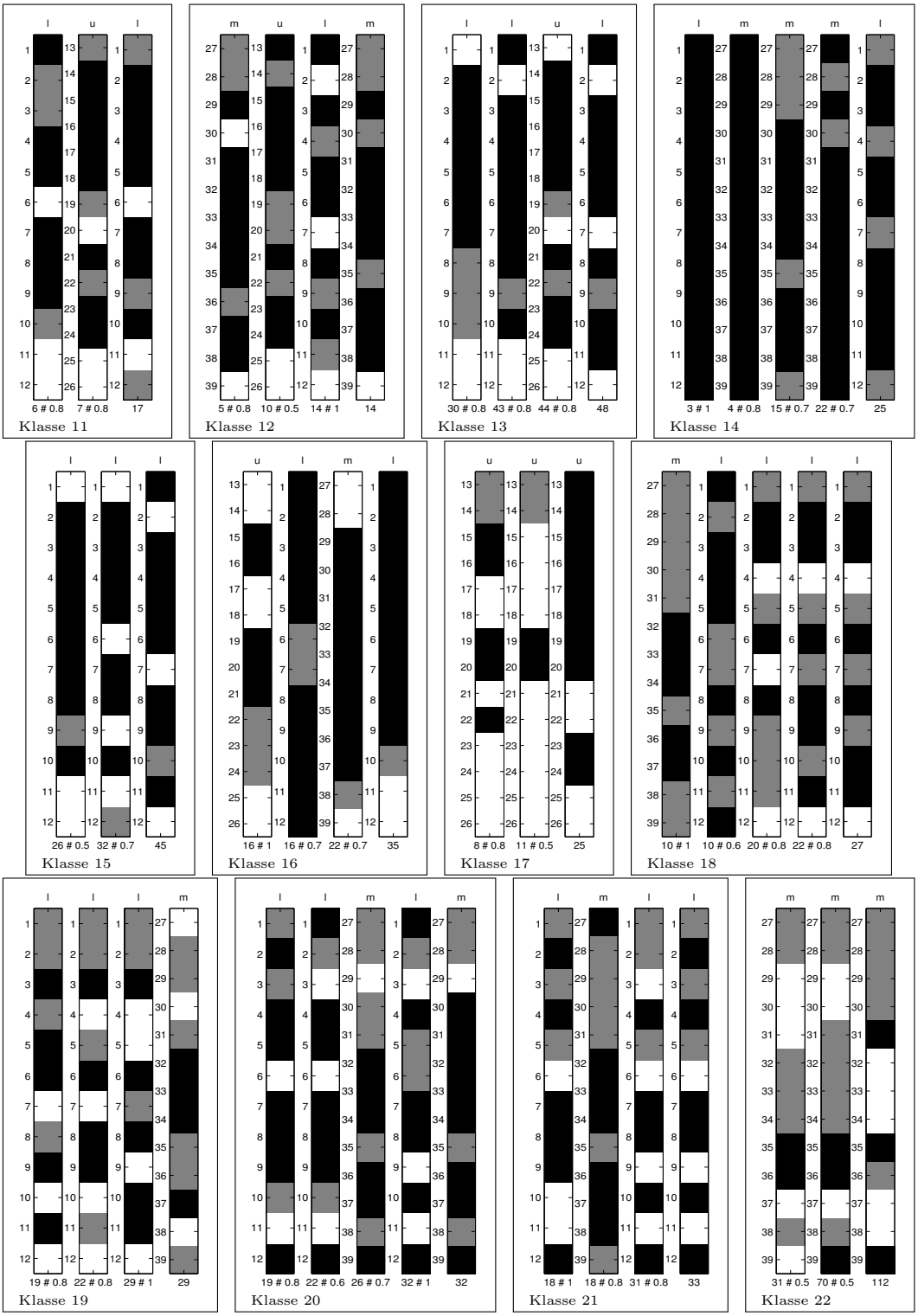
In diesem Anhang werden die in den Retrieval-Experimenten verwendeten Keyframes aufgelistet. In Abschnitt 5.2 wurde beschrieben, wie diese Keyframes gewonnen wurden, in Abschnitt 5.3 wird die Qualität der Keyframes beurteilt. Im Folgenden wird für jede Klasse der $\mathcal{D}^{57}$ eine Abbildung gezeichnet, die die Keyframes dieser Klasse beschreibt. Die Nummerierung der Klassen ist konsistent mit der Datenbank $\mathcal{D}^{57}$ aus Anhang A. In jeder Abbildung entspricht eine Spalte einem Keyframe. Die Spaltenüberschrift enthält einen Hinweis, zu welcher Featuremenge dieser Keyframe gehört: „ℓ“ für „lower“, „u“ für „upper“, „m“ für „mix“. Unter jedem Keyframe sind zwei Werte aufgetragen. Zuerst wird die Position des Keyframes innerhalb der Bewegungssequenz genannt. Die Position ist als Frameangabe im Bezug auf eine 30 Hz-Abtastrate gegeben. Für den ersten Keyframe der ersten Klasse bedeutet also die Frameangabe „44“, dass dieser Keyframe $44/30 \approx 1.47s$ nach dem Beginn der Bewegung auftritt. Der zweite Keyframe der ersten Klasse tritt $57/30 = 1.9s$ nach dem Beginn der Bewegung auf. An zweiter Position steht der Stiffness-Wert, der für den Abstand zum nächsten Keyframe gilt. Bei den meisten Bewegungen variiert der Stiffness-Wert mit den Keyframes. Die Beschriftung der Y-Achse gibt die Nummer des Features im Bezug auf die Auflistung der verwendeten Features in Tabelle 5.1, Seite 65, an.

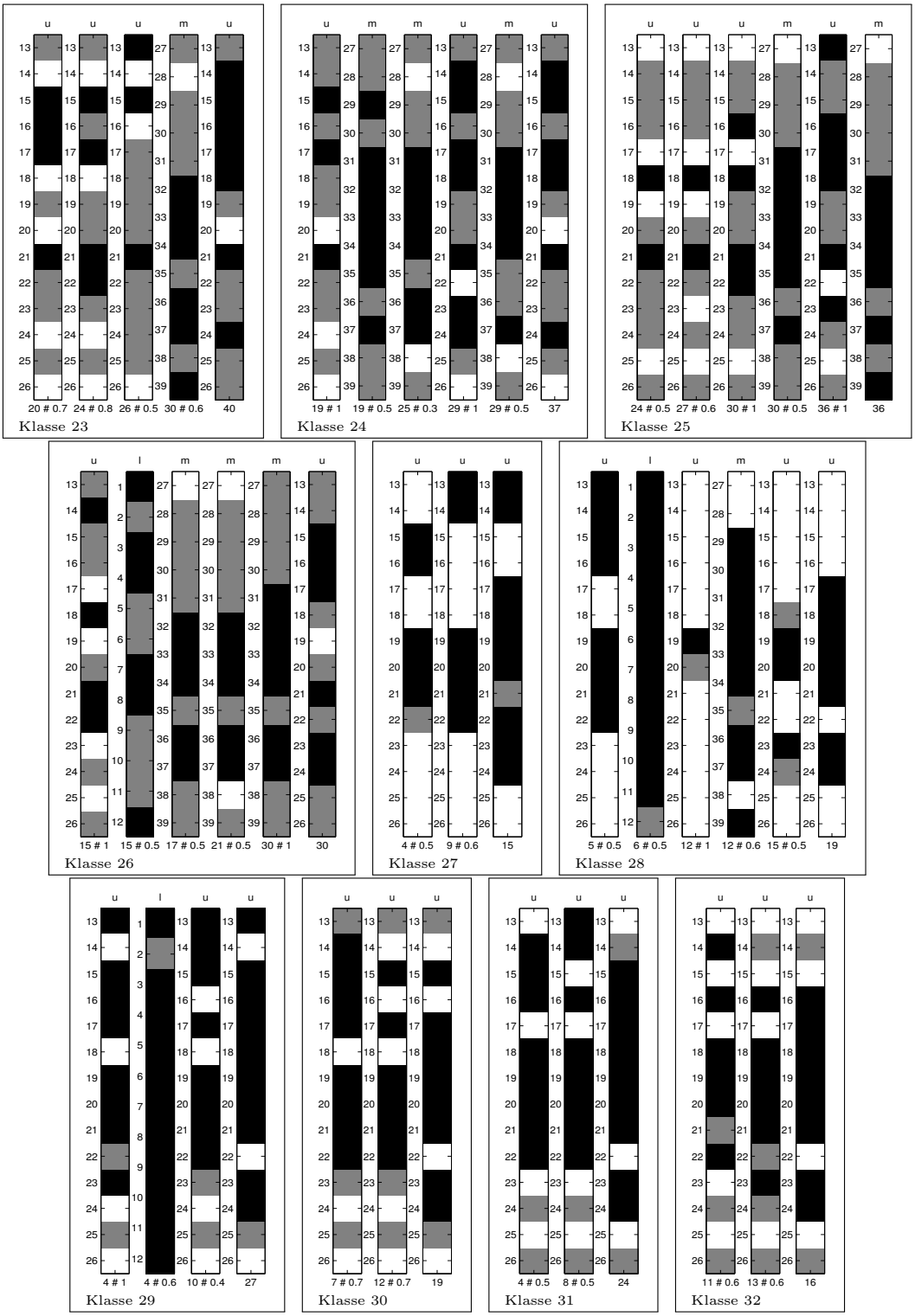
Der Featurewert 1 wird mit der einem weißen Block codiert, der Wert 0 mit einem schwarzen Block. 0.5-Werte werden als graue Blöcke dargestellt. Stellt man zum Beispiel den ersten Keyframe der ersten Klasse als Vektor dar, so ist dieser gegeben durch

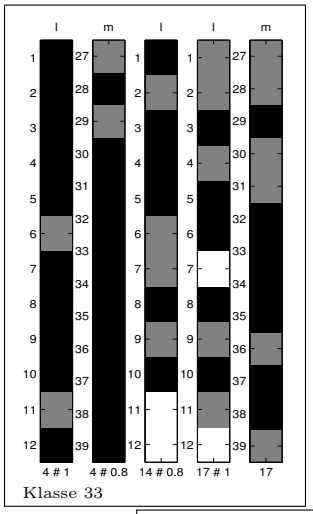
$$V = (1, 1, 0.5, 1, 0, 1, 0, 1, 0.5, 0, 1, 1, 1).$$

Der für den Algorithmus 4.1 benötigte Abstand zwischen den Keyframes berechnet sich durch die Differenz der Abstände zweier benachbarter Keyframes. Die Keyframes müssen dabei so angegeben werden, dass die Abstände immer ≥ 0 sind. Ist zwischen den Keyframes V_1 und V_2 der Abstand $\delta_1 = 0$, so bedeutet dies, dass die Keyframes – unabhängig von der Stiffness – gleichzeitig auftreten müssen.

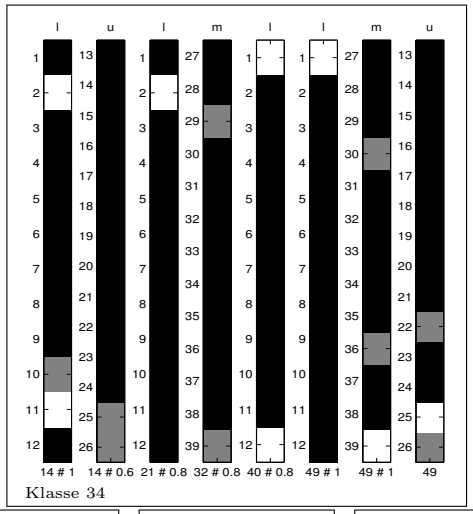

m m m
44 # 0.7 57 # 0.7 64
Klasse 1
m l u u m
3 # 1 3 # 1 3 # 0.9 4 # 0.9 9
Klasse 2
m l u m m
3 # 0.8 3 # 0.8 4 # 0.8 9 # 0.8 11
Klasse 3
l m l m u
27 # 0.5 31 # 0.8 31 # 0.2 39 # 0.5 42
Klasse 4
m u m l u u m
10 # 0.7 14 # 0.5 30 # 0.7 30 # 0.7 30 # 0.6 54 # 0.5 65
Klasse 5
m l l
12 # 0.5 17 # 0.8 24
Klasse 6
l m l l
9 # 0.5 11 # 0.6 16 # 0.6 25
Klasse 7
m l m l m l u
21 # 1 21 # 0.4 36 # 1 36 # 0.4 44 # 1 44 # 1 44
Klasse 8
l m u u m u
5 # 1 5 # 0.4 12 # 0.5 33 # 1 33 # 0.5 41
Klasse 9
m l u l m
5 # 0.8 10 # 0.8 15 # 0.5 19 # 0.6 24
Klasse 10


l u l
6 # 0.8 7 # 0.8 17
Klasse 11
m u l m
5 # 0.8 10 # 0.5 14 # 1 14
Klasse 12
l l u l
30 # 0.8 43 # 0.8 44 # 0.8 48
Klasse 13
l m m m l
3 # 1 4 # 0.8 15 # 0.7 22 # 0.7 25
Klasse 14
l l l
26 # 0.5 32 # 0.7 45
Klasse 15
u l m l
16 # 1 16 # 0.7 22 # 0.7 35
Klasse 16
u u u
8 # 0.8 11 # 0.5 25
Klasse 17
m l l l l
10 # 1 10 # 0.6 20 # 0.8 22 # 0.8 27
Klasse 18
l l l m
19 # 0.8 22 # 0.8 29 # 1 29
Klasse 19
l l m l m
19 # 0.8 22 # 0.6 26 # 0.7 32 # 1 32
Klasse 20
l m l l
18 # 1 18 # 0.8 31 # 0.8 33
Klasse 21
m m m
31 # 0.5 70 # 0.5 112
Klasse 22

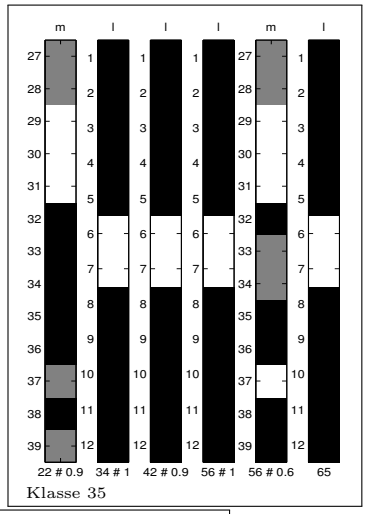

u u u m u
20 # 0.7 24 # 0.8 26 # 0.5 30 # 0.6 40
Klasse 23
u m m u m u
19 # 1 19 # 0.5 25 # 0.3 29 # 1 29 # 0.5 37
Klasse 24
u u u m u m
24 # 0.5 27 # 0.6 30 # 1 30 # 0.5 36 # 1 36
Klasse 25
u l m m m u
15 # 1 15 # 0.5 17 # 0.5 21 # 0.5 30 # 1 30
Klasse 26
u u u
4 # 0.5 9 # 0.6 15
Klasse 27
u l u m u u
5 # 0.5 6 # 0.5 12 # 1 12 # 0.6 15 # 0.5 19
Klasse 28
u l u u
4 # 1 4 # 0.6 10 # 0.4 27
Klasse 29
u u u
7 # 0.7 12 # 0.7 19
Klasse 30
u u u
4 # 0.5 8 # 0.5 24
Klasse 31
u u u
11 # 0.6 13 # 0.6 16
Klasse 32



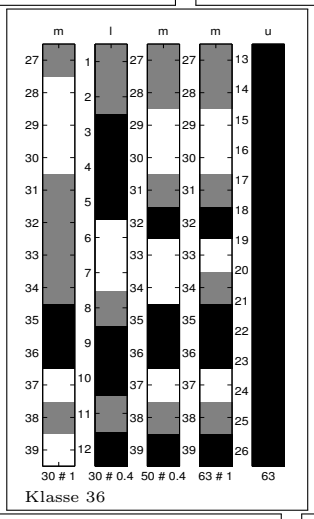
Klasse 33



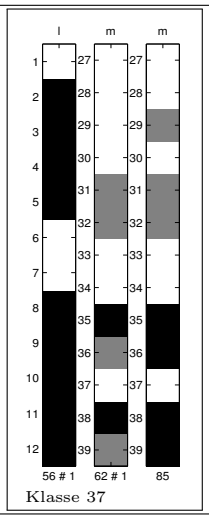
Klasse 34



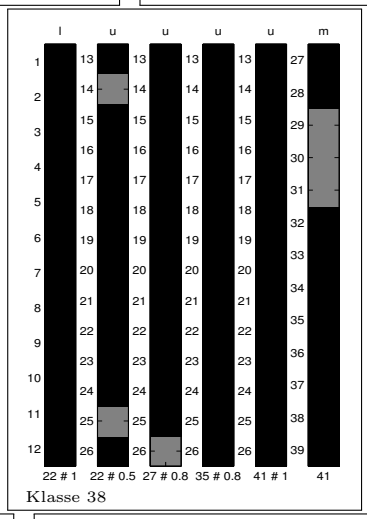
Klasse 35



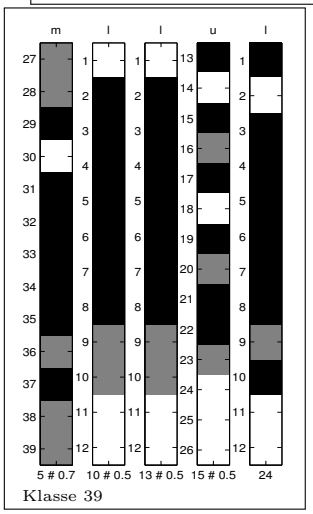
Klasse 36



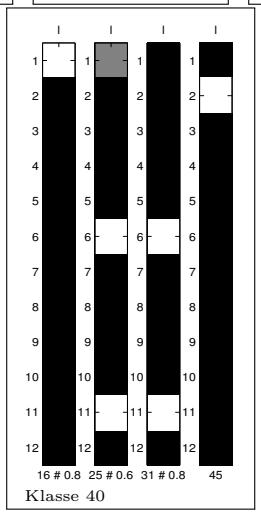
Klasse 37



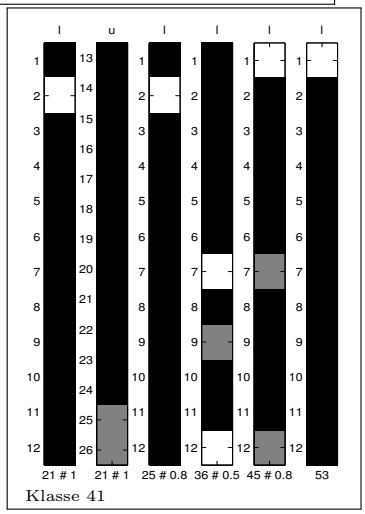
Klasse 38



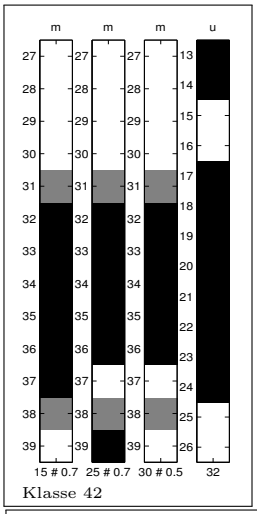
Klasse 39



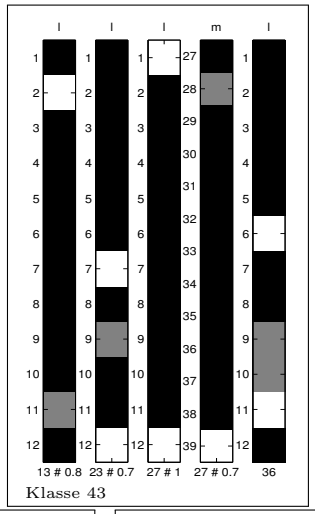
Klasse 40



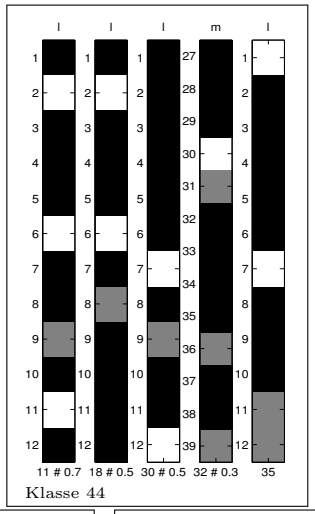
Klasse 41



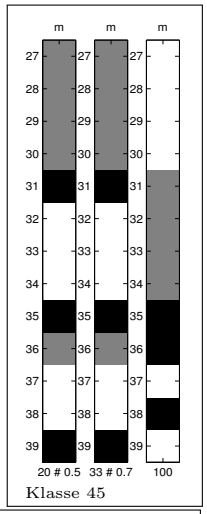
Klasse 42



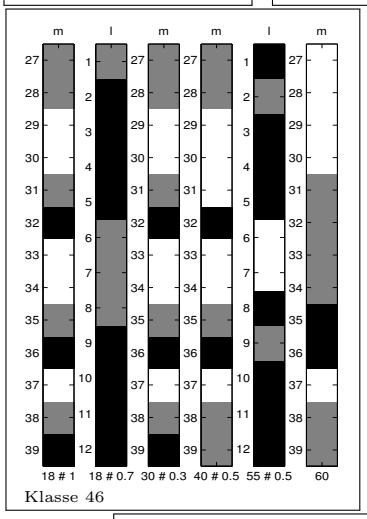
Klasse 43



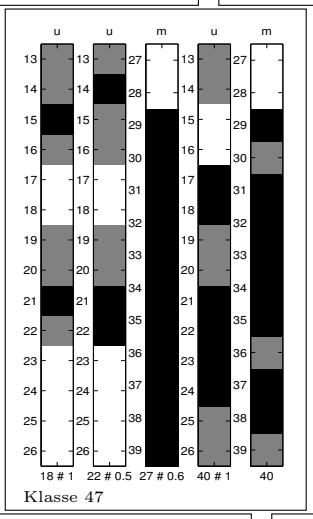
Klasse 44



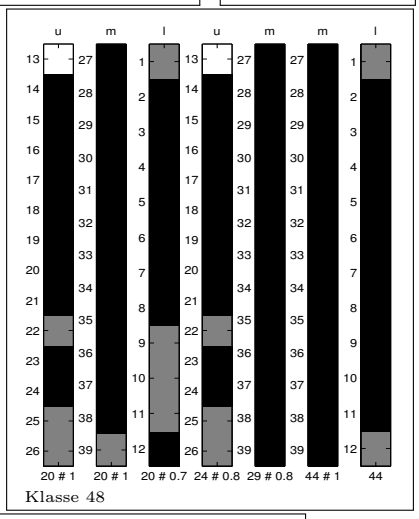
Klasse 45



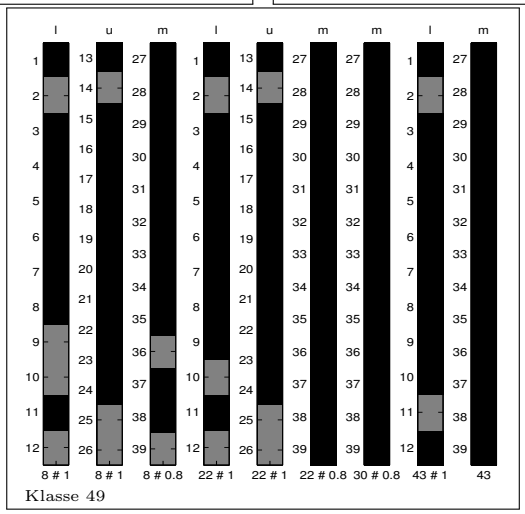
Klasse 46



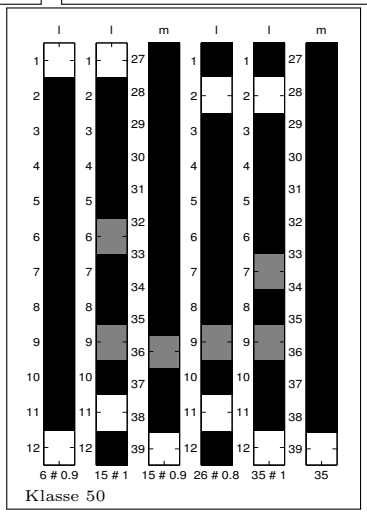
Klasse 47



Klasse 48



Klasse 49



Klasse 50

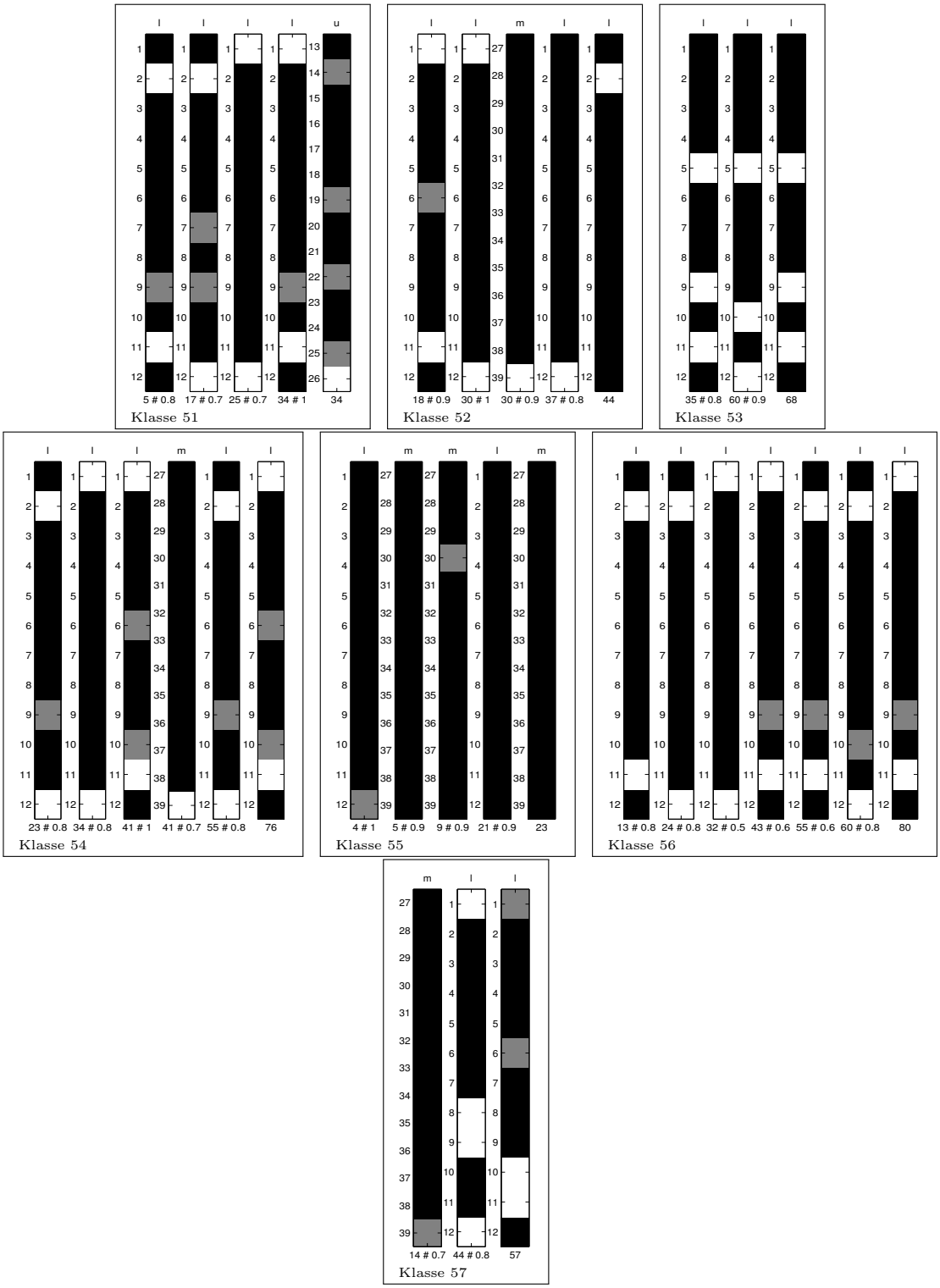


Abbildung B.1. Die bei den Experimenten verwendeten Keyframes sind hier abgebildet.

Erklärung der selbständigen Arbeit

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Bonn, den

Andreas Baak

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Literaturverzeichnis

[ASF99] ASF/AMC. Motion Capture Data Format. University of Wisconsin, `http://www.cs.wisc.edu/graphics/Courses/cs-838-1999/Jeff/ASF-AMC.html`, 1999.

[C3D06] C3D.org. C3D mocap format. `http://www.c3d.org`, 2006.

[CM07] Michael Clausen and Meinard Müller. Inhaltsbasiertes Multimediaretrieval, 2007.

[DRME06] Bastian Demuth, Tido Röder, Meinard Müller, and Bernhard Eberhardt. An information retrieval system for motion capture data. In *Proc. 28th European Conference on Information Retrieval (ECIR 2006)*, volume 3936 of *LNCS*, pages 373–384. Springer, 2006.

[Ewe07] Hendrik Ewe. Effizientes Retrieval auf markerbasierten Bewegungsdaten. Master's thesis, University of Bonn, 2007.

[MR06] Meinard Müller and Tido Röder. Motion templates for automatic classification and retrieval of motion capture data. In *Proc. 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA 2006)*. Eurographics Association, 2006.

[MRC05] Meinard Müller, Tido Röder, and Michael Clausen. Efficient content-based retrieval of motion capture data. *ACM Trans. Graph.*, 24(3):677–685, 2005.

[MRC+07] Meinard Müller, Tido Röder, Michael Clausen, Bernhard Eberhardt, Björn Krüger, and Andreas Weber. Documentation mocap database hdm05. *Technical report, No. CG-2007-2*, June 2007.

[Mül07] Meinard Müller. *Information Retrieval for Music and Motion*. Monograph, Springer, 2007.

[Röd06] Tido Röder. *Similarity, Retrieval, and Classification of Motion Capture Data*. PhD thesis, University of Bonn, 2006.