Aktionsklassifikation in VR

Projektgruppe WS 2016/2017

Dominik Blitsch, Leon Hüber, Can Tosun, Mohammad Vosoughi

19. März 2017

1 Einführung

Ein wesentlicher Bestandteil künstlicher Intelligenz ist die Erkennung und die daraus geschlussfolgerte Voraussage von menschlichem Verhalten. Unsere Arbeit ist der Versuch in Virtual Reality (VR) eine Live-Klassifizierung von menschlichem Verhalten zu ermöglichen.

2 Verwandte Arbeiten

Voraussagen über menschliches Verhalten war schon früher Gegenstand der Forschung. Bereits 1977 gab es Forschungen über den Angebotscharakter (Affordances) von Gegenständen [J77]. Dabei ging es darum, den Gebrauchscharakter von Objekten für Subjekte wie den Menschen zu erkennen. Zum Beispiel kann ein Mensch auf einem Stuhl sitzen oder sich auf ihn stellen, um höher gelegene Gegenstände zu erreichen. Für Elefanten gilt dies im Allgemeinen nicht. Diese Erkenntnisse konnten im Bereich der Benutzerschnittstellen verwendet werden, um technische Gegenstände besser zu charakterisieren. Beispielsweise impliziert ein Schatten um einen Knopf oftmals, dass man ihn drücken kann.
Darauf aufbauend gab es Versuche, diese Ergebnisse im Bereich der Robotik zu nutzen. Das Ziel war es, dass autonome Roboter menschliche Interaktionen wahrnehmen und daraus lernen sollten, menschliches Verhalten vorherzusehen [FPB+06, MLBSV08, SLZ+08, SMBR10].
Der Mensch tendiert dazu bestimmte Aktionen eher in bestimmten Bereichen zu vollziehen. Forscher der Universität Stanford haben daher die Interaktion des Menschen in den Kontext seiner Umgebung gestellt, um so die Vorhersagen zu verbessern. Dazu wurden sogenannte Action-Maps von 3D-Umgebungen erstellt, mit denen Aktionen und bestimmte Regionen verknüpft werden können. Beispielsweise ist die Wahrscheinlichkeit der Aktion „Sitzen“ in der Nähe von Stühlen höher als anderswo im Raum. Um dies zu erreichen, haben die Forscher mithilfe von RGB-D Sensoren die Umgebung und die Interaktionen aufgenommen und virtualisiert. Darauffolgend wurden Interaktionen aufgenommen und benannt. Unter Verwendung von Machine Learning Algorithmen wurden dann die Interaktionen erlernt. Auf der Grundlage des erlernten Modells konnten anschließend neue

Observationen live klassifiziert werden.
Im Gegensatz zu der Forschung aus Stanford ist das Augenmerk bei unserer Arbeit nicht auf die Interaktionen in realen Umgebungen gesetzt, sondern auf die Interaktion in virtuellen 3D-Umgebungen, beziehungsweise auf Virtual Reality (VR). In VR Anwendungen werden die Bewegungen des Benutzers in die virtuelle Welt übertragen.

3 Das Framework

Für die Klassifikation arbeiten wir im Gegensatz zu [SCH+14] nicht in realen Umgebungen, sondern verwenden virtuellen Szenen, in denen wir uns mit einer HTC Vive [Viv17] bewegen und den Szenenobjekten interagieren können. Dadurch können wir nicht nur einfacher verschiedene Umgebungen simulieren, sondern haben so auch die Möglichkeit mehr Informationen zu den Szenen und deren Objekten zu erhalten. Für die Umsetzung haben wir ein Framework erstellt, das auf der Unreal Engine 4.14 [EG17] aufbaut. Die Unreal Engine stellt die grundsätzlichen Funktionen, wie das Erstellen von Szenen und die Anbindung an die HTC Vive zur Verfügung. Das Framework erweitert die Funktionalität noch um das Aufnehmen und Abspielen von Aktionen, das Erstellen der Merkmalsvektoren aus diesen Aktionen (siehe Abschnitt 4.5) und der direkten Klassifizierung von neuen Aktionen.

3.1 HTC Vive

Für die Interaktion mit den virtuellen Szenen verwenden wir das VR-System HTC Vive. Zu dessen Umfang gehört ein Head-Mounted Display (HMD), zwei Controller und zwei Infrarotsensoren. Diese sogenannten Basisstationen werden am Rand des zuvor festgelegten Feldes auf gegenüberliegenden Seiten aufgestellt und berechnen die Positionen und Orientierungen des HMDs (dem Kopf) und der Controller (den Händen). In diesem Feld kann sich der Benutzer frei bewegen und die Szenen durchlaufen. Um größere Distanzen zu überbrücken, haben wir die Szenen so angepasst, dass man sich zu jeder Stelle teleportieren kann. Die HTC Vive bietet mit dem Tracking von Kopf und Händen und der freien Bewegung im Raum schon eine gute Flexibilität, allerdings sind von den Posen des Nutzers so auch ohne Weiteres nur diese Positionen bekannt. Das Erkennen aller Gelenke wie in [SCH+14] ist so nicht möglich (siehe Abschnitt 5).

3.2 Szenen

Jede Szene wird als eigenständiges Level erstellt und im Order „Level“ gespeichert. Alle Level aus diesem Ordner werden später in der Anwendung aufgelistet und können dort vom Nutzer geladen werden. Dabei wird das ausgewählte Level in ein Basislevel „gestreamt“, das heißt, es werden alle Funktionalitäten und Szeneninhalte aus beiden Leveln zusammen verwendet und dargestellt. Das hat in unserem Fall den Vorteil, dass die von uns implementierten Funktionen, wie die Aktionsaufnahme oder das Teleportieren in der Szene nur im Basislevel verankert sein müssen, aber trotzdem in jeder Szene genutzt werden können. Gerade das Hinzufügen weiterer Szenen wird so erleichtert.

3.3 Kalibrierung

Damit die ausgeführten Aktionen unabhängig von der Statur des Benutzers bleiben, wird vor der Aufnahme und Klassifizierung dessen Größe und Armlänge ermittelt. Dafür wird bei aufrechtem Stand und ausgestreckten Armen die Höhe des HMDs und dessen Abstand zu den Controllern gemessen. Diese Informationen werden mit jeder Aufnahme gespeichert und der Merkmalsberechnung zur Verfügung gestellt.

3.4 Aktionsaufnahme und Wiedergabe

Der Benutzer startet die Aufnahme in einer Szene seiner Wahl und führt die gewünschte Aktion aus. Währendessen wird in regelmäßigen Abständen die aktuelle Position und Orientierung des HMDs, der Controller und jedes Szenenelements mitgeschrieben. Diese Informationen werden als Jsondokument gespeichert und lassen sich so auch gut für andere Zwecke weiterverwenden. Um gleiche Aktionen später zu gruppieren, müssen deren Aufnahmen gleich benannt werden. Sie dürfen sich im Namen daher nur um Zahlen unterscheiden.

Bei der Wiedergabe werden das HMD und die Controller durch Modelle von Kopf und Händen dargestellt. Sie besitzen Hüllkörper, mit denen die Gegenstände in der Nähe der Hände bzw. des Körpers ermittelt werden. Sie repräsentieren somit wie in [SCH+14] die *aktiven Objekte*. Wir verwenden hierzu ebenfalls Kugeln für die Hände und eine Halbkugel, die vom Kopf senkrecht nach unten verläuft für den Bereich vor dem Körper (siehe Abbildung 1. Hier ist die Tasse in der rechten Hand ein aktives Objekt). Die Radien sind über die Einstellungen anpassbar. Zudem können bei der Wiedergabe zugleich die Merkmalsvektoren mit berechnet werden.

3.5 Merkmalsvektoren erstellen

Welche Merkmale verwendet werden sollen, wird im Quelltext angegeben. Die Unreal Engine bietet mit den sogenannten Blueprints eine graphenbasierte Alternative zur C++-Programmierung an. Die von uns erstellte Blueprintfunktion „logFeatures“ hat als Inputparameter die Kalibrierungsdaten, alle Informationen zu Kopf und Händen und eine Liste mit den Featurevektoren, an den jeder neu berechnete Vektor angehangen wird. Ein kurzes Beispiel ist in Abbildung 2 zu sehen. Dort werden als Merkmale die Höhe des Kopfes und dessen Abstände zu den beiden Händen berechnet. Dafür wird zunächst die Raumposition („GetWorldLocation“) der Hände und des Kopfs bzw. deren Hauptkomponenten („Default Scene Root“) ausgelesen. Das erste Merkmal ist die Z-Komponente, also die Höhe des HMDs über dem Boden, dividiert durch die zuvor in der Kalibrierung gemessene Höhe des Nutzers. Die Merkmale zwei und drei ergeben sich aus dem euklidischen Abstand zwischen dem Kopf und den Händen. Auch hier wird zur Normierung durch die zuvor gemessene Armlänge geteilt. Die Merkmale werden zusammengefasst und an die Liste mit den vorherigen Merkmalsvektoren angehangen. Die von uns verwendeten Merkmale werden im Abschnitt 4.5 genauer beschrieben.

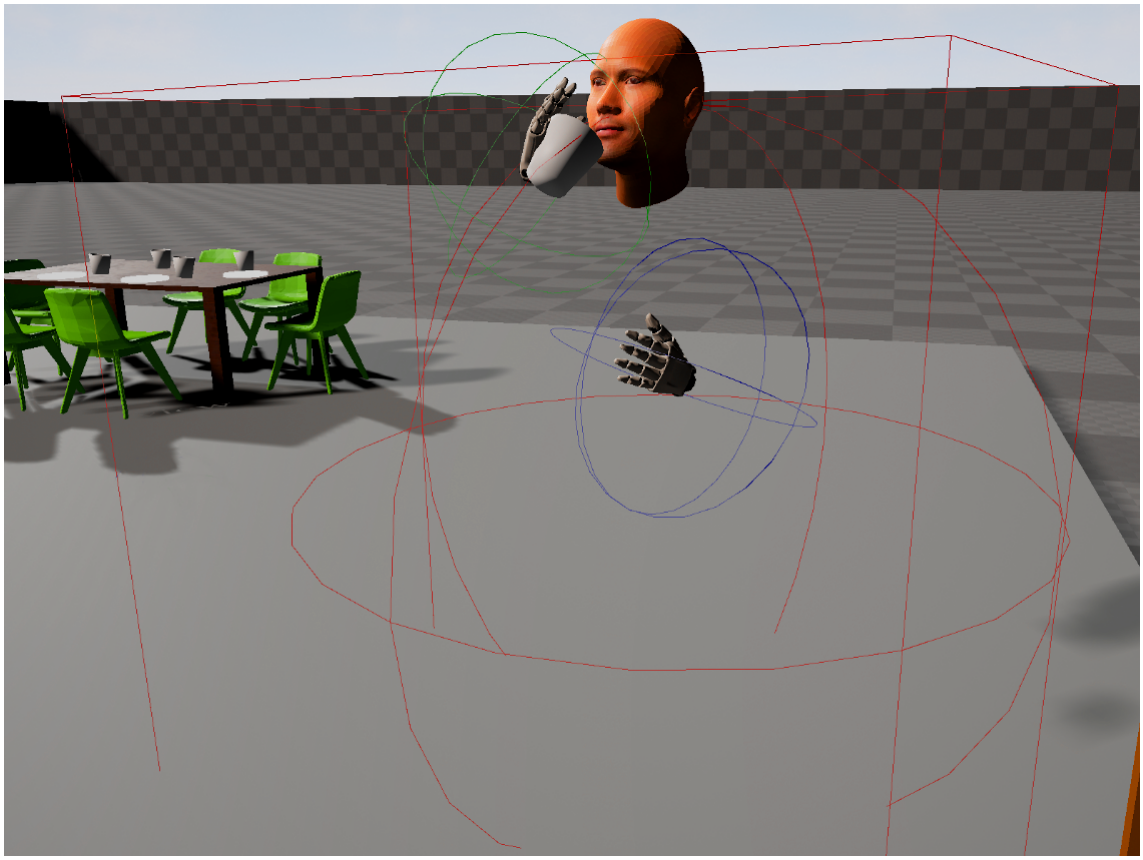

Abbildung 1: Beispiel einer Wiedergabe mit den drei Hüllkörpern. Die Kugeln um die Hände sind hier blau und grün dargestellt. Die Halbkugel für den Körper wird aus dem Schnitt des roten Quaders und der roten Kugel gebildet, da die Unreal Engine keine Halbkugeln als Hüllkörper anbietet.

3.6 Trainings- und Testdaten erstellen

Nachdem die Featurevektoren herausgeschrieben wurden, kann das Trainingsmodell erstellt werden. Hierfür verwenden wir die Support-Vector-Machine libsvm (siehe Abschnitt 4.4). Um das Modell zu generieren, benötigt die SVM Trainings- und Testdaten, die wir aus den aufgenommenen Merkmalsvektoren erstellen. Diese liegen in dem Unterordner „Features“ des Aufnahmeverzeichnisses. Zu jeder Aufnahme existiert eine entsprechende Datei mit den Featurevektoren. Da die libsvm nur Zahlen als Label der Aktionen verwendet, befindet sich dort zudem eine Datei, die jeder Gruppe von Aktionen einen Index zuordnet. Mit einem Skript können die Hälfte der Aufnahmen jeder Gruppe in einer Testdatei und die andere Hälfte in einer Trainingsdatei zusammengefügt werden. Aus den Trainingsdaten erstellt die libsvm im Anschluss ein Trainingsmodell, auf dessen Grundlage die Klassifizierung der Test- bzw. Livedaten erfolgt.

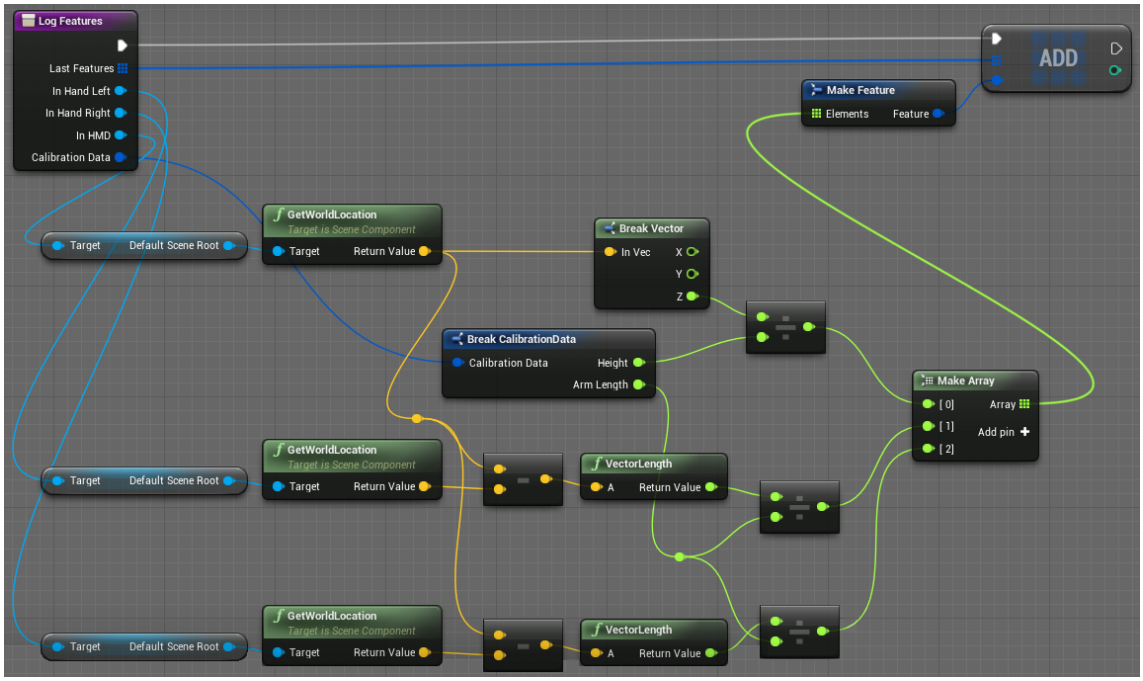

Abbildung 2: Merkmalsvektoren im Blueprint erstellen

3.7 Live Klassifizierung

Bei der Live Klassifizierung werden vom Benutzer Aktionen ausgeführt und daraus in Echtzeit die Featurevektoren bestimmt. Zu diesem Zweck haben wir die libsvm über C++ an das Framework angebunden ist. Zu Beginn wird das zuvor generierte Klassifizierungsmodell von der SVM geladen. Während der Live Klassifizierung werden dann, wie zuvor beim Erstellen der Trainingsdaten, in regelmäßigen Abständen die Merkmalsvektoren für den aktuellen Zeitpunkt berechnet. Die SVM wertet jeden Featurevektor auf der Grundlage des trainierten Modells aus und prognostiziert die Art der dargestellten Aktion. Wir stabilisieren das Ergebnis im Anschluss noch, indem immer die Prognose ausgeben wird, die unter den letzten 10 Klassifizierungen am häufigsten aufgetreten ist.

4 Support Vector Machine SVM

4.1 Einführung

Eine Support Vector Machine führt eine Klassifikation von Objekten anhand von bereits klassifizierten Objekten aus. Die Objekte werden als n-dimensionale Vektoren v im Objektraum $X \subseteq \mathbb{R}^n$ dargestellt, bei denen je ein Wert ein bestimmtes Merkmal (hier Feature) des Objektes darstellt. Zur Klassifizierung wird der Objektraum durch Hyperebenen so geteilt, dass jede Klasse von den anderen getrennt ist. Das heißt es erfolgt eine Zuordnung des Objektraumes X zum Ergebnisraum $Y = 1, ..., m \subseteq \mathbb{R}$, der

alle Klassen beinhaltet. Zur Berechnung dieser Hyperebenen dienen die Trainingsdaten $T = ((x_1, y_1), ..., (x_l, y_l)) \subseteq (X \times Y)$, die der SVM übergeben werden. Somit besteht jedes Trainingsset aus einem Objekt $x_i \in X$ und der dazugehörigen Klasse $y_i \in Y$. Die Hyperebenen werden so gewählt, dass der mögliche Unterschied innerhalb einer Klasse möglichst groß ist. Dies wird dadurch realisiert, dass die Abstände der Vektoren, die der Hyperebene am Nächsten liegen maximiert werden (Maximum Margin). Diese der Hyperebene am nächsten liegenden Vektoren werden Support-Vectors genannt, da durch diese die optimale Hyperebene „stabilisiert“ wird. Jedoch lassen sich nicht alle Datenpunkte durch Hyperebenen trennen, da es Ausreißer geben kann und Klassen sich teilweise sogar überschneiden. Um diese Daten zu klassifizieren, bedient sich die SVM des Kerneltricks. [Lä06]

4.2 Lineare Klassifikation

Die einfachste Art der linearen Klassifikation ist die binäre Klassifikation. Hierbei werden die Vektoren mit Hilfe der Funktion $f : X \subseteq \mathbb{R} \to \mathbb{R}$ der positiven ($f(x) > 0$) oder negativen ($f(x) < 0$) Klasse zugeordnet. Die trennende Hyperebene lässt sich durch einen Normalenvektor $\vec{\omega} \in X$ und ein Offset b beschreiben. Damit gilt:

$$f(\vec{x}) = \langle \vec{\omega} \cdot \vec{x} \rangle \ + b = \sum_{i=1}^{n} \omega x + b$$

Mit der Entscheidungsfunktion $h : \mathbb{R} \to \{-1, 1\}$:

$$h(x) = sign(f(x)) = sign(\sum_{i=1}^{n} \omega x + b)$$

Eingabevektoren, die sich auf diese Weise trennen lassen, werden linear separabel genannt. Ein Beispiel für linear separierbare Daten ist in Abbildung 3 zu sehen. Hier kann man gut erkennen das sich auf einer Seite der Hyperebene (im $\mathbb{R}^2$ eine Gerade) auch nur eine Klasse von Daten (rot oder grün) befindet.

Eine lineare Klassifikation, deren Ausgaberaum mehr als 2-dimensional ist ($Y = /1, ..., n/$, lässt sich durch n binäre Klassifikationen simulieren. Hierbei wird für jede Klasse eine binäre Klassifikation durchgeführt und dann die Eingabe x der Klasse zugeordnet, deren Hyperebene am weitesten entfernt ist.

Um die optimale Hyperebene zu einer binären Klassifikation mittels „Maximum Margin Classificator“ zu finden, benötigen wir einige Definitionen:
Der Abstand γ_i (bzw. Margin) zwischen einem Beispiel (x_i, y_i) und einer Hyperebene ist gegeben durch:

$$\gamma_i = y_i(\langle \omega \cdot x_i \rangle + b)$$

Des weiteren ist der Abstand einer Hyperebene γ zu mehreren Trainingsdaten das Minimum über alle γ_i.

$$\gamma = min_{i \leq l} \gamma_i$$

Die optimale Hyperebene ist nun diejenige, deren Margin zu dem Trainingsset maximal ist. Hierzu werden 2 zu der Hyperebene parallele Ebenen mit dem Abstand γ definiert.

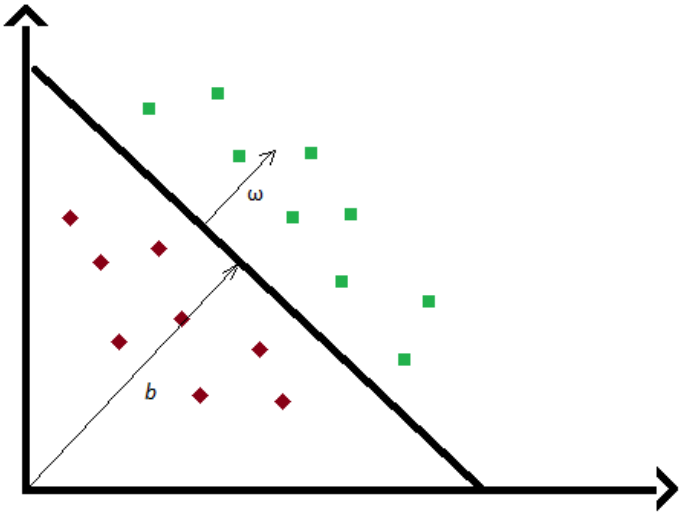

Abbildung 3: Beispiel für linear separierbare Eingabevektoren. Die Vektoren werden mit Hilfe von h(x) in 2 Klassen überhalb (positiv bzw. grün) und unterhalb (negativ bzw. rot) der Geraden eingeteilt. ω bestimmt hierbei die Steigung der Geraden, während b sie parallel verschiebt.

Die Vektoren, die γ beschränken, deren Abstand zur Hyperebene minimal ist, sind die „Support-Vectors“. Damit lässt sich der γ über den Normalenvektor der Hyperebene berechnen:

$$\gamma = \frac{1}{\|\omega\|_2}$$

Damit ergibt sich folgendes Optimierungsproblem:

$$\text{minimiere}_{\omega,b} \qquad \langle \omega \cdot \omega \rangle,$$

$$\text{in Abhängigkeit von} \qquad \gamma_i(\langle \omega \cdot x_i \rangle + b) \geq 1$$

Mit Hilfe der Formel von Lagrange lässt sich dieses Problem durch die Lagrange-Multiplikatoren α_i in primärer Form schreiben:

$$L(\omega, b, \alpha) = \frac{1}{2}\langle \omega \cdot \omega \rangle - \sum_{i=1}^{l} \alpha_i[\gamma_i(\langle \omega \cdot x_i \rangle + b) - 1]$$

Durch Ableiten nach ω und b und anschließender Resubstitution in die primäre Form ergibt sich das Optimierungsproblem in dualer Form:

$$\text{maximiere} \qquad L(\omega, b, \alpha) = \sum_{i=1}^{l} \alpha_i - \tfrac{1}{2} \sum_{j=1}^{l} \gamma_i \gamma_j \alpha_i \alpha_j \langle x_i \cdot x_y \rangle$$

in Abhängigkeit von $\sum_{i=1}^{l} \gamma_i \alpha_i = 0$

$$\alpha_i \geq 0 : i = 1, ..., l$$

Dieses Problem kann iterativ gelöst werden.[Lä06]

4.3 Kerneltrick

Wie bereits angesprochen lassen sich nicht alle Klassen im Objektraum durch Hyperebenen trennen. Häufig sind die Klassen nicht linear separabel und können somit nicht durch den „Maximal Margin Classifier“ getrennt werden. Um Datenpunkte dieser Klassen dennoch richtig zuordnen zu können, bedient man sich des Kerneltricks. Hierbei wird der Objektraum X durch die Funktion ϕ auf einen höherdimensionalen Merkmalsraum F abgebildet:

$$\phi : X \subseteq \mathbb{R}^l \rightarrow F \subseteq \mathbb{R}^k \; mit \; l < k$$

In dem Merkmalsraum F kann nun die SVM eine Hyperebene finden und somit können die Datenpunkte klassifiziert werden. In Abbildung 4 sind Datenpunkte zweier Klassen (rot und grün) zu erkennen, die durch die Funktion ϕ in den Merkmalsraum übertragen werden und somit linear separabel werden.

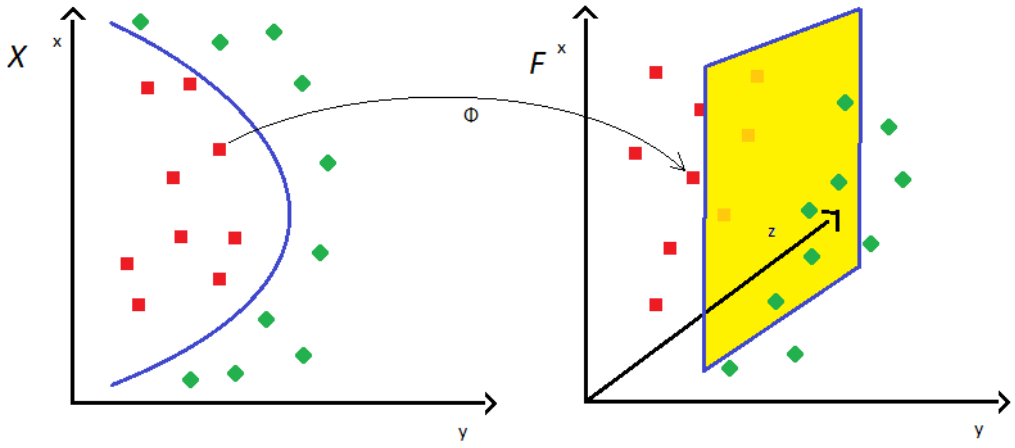

Abbildung 4: Die nicht linear separablen Datenpunkte im Objektraum $X \subseteq \mathbb{R}^2$ (linkes Koordinatensystem) werden durch die Funktion ϕ in einen Merkmalsraum $F \subseteq \mathbb{R}^3$ (rechtes Koordinatensystem) abgebildet und können dort durch eine Ebene getrennt werden.

Da die Vektoren im Optimierungsproblem nur als Skalarprodukte auftreten, muss man den Merkmalsraum nicht kennen. Es reicht, wenn man die Skalarprodukte im Objektraum

durch eine Funktion K berechnen kann:

$$K(x, y) = \langle \phi(x) \cdot \phi(y) \rangle, \ x, y \in X$$

Eine solche Abbildung nennt man Kernel. Diese lässt sich für das Skalarprodukt in das Optimierungsproblem einsetzen, sodass man nun die Datenpunkte klassifizieren kann.[Lä06, Say]

4.4 LIBSVM

Die Featurevektoren werden wie in den Abschnitten 3.5 und 3.6 beschrieben mit der Unreal Engine berechnet und jeder Klasse eine eindeutige Zahl zugewiesen. Danach wird ein Trainingsset $(x_i : y_i), x_i = (x_{i1}, x_{i2}, ..., x_{il}) \in X, y_i \in Y$ LIBSVM in einer Textdatei folgendermaßen übergeben:

...
$y_i \quad 1 : x_{i1} \quad 2 : x_{i2} \quad ... \ l-1 : x_{il-1} \quad l : x_{il}$
$y_{i+1} \ 1 : x_{i+11} \ 2 : x_{i+12} \ ... \ l-1 : x_{i+1l-1} \ l : x_{i+1l}$
...

Als Implementierung der SVM haben wir die C++ bzw. Java-Bibliothek LIBSVM genutzt. Der Vorteil hierbei ist, dass diese frei verfügbar ist und gewisse Funktionen bereits enthält. So kann ein Datensatz aufgeteilt werden, um mit dem ersten Teil zu trainieren und mit dem zweiten Teil zu testen, wie gut die durch das Training erstellte Zuordnung ist (Cross-Validation). Dabei wird der Prozentsatz der richtig klassifizierten Datensätze ausgeben, sodass man sehr schnell und einfach einen Vergleich der ausgewählten Features durchführen kann.

4.5 Auswahl der Featurevektoren

Die richtige Anzahl und Auswahl der Merkmale im Featurevektor ist essenziell für eine gute Klassifizierung, da die Vektoren unterschiedlicher Klassen auch möglichst unterschiedlich sein müssen, damit die SVM trennende Hyperebenen findet. Deshalb haben wir uns zunächst für einen Merkmalsraum in der 10. Dimension entschieden, um möglichst viele Unterschieden zwischen Klassen erkennen zu können. Unsere Features hierbei waren:

- Blickrichtung $b \in \mathbb{R}^3$
- Richtungsvektor von Kopf zu rechter Hand $r \in \mathbb{R}^3$
- Richtungsvektor von Kopf zu linker Hand $l \in \mathbb{R}^3$

Wie in Abbildung 5 zu sehen, sind die Vektoren einer Klasse häufig nicht geclustert, sondern über große Bereiche verteilt. So sieht man zum Beispiel, dass die Daten der Aktion „PC benutzen“ in zwei Bereichen auftauchen. Diese Aktion wurde von uns in zwei verschiedenen und unterschiedlich orientierten Szenen aufgenommen. Dadurch, dass wir die

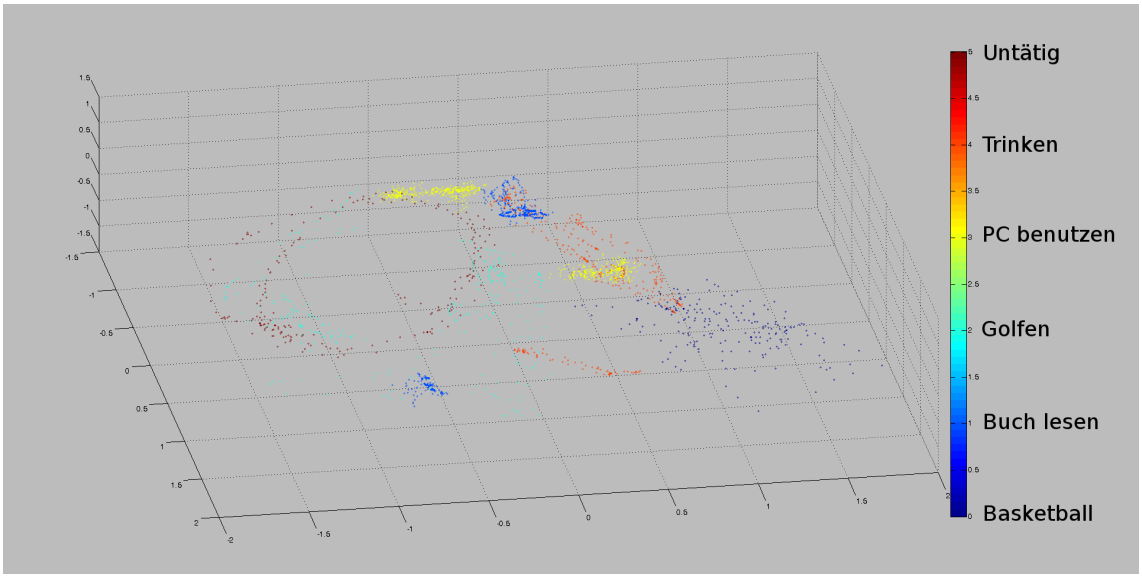

Abbildung 5: Hier sind die Datensätze der ersten Features (b, r, l) zu sehen. Um den 10-Dimensionalen Featureraum beobachtbar zu machen, wurde er per Multi-Dimensional-Scaling in den $\mathbb{R}^3$ überführt.

Blickrichtung b als Feature genommen haben, sind die Vektoren je nach Szene sehr unterschiedlich, obwohl jeweils die gleiche Aktion ausgeführt wurde. Dies sorgt dafür, dass die einzelnen Aktionen nicht gut voneinander getrennt werden können und somit eine passende Klassifikation fehlschlägt. Daran erkennt man sehr gut, dass die Blickrichtung als Feature nicht geeignet ist. Auch die Vektoren zu den Händen l, r sind richtungsabhängig und daher als Merkmale ebenfalls nicht gut geeignet.
Dieser Featurevektor ist also nicht gut besonders ideal um Aktionen zu klassifizieren. Das zeigt sich auch in der Trainings- und Testfunktion der LIBSVM. Dort war die Klassifizierung der Testdaten nach dem Training nur bei etwas über 40%. Bei 6 Aktionen ist das zwar deutlich über einer randomisierten Klassifikation, aber für den betriebenen Aufwand deutlich zu wenig.
Deshalb haben wir uns dann entschlossen die Features etwas zur reduzieren, sodass diese innerhalb einer Klasse möglichst ähnlich bleiben und nicht von dem verwendeten Szenen oder der Richtung abhängen. Außerdem haben wir die in [SCH+14] definierten Bounding Boxes um den Körper genutzt, um Informationen über möglicherweise verwendete Szenenelemente zu bekommen. Dadurch ergaben sich folgende Merkmale:

- Höhe des Kopfes h
- kleinere Entfernung vom Kopf zur Hand $min(\|l\|, \|r\|)$
- größere Entfernung vom Kopf zur Hand $max(\|l\|, \|r\|)$
- Gewicht des Objektes mit dem geringsten Abstand zu einer Hand g

Um eine Unabhängigkeit bei der Ausführung einer Aktion mit rechter oder linker Hand zu gewährleisten, haben wir die Entfernungen zu den Händen als größere und kleinere Entfernung in den Featurevektor eingetragen. Somit haben wir einen Featurevektor der Aktionen gut unterscheidet, auch wenn diese in unterschiedlichen Szenen oder in unterschiedlichen Richtungen ausgeführt werden.

4.6 Qualität der Featurevektoren

Als ersten Test der neuen Featurevektoren haben wir die Testfunktion der LIBSVM genutzt. Hierbei bekamen wir eine richtige Klassifizierung in 98% der Testdaten. Eine so gute Klassifizierung kann häufig auch einer „Überfittung“ geschuldet sein. Das bedeutet, dass die Hyperebenen die Datenpunkte sehr eng umschließen und somit sehr ähnliche Featurevektoren sehr gut erkannt werden, aber sobald eine leichte Abweichung des Standardfalls auftritt, wird der Vektor der falschen Klasse zugeordnet. Um dies auszuschließen, haben wir mehrere Live-Klassifizierungs-Tests gemacht, in denen wir die Aktionen mit sehr unterschiedlichen Bewegungsabläufen wiederholt haben. Auch hier wurde unsere Bewegung sehr gut klassifiziert, sodass eine Überfittung ausgeschlossen werden konnte. Wie in Ab-

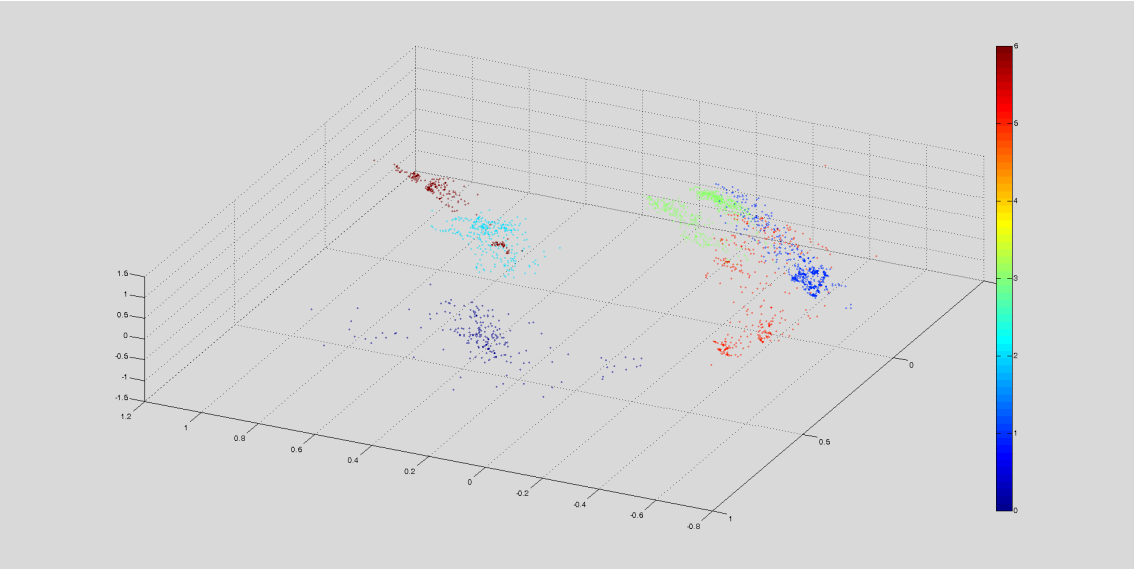

Abbildung 6: Hier sind die Datensätze der finalen Features zu sehen. Auch hier wurde der Featureraum mittels Multi-Dimensional-Scaling in den $\mathbb{R}^3$ überführt.

bildung 6 zu sehen, sind die Datenpunkte einer Aktion jeweils in einem Bereich geclustert. Dadurch kann die SVM sehr leicht trennende Hyperebenen finden, sodass eine Aktion richtig klassifiziert wird.

4.7 Gewicht ohne Virtual Reality

Um Aktionen ohne Virtual Reality durchzuführen, müssen sowohl die Datenaufnahme als auch die Featurevektoren sinnvoll überführt werden. Die Datenaufnahme kann durch verschiedene Scansysteme ausgeführt werden, sodass Bewegungen und Aktionen in für Computer verständliche Daten verarbeitet werden. Somit können Abstände sehr gut eingescannt werden. Um allerdings das Gewicht der Objekte zu erfassen, müsste jedes verwendete Objekt gewogen werden. Da dies im Allgemeinen nicht praktikabel ist, kann das Gewicht nicht als Feature zur Klassifizierung genutzt werden. Allerdings können andere Größen von Objekten besser erfasst in somit verwendet werden. Zum Beispiel kann durch einen Scanner das Volumen oder die Oberfläche eines Objektes gut eingelesen werden. Das kann selbst dann funktionieren, wenn das Objekt nicht alleine in einem Raum ist. Also können auch ohne Virtual Reality Eigenschaften von Objekten für die Klassifizierung von Aktionen genutzt werden. Das diese Technik zu einer sehr guten Qualität der Klassifikation führt, zeigt die zuverlässige Vorhersage während der Live-Klassifizierung.

5 Limitationen

Wir konnten den Aufwand Aktionen in realen Szenen aufzunehmen durch die Verwendung von virtuellen Szenen und dem VR-System HTC Vive stark reduzieren. Allerdings treten dadurch auch neue Schwierigkeiten auf. Mit der HTC Vive lassen sich nur die Positionen und Orientierungen der Hände und des Kopfs verfolgen. Bei vielen Aktionen ist jedoch auch die Interaktion von anderen Körperteilen mit der Umgebung entscheidend. Zum Beispiel ist es für die Aktion „Fußball spielen“ entscheidend, dass sich ein Ball in der Nähe eines Fußes befindet. Um solche Aktionen genau zu beschreiben, ist es dann notwendig, mehrere Tracker zu verwenden. In Abbildung 7 sind die relevanten Positionen für solche Tracker markiert. Ein interessanter Ansatz mit weniger Trackern auszukommen wird in [JYF16] beschrieben. Dort wird eine Methode vorgestellt, mit der sich die Körperhaltung in Echtzeit nur aus den Positionen und Orientierungen von Kopf und Händen rekonstruieren lässt.

Wie im Abschniit 4.5 beschrieben, verwenden wir neben der Masse des nächsten Gegenstandes die Kopfhöhe und den minimalen und maximalen Abstand der Hand zum Kopf. Diese werden in unserem Fall 5 mal in der Sekunde aufgenommen und beschreiben so die dargestellte Pose. Eine Aktion ist eine geordnete Abfolge dieser Merkmale. Wir haben aber die Reihenfolge der Merkmale beim Trainieren unserer SVM nicht berücksichtigt. Die Merkmalsvektoren sind daher auch bei verschiedenen Aktionen oftmals recht ähnlich. In Abbildung 8 ist das gut zu erkennen. Hier liegen die Merkmale der Aktionen Basketball (dunkelblau) und Trinken (dunkelrot) nah zueinander, da sowohl beim Basketball als auch beim Trinken eine Nickbewegung des Kopfes ausgeführt wird.

Aufgrund der Ähnlichkeit fällt in diesen Fällen die Masse des nächsten Gegenstandes stärker ins Gewicht. Haben dann die Szenenobjekte ähnliche Massen, wie die Getränkeflasche und der Basketball, kann das zu Fehlklassifizierungen führen. Wird zum Beispiel im Stehen aus einer Flasche getrunken, so kann die Aktion fälschlicherweise als

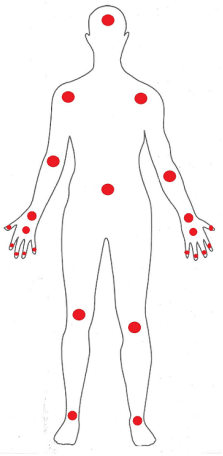

Abbildung 7: Menschlicher Körper. Die roten Punkte beschreiben die annehmbaren Positionen der Tracker

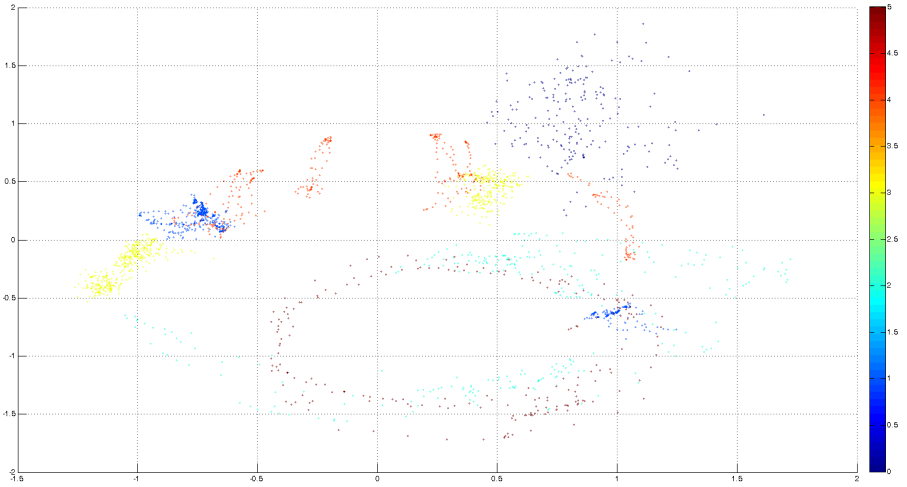

Abbildung 8: Featurevektor-Diagramm. Rechts sind die Aktionen und deren Farben im Diagramm aufgelistet. Von unten nach oben: 1 - Basketball spielen, 2 - Buch lesen, 3 - Golfen, 4 - PC benutzen, und 5 - trinken

"Basketball spielen"klassifiziert werden.

6 Diskussion und Ausblick

Wir haben eine Methode zur Erforschung von Aktionen vom Menschen in virtueller Realität vorgestellt. Diese Methode bildet Merkmalsvektoren aus der Pose eines Menschen und den Eigenschaften der Gegenstände aus der Szene und verwendet dabei eine SVM als Klassifizierer. Um die Pose der durchführenden Person darzustellen, wurden die Kopfhöhe und die Abstände von den Händen zum Kopf genutzt und unter Berücksichtigung der Körpermaße des Nutzers kalibriert.

6.1 Maschinelles Lernen kombiniert mit Automaten

Da wir die Reihenfolge der Merkmalvektoren in Relation mit den Aktionen nicht berücksichtigen, bietet es sich an, die Pose in Zustände zu unterteilen und die Reihenfolge der verschiedenen Zustände und die jeweilige Interaktion mit den Gegenständen als eine Aktion zu betrachten. Damit sollte die verwendete SVM in der Lage sein, nicht lineare multidimensionale Funktionen voneinander zu unterscheiden.

6.2 Körperrekonstruktion

Für genauere Aktionserkennungen sollte die Pose des Nutzers besser rekonstruiert werden. Hierfür bietet es sich an, mehr Körperteile zu tracken oder die in [JYF16] beschriebene Methode zur Echtzeit-Körperrekonstruktion zu implementieren.

Literatur

[EG17] Epic Games. Unreal Engine. Website, 2017. `https://www.unrealengine.com/`; abgerufen am 16. März 2017.

[FPB+06] G. Fritz, L. Paletta, R. Breithaupt, E. Rome, and G. Dorffner. Learning predictive features in affordance based robotic perception systems. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3642–3647, Oct 2006.

[J77] Gibson J J. *"The theory of affordances," in Perceiving, Acting, and Knowing. Towards an Ecological Psychology.* Number eds Shaw R., Bransford J. Hoboken, NJ: John Wiley & Sons Inc., 1977.

[JYF16] Fan Jiang, Xubo Yang, and Lele Feng. Real-time full-body motion reconstruction and recognition for off-the-shelf vr devices. In *Proceedings of the 15th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry - Volume 1*, VRCAI '16, pages 309–318, New York, NY, USA, 2016. ACM.

[Lä06] Johannes Lächele. Support vector machines. Proseminararbeit an der Universität Tübingen, July 2006. Online erhältlich unter `http://www.ra.cs.uni-tuebingen.de/lehre/ss06/pro_learning/JohannesLaechele.pdf`; abgerufen am 17 März 2017.

[MLBSV08] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor. Learning object affordances: From sensory–motor coordination to imitation. *IEEE Transactions on Robotics*, 24(1):15–26, Feb 2008.

[Say] Dr. Sead Sayad. Support vector machines. Vorlesungsskript an der Universität Toronto. Online erhältlich unter `http://chem-eng.utoronto.ca/~datamining/Presentations/SVM.pdf`; abgerufen am 17 März 2017.

[SCH+14] Manolis Savva, Angel X. Chang, Pat Hanrahan, Matthew Fisher, and Matthias Nießner. Scenegrok: Inferring action maps in 3d environments. *ACM Trans. Graph.*, 33(6):212:1–212:10, November 2014.

[SLZ+08] Michael Stark, Philipp Lies, Michael Zillich, Jeremy Wyatt, and Bernt Schiele. *Functional Object Class Detection Based on Learned Affordance Cues*, pages 435–444. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[SMBR10] Jie Sun, Joshua L. Moore, Aaron Bobick, and James M. Rehg. Learning visual object categories for robot affordance prediction. *The International Journal of Robotics Research*, 29(2-3):174–197, 2010.

[Viv17] Vive. HTC Vive. Website, 2017. `https://www.vive.com/de/`; abgerufen am 16. März 2017.