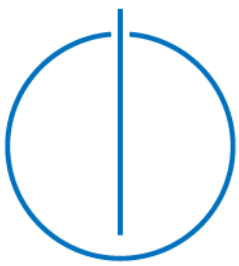
Technische Universität München

Fakultät für Informatik

Bachelorarbeit in Informatik: Games Engineering

Deutgestenerkennung zur Steuerung von intelligenten Umgebungen

Alexander Biederer



Technische Universität München

Fakultät für Informatik

Bachelorarbeit in Informatik: Games Engineering

Deutgestenerkennung zur Steuerung von intelligenten Umgebungen

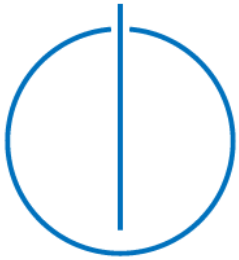
Pointing Gesture Recognition for Controlling Intelligent Spaces

Autor: Alexander Biederer

Aufgabenstellerin: Prof. Dr. Gudrun Klinker

Betreuer: Sandro Weber

Abgabedatum: 15.11.2017



ERKLÄRUNG

Ich versichere, dass ich diese Bachelorarbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Datum, Unterschrift

Deutgestenerkennung zur Steuerung intelligenter Umgebungen

ABSTRACT

Im Rahmen dieser Arbeit wird die Eignung von Deutgesten zur Steuerung intelligenter Umgebungen evaluiert. Dabei werden unterschiedliche Herangehensweisen an das Problem der Deutgestenerkennung in einer praktischen Implementierung umgesetzt und diskutiert. Der Fokus liegt bei der Implementierung auf der kameragestützten, performanten Identifikation und Erkennung von Deutgesten. Mithilfe der Referenzimplementierung werden dann im Rahmen einer praktischen Studie Deutgesten auf Praktikabilität, Intuitivität und Akzeptanz im Kontext der Steuerung von intelligenten Umgebungen evaluiert.

The goal of this thesis is to evaluate the applicability of pointing gestures for controlling intelligent spaces. Therefore, different approaches for pointing gesture detection will be implemented and discussed. A low cost and efficient camera based approach will be chosen to evaluate the acceptance towards and how practical and intuitive and pointing gestures can be for controlling intelligent spaces.

VORWORT

Ich möchte allen danken, die mir geholfen haben, dieses Projekt umzusetzen. Zuerst möchte ich Professor Gudrun Klinker und dem Lehrstuhl für Augmented Reality danken, mir die Arbeit an dieser Thesis ermöglicht zu haben. Sandro Weber möchte ich für seine Unterstützung während der Arbeit, aber auch für die Freiheiten, welche ich währenddessen hatte, danken. Florian Bayer möchte ich danken für seine Arbeit am Interface zu intelligenten Umgebungen, welche meine Arbeit erst möglich gemacht hat. Zuletzt geht mein Dank an alle anonymen Probanden, welche sich die Zeit genommen haben, an meiner Studie teilzunehmen und Feedback zu geben.

Inhaltsverzeichnis

INHALT

Einleitung

MOTIVATION

Der Begriff „Intelligente Umgebung“ (Smart Environment) bezeichnet eine „physikalische Welt die, tief und unsichtbar verknüpft ist mit Sensoren, Aktuatoren, Displays und Verarbeitenden Elementen, die nahtlos in tagtägliche Gebrauchsgegenstände eingebunden, und durch ein durchgehendes Netzwerk verbunden sind.“ [1]

Smart Environments sind auf dem Vormarsch, und immer mehr Umgebungen werden intelligent. Sprachassistenten, wie Amazon‘s Alexa, oder Apple‘s Siri, dienen nun neben herkömmlichen Schaltern und Fernbedienungen als Sensoren, die Aktuatoren wie Lampen, Fernseher, Radios und Toaster kontrollieren. Displays finden sich auf jedem zweiten Kühlschrank und auch die ein oder andere Zeitung ist dem Touchscreen gewichen. Dabei ist alles verbunden mit dem heimischen (oder öffentlichen) WLAN.

All diese Neuerungen, die erst durch den Fortschritt der Technologie ermöglicht wurden, vereinfachen das Leben des Benutzers. Anstatt morgens den Wecker von Hand vom Nachtkästchen zu schlagen, reicht heute „Alexa, Stopp!“, der Kühlschrank benachrichtigt den Benutzer, wenn die Milch alle ist, und bestellt im besten Falle gleich noch eine nach.

Die Sensoren werden dabei immer intuitiver, ergonomischer und zugänglicher: Lichter können nicht mehr nur über Schalter an der Wand gesteuert werden; mittlerweile gibt es Fernbedienungen, Smartphone-Apps oder sogar Sprachassistenten, die die Leuchtmittel direkt kontrollieren. Die Aktuatoren sind aber auch immer komplexer geworden: Die Leuchtmittel lassen teilweise ihre Helligkeit, sowie die Farbe stufenlos einstellen.

Der Aufwand, den der Benutzer zur Steuerung seines Heimes tätigen muss, wird ständig reduziert, und die Bedienung so vereinfacht, dass heutzutage selbst Kinder dazu in der Lage sind. Die Verwendung von komplexen Fernbedienungen, die unter Umständen dann noch Überladen mit Funktionalität sind, ist einfach nicht mehr notwendig. Neue Geräte benötigen im besten Fall nur einen Bruchteil der Zeit, bis der Nutzer die Bedienung verstanden hat. Dies wird erreicht dadurch, dass bereits bestehende Fähigkeiten wie das Sprechen, oder die Bedienung eines Touchscreens zur

Steuerung benutzt werden. Generell gilt: je universaler und einfacher die Fähigkeit, die der Bedienung zu Grunde liegt, desto intuitiver wird diese Bedienung aufgefasst.

Eine der einfachsten und universellsten Fähigkeiten, die zur Bedienung verwendet werden kann, ist nun das Deuten. Bereits im Alter von 12 Monaten verwenden Kinder das Deuten als prä-vokabulare Ausdrucksart [2] , damit zählt es zu den frühsten erlernten Fähigkeiten, welche sich gleichzeitig noch zur Bedienung von intelligenten Systemen eignen könnten. Dabei ist das Deuten zusätzlich auch noch unabhängig von der Erziehung und dem Kontext der Kinder; In allen Kulturen wird das Deuten ähnlich verstanden und praktiziert. In der Evolution des Menschen ist das Deuten sogar schon bei Schimpansen wahrnehmbar. [3]

Dies alles spricht für das Potential, welches das Deuten als intuitive Bedienungsform birgt.

AUFGABE

Das Potential des Deutens als Möglichkeit zur Steuerung soll im Laufe dieser Arbeit erörtert werden. Bedient werden sollen hierbei intelligente Umgebungen, also Räume mit vernetzten Sensoren und Aktuatoren. Als Aktuatoren werden hier Lichter, sowie Fernseher als häufigste Vertreter von intelligenten Aktuatoren verwendet, das angewendete Prinzip ist allerdings auf jede Art von Aktuator anwendbar.

Um das Potential von Deutgesten zur Steuerung von intelligenten Umgebungen testen zu können, wird ein Beispielsystem entwickelt, welches Deutgesten in Kommandos für intelligente Aktuatoren umwandelt.

Im Anschluss soll mithilfe des fertigen Systems das Deuten als Bedienungsform, sowie das entwickelte System selbst im Hinblick auf Intuitivität, Praktikabilität und Akzeptanz überprüft werden.

DEFINITION VON DEUTEN

Das Deuten selbst wird von Duden als „(mit dem Finger, einem Gegenstand) auf etwas zeigen, hinweisen“ [4] beschrieben. Um eine Deutgeste jedoch von einer anderen Geste, zum Beispiel von einem ausgestreckten Daumen oder dem Winken mit der Hand zu unterscheiden, muss das Deuten erst Formal definiert werden.

Eine Deutgeste ist deshalb im Kontext dieser Arbeit definiert am Vorbild Kleinkind als Vorgang des Ausstreckens und Haltens des Armes mit ausgestrecktem Zeige- oder Mittelfinger (ab jetzt zeigender Finger genannt) und nicht-ausgestreckten Ring- und kleinem Finger, wobei die Richtung des ausgestreckten Armes gleich oder ähnlich zur Richtung des ausgestreckten Fingers ist. Der Arm ist dabei weg vom Körper gestreckt, Arm und Finger definieren zusammen einen dreidimensionalen Richtungsvektor (hier genannt $\overline{v}$). Dieser Vektor entspringt an der Fingerspitze des zeigenden Fingers (hier genannt P), und verläuft in die Richtung $\overline{v}$, welche vom Finger (und laut Definition auch durch den Arm) vorgegeben werden.

Das etwas, auf das gezeigt wird, also das Ziel der Deutgeste, soll hier definiert sein als Körper t_i, welcher einen Punkt T_i enthält, welchen der Richtungsvektor zuerst schneidet, oder falls es keinen solchen Schnittpunkt gibt, als Körper bei dem der Winkel φ zwischen den Geraden $\overline{PT_i}$ und $\overline{v}$, minimal ist. T_i ist dann hierbei der Punkt im Zielkörper, welcher den geringsten Abstand zu $\overline{v}$ hat.

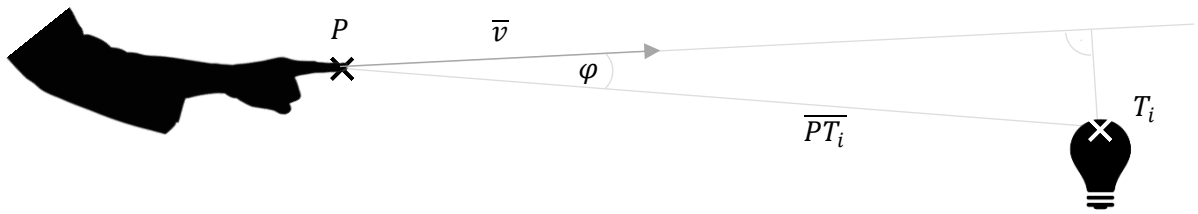


Fig. 1 Schematische Zeichnung einer Deutgeste

Der Winkel φ ist dabei dann definiert als:

$$1)\ \varphi_{T,P,\overline{v}} = \cos^{-1}\frac{\overline{PT} \circ \overline{v}}{|\overline{PT}| \cdot |\overline{v}|}$$

Für einen beliebigen Körper mit Körperindex k kann unter der Annahme eines konstanten P und eines dann nur von k abhängigen T_k kann die Funktion vereinfacht werden zu

$$2)\ \varphi_k = \cos^{-1}\frac{\overline{PT_k} \circ \overline{v}}{|\overline{PT_k}| \cdot |\overline{v}|}$$

Mit anderen Worten erfüllt der Zielkörper t_i (der Körperindex des gesuchten Körpers ist i) folgende Gleichung:

$$3)\ \varphi_i = \min(\varphi_1, \dots, \varphi_n)$$

für alle Körper $t_1, \dots, t_n$.

Bei einem direkten Schnitt von $\overline{v}$ und dem Objekt mit Index k gilt: $\varphi_k = 0°$

Damit ist Gleichung 3) auch bei einem direkten Schnitt erfüllt, und die beiden Fälle müssen nicht unterschieden werden.

Um zu vermeiden, dass mögliche Zielkörper hinter dem zeigenden Finger in Betracht gezogen werden, muss der Punkt T_k eines Körpers t_k auf der Seite liegen, in die auch $\overline{v}$ zeigt:

$$4)\ \frac{\overline{F_i P}}{|\overline{F_i P}|} = \overline{v}$$

F ist dann hierbei der Punkt auf $\overline{v}$, welcher den geringsten Abstand zu T_k hat.

Nun ist jedoch nicht jeder Körper ein Ziel, auch wenn für ihn die Gleichungen 3) und 4) gelten. Falls es beispielsweise keinen Körper gibt, auf den gezeigt wird, dann ist nicht derjenige Körper, welcher die Gleichungen erfüllt, Ziel dieser Deutgeste. Um eine falsche Zuordnung in diesem Fall zu vermeiden darf also der Zielkörper nicht zu weit vom eigentlichen Zentrum der Deutgeste entfernt sein. Der Winkel φ darf also ein gewisses maximum $\Delta\phi_{max}$ nicht überschreiten:

$$5)\ \varphi < \Delta\phi_{max}$$

Um sicherzustellen, dass bei mehreren Körpern $t_1, \dots, t_n$ mit ähnlichem $\varphi_1, \dots, \varphi_n$, entsprechend einer visuellen Verdeckung, derjenige als Deutziel identifiziert wird, welcher am nächsten am zeigenden Finger liegt:

$$Für\ t_1, \dots, t_n\ mit\ \varphi_1, \varphi_2, \dots, \varphi_{n-1}, \varphi_n$$
$$und\ d(k) = |\overline{F_k P}|$$

$$6)\ d(i) = \min(d(1), \dots, d(n))$$

Entwicklung eines Deutgestenerkenners

ANFORDERUNGEN

Durch die stationäre Natur von intelligenten Umgebungen bieten sich stationäre Lösungen für die Erkennung von Deutgesten an. Diese haben den Vorteil, dass sie nicht auf mobile Stromversorgung, z.B. durch Akkus, angewiesen sind, und sich einfach in bestehende, kabelgebundene Netzwerke eingliedern lassen. Der Deutgestenerkenner soll also als stationärer Sensor im intelligenten Raum agieren.

Im Hinblick darauf, dass (Deut-) Gestensteuerung in vielen unterschiedlichen Kontexten denkbar ist, soll der vorgeschlagene Deutgestenerkenner diese möglichen Kontexte nicht einschränken. Zudem ist teure Hardware ein Faktor, welcher sich negativ auf eine potenzielle Kundschaft eines solchen Systems auswirkt. Deshalb soll der Erkenner auch auf günstiger Hardware gut funktionieren. Ein Raspberry Pi 3 wurde im Rahmen dieser Arbeit als Maßstab und Zielsystem des Erkenners verwendet.

Doch nicht nur die rechnende Hardware, sondern auch eventuelle Sensoren sollten sich preislich in Grenzen halten.

Neben den kontextuellen Anforderungen gibt es natürlich auch Anforderungen an den Erkenner selbst: Er soll über eine gute Erkennungsquote von Deutgesten verfügen, dabei soll die Auftrittswahrscheinlichkeit von Falscherkennungen möglichst niedrig gehalten werden. Anders formuliert solle die Auftrittswahrscheinlichkeit von false negatives und false positives minimiert werden.

Im Raum soll der Erkenner den normalen Tagesablauf nicht stören, vor allem soll er den Bewegungsfreiraum des Benutzers nicht einschränken. Zudem ist die Deutgestenerkennung optimalerweise auch unabhängig von Peripher-Geräten des Gastes, wie z.B. Smartphones oder Smartwatches.

Zudem soll eine Deutgestenerkennung weitgehend unabhängig von Einflussfaktoren wie dem Benutzer, der Umgebung, den Lichtverhältnissen, der Temperatur oder anderen Einflussgrößen in einem möglichst großen Teil der intelligenten Umgebung funktionieren.

VORGESCHLAGENE HERANGEHENSWEISE

Klärung der Begriffe

Sensor:	Gerät, welches im Kontext einer intelligenten Umgebung gewisse Eigenschaften quantitativ oder qualitativ erfassen kann
Aktuator:	Gerät, welches im Kontext einer intelligenten Umgebung elektronische Signale in mechanische Bewegung oder andere physikalischen Größen (Licht, Schall) umwandelt
(Deutgesten-) Erkenner:	Ein System, welches Deutgesten identifizieren und auswerten können soll
(Deutgesten-) Steuerung:	Ein Gesamtsystem, welches Deutgesten identifiziert, auswertet und in Kommandos für mögliche Aktuatoren umwandelt
Referenzimplementierung:	Der im Rahmen dieser Arbeit entwickelte Algorithmus zur Deutgestenerkennung
False positive:	(in diesem Kontext) Eine fälschliche Erkennung einer nicht vorhandenen Deutgeste
False negative:	(in diesem Kontext) Eine fälschliche Nichterkennung einer vorhandenen Deutgeste

Mobile Zeigegeräte

Eine Möglichkeit, eine Deutgestensteuerung zu implementieren, stellen Zeigegeräte (Pointer) dar. Ein Sensor, welcher an der Hand oder am Arm befestigt oder getragen wird übermittelt Daten, aus denen dann Deutrichtungen berechnet werden können. Vorstellbar ist beispielsweise die Verwendung einer Smartwatch, oder eines Smartphones und deren Kompass- Gyroskop- GPS- und Beschleunigungssensordaten. Aus diesen können P und $\overline{v}$ bestimmt werden.

Eine Schwierigkeit bei diesem Ansatz ist, zu erkennen wann es sich bei einer Geste um eine Deutgeste handelt, und wann nicht, da kein Rückschluss auf die Position und Anordnung der Finger gemacht werden kann. Zudem ist die Genauigkeit der Erkennung beschränkt durch die Genauigkeit der verwendeten Sensoren, und zumindest die Daten welche GPS als Mittel zur Positionsbestimmung liefert bergen die Gefahr, zu ungenau für eine robuste Erkennung zu sein.

Um ein wenig in die Zukunft zu denken: Andere Sensoren könnte ja ihren Weg in die Smartwatches der Zukunft finden: Mit gerichteten Kameras oder Lasern könnten Deutgesten auf ganz andere Arten erkannt werden: Eine an der Smartwatch angebrachte, in Deutrichtung zeigende Kamera könnte automatisch verschiedene Aktuatoren erkennen, identifizieren (z.B. mit einer am Aktuator angebrachten Markierung), und dann per WLAN triggern. Über diese Kamera könnte auch die von der Hand angenommene Pose differenziert werden.

Alle diese möglichen Ansätze teilen den inhärenten Nachteil, dass explizit Anforderungen an einen möglichen Benutzer gestellt werden. Um die Deutgestenerkennung verwenden zu können, muss der Benutzer ein solches Zeigegerät besitzen, und wahrscheinlich auch für den Gebrauch kalibrieren. Dies widerspricht den hier gestellten Anforderungen an die Funktionsweise eines Deutgestenerkennungssystems.

Stationäre Kameras

Aus den Anforderungen an einen Deutgestenerkenner bieten sich mehrere Herangehensweisen an, doch vor allem die Verwendung von einer oder mehreren stationären Kameras erscheint besonders geeignet: Zum einen schränken diese den Benutzer nicht in seinem Tagesablauf ein, zum anderen sind Kameras mit Ausreichender Auflösung schon sehr preisgünstig zu erwerben. Ein weiterer Vorteil von stationären Kameras ist, dass diese nur einmal aufgebaut und kalibriert werden müssen, und danach keine weiteren Anpassungen mehr benötigen. Deshalb werden im Kontext dieser Arbeit stationäre Kameras als Sensoren für die Deutgestenerkennung verwendet.

Ein weiterer Vorteil stationärer Kameras ist, dass diese von fast allen Außenfaktoren unabhängig sind, nur schlechte Lichtverhältnisse (zu hell, zu dunkel, zu stark wechselnd) haben möglicherweise einen Einfluss. Um in dunklen Umgebungen immer noch verwertbares Bildmaterial zu generieren, kommen auch Infrarotkameras in Frage.

Je nach Art und Anzahl der verwendeten Kameras lassen sich potenziell bis zu 100% der intelligenten Umgebung und möglichen Deutrichtungen abdecken. Dafür müssen die Kameras natürlich geeignet platziert werden, beispielsweise in den Ecken eines intelligenten Raumes, mit Blick zum Zentrum.

Alle verwendeten Kameras werden dann per Kabel oder kabellos an einen Rechner angeschlossen, und liefern diesem einen permanenten Strom von Bilddaten.

In der Referenzimplementierung wurden USB Webcams (Microsoft LifeCam 3000) verwendet.

Allgemeine Grundlagen von bildbasierter Deutgestenerkennung

Von nun an sollen in dieser Arbeit nur noch Bilderserien als Input für einen Deutgestenerkenner betrachtet werden. Die allgemeine Definition einer Deutgeste und eines Deutgestenziels lässt sich einfach auf den 2-dimensionalen Raum übertragen.

Der Erste Schritt zur Steuerung von intelligenten Umgebungen mithilfe von Deutgesten muss also nun die Identifizierung von P und $\overline{v}$ des sein, also dem Ursprung und der Richtung einer Deutgeste. Davor ist jedoch erst einmal wichtig, die Deutgeste an sich zu erkennen, also eine Deutgeste von allem anderen zu unterscheiden. Der hier verfolgte Ansatz versucht dies über computer vision.

Referenzimplementierung

Um den vorgeschlagene Herangehensweise zu überprüfen, muss diese natürlich auch implementiert werden. Die Zielhardware, ein Raspberry Pi 3, wurde durch eine Virtuelle Maschine mit vergleichbaren Leistungscharakterisika (4 Kerne mit je 1,5 GHz Taktung) emuliert. Als Betriebssystem wurde dabei ein Debian-Linux verwendet.

Zurückgegriffen wurde vor allem auf die freien Libraries OpenCV [5] und dlib [6], implementiert wurde in Python und C++.

Einzel- / Mehrbildbetrachtung

Der Bildstrom, welchen die Kameras liefern, kann als Serie von Einzelbildern entweder mit oder ohne Zusammenhang betrachtet werden. Betrachtet man Bilder immer im Kontext der vorangehenden und eventuell auch der nachfolgenden Bilder, so lassen sich einige zusätzliche Informationen gewinnen: Vor allem Bewegungen erschließen sich erst bei der Betrachtung mehrerer Bilder deutlich, und Bildsegmente lassen sich besser unterscheiden. Dabei darf der zeitliche Abstand zwischen den Bildern jedoch nicht zu groß werden, bei größerem Abstand gehen unter Umständen einige Informationen, die bei Betrachtung von Zwischenbildern gewonnen werden könnten, verloren. Solche geringen Abstände sind jedoch abhängig von der Bearbeitungszeit bei gegebener Hardware pro Bild potenziell nicht einhaltbar, weshalb versucht wird, die Bearbeitungszeit pro Bild möglichst zu minimieren.

Vordergrund-Hintergrund Segmentierung

Durch die Verwendung von stationären Kameras bietet sich die Segmentierung von Vordergrund und Hintergrund an. Vordergrund wären unter Betrachtung von Deutgesten optimalerweise alle Benutzer des Systems. Der Hintergrund wäre der Raum, in dem sich die Benutzer befinden, und welchen diese steuern wollen.

Eine gute Vordergrund-Hintergrund Segmentierung bietet den Vorteil, dass der interessante, zu betrachtende Bereich innerhalb eines Bildes innerhalb des Bildstroms mit nur einer Operation ohne

zu große Performanceeinbußen sehr stark eingegrenzt werden kann. Da eine Deutgeste immer mit einer Bewegung des Armes verbunden ist, ist die Kontur der Deutgeste nach einer Vordergrund-Hintergrund Segmentierung sehr gut erkennbar.

Der Input des Segmentierers ist eine Serie von Bildern, alle aufgenommen aus derselben Perspektive. Der gewählte Algorithmus „Mixture of Gaussians“ ordnet Pixelbereiche über eine Zuordnung zu verschiedenen bereits gebildeten Gaussglocken zu, und aktualisiert diese dann zeitgleich. Falls eine Zuordnung nicht erfolgen kann, wird dieser Bereich als Vordergrund identifiziert. Das Ergebnis des Algorithmus ist dann eine Pixelmaske mit Größe des Ursprünglichen Bildes, welches jedem Pixel entweder als Vordergrund (weiß), Hintergrund (schwarz), oder Schatten von Bewegung (grau) einstuft.

Fig. 2 Videosignal der Kamera

Fig. 3 Ausgabe des Hintergrundsegmentierers (ergänzt um eine Bounding Box)

In der Referenzimplementierung wurde zuerst mit einer naiven pixelweisen Vergleich mit einer Historie gearbeitet: Dabei wurde jeder Pixel mit dem Durchschnittswert der Pixel an derselben Position der letzten x Bilder verglichen, wobei x die Größe der Historie genannt wird. Diese Historie (oder auch Hintergrundmodell genannt) wurde dann um jedem neuen Pixel aktualisiert, zu alte Werte wurden wieder entfernt. Falls ein neuer Pixel um mehr als einen festgesetzten Prozentsatz vom Hintergrundmodell abwich, wurde dieser als Vordergrund markiert. Dieser Ansatz erwies sich trotz sehr guter Performance als eher ungeeignet, da er vor allem Kanten von bewegten Objekten mit homogener Farbe markiert, und generell sehr viel Rauschen produziert.

Deshalb wurde auf den BackgroundsubstractorMOG2 von OpenCV zurückgegriffen, welcher geeignet parametrisiert wurde, um die Performanceansprüche zu erfüllen, und gleichzeitig eine ausreichend

gute Differenzierung zwischen Vorder- und Hintergrund ermöglicht. Zusätzlich wurde Rauschen durch verwenden von morphologischem Öffnen verringert.

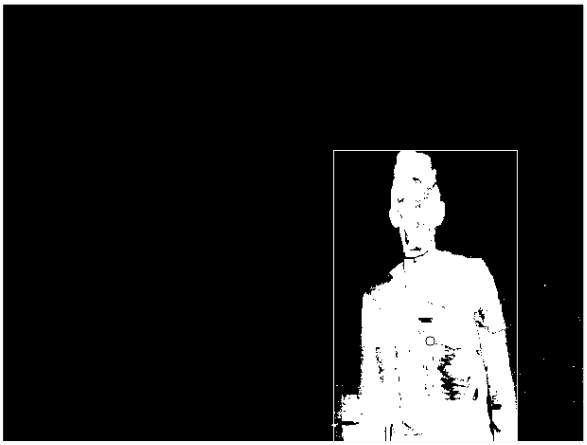


Fig. 4 Ausgabe des Hintergrundsegmentierers (ergänzt um eine Bounding Box)

Fig. 3 Rauschentfernung durch morphologisches Öffnen und Schließen

Weitere Eingrenzung des zu betrachtenden Bereichs

Durch die Vorder-Hintergrundsegmentierung wurde der interessante Bildausschnitt schon deutlich verringert. Konturen der Deutgesten sind deutlich auf der Maske erkennbar. Nun soll der genauer zu untersuchende Bereich weiter eingeschränkt werden. Dafür wird auf die Annahme zurückgegriffen, dass eine Deutgeste immer einen Arm beinhaltet. In der Maske ist der Arm erkennbar als zusammenhängender Bereich von Vordergrundpixeln, welche eine längliche Form aufweisen die in sich in eine Richtung verjüngt bis sie in der Zeigenden Hand endet.

Fig. 4 Deutgeste erkennbar auf Vordergrund-Maske

Um diese Eigenschaft auszunutzen gibt es mehrere Möglichkeiten:

Morphologische Operationen

Zuerst erscheint die morphologische Operation Top Hat (Differenz aus Bild und morphologischer Öffnung des Bildes) geeignet, die großen, gefüllten Bereiche (den Körper) aus der Maske zu entfernen, und nur die feineren Strukturen wie Arme und Beine zu konservieren. Diese Methode birgt aber auch einige Nachteile: Vor der Operation muss die ungefähre Größe der Arme bekannt sein, und je nach Kameraposition dann wieder verändert werden. Zudem ändert sich die Größe der Deutgeste mit wachsender oder schrumpfender Entfernung von der Kamera, und Deutgesten,

welche sich zu nah an der Kamera abspielen, könnten so nicht mehr erkannt werden. Des Weiteren muss für die Top Hat Operation ein Kernel verwendet werden, welcher mindestens so groß sein muss wie der Arm selbst, sodass bei der morphologischen Öffnung der Gesamte Arm abgedeckt wird. Je nach Auflösung des Bildes und maximal zugelassener Größe wird diese Kernelfunktion sehr groß, und Rechenzeit damit sehr lang. Aus diesen Gründen wird dieser Ansatz nicht weiterverfolgt.

In der Referenzimplementierung wurde diese Operation mittels der Top Hat Implementierung von OpenCV getestet.

Eine andere, auf morphologischen Operationen beruhende Möglichkeit ist das Skeletonizing. Dabei können Bereiche reduziert werden auf ein zentrales „Skelett“. Skeletonizing setzt sich zusammen aus iterativen Abfolgen von morphologischen Erosionen. Vor allem lange, dünne Bereiche ergeben langgezogene Skelette. Große Bereiche äußern sich in kurzen, eher verwobenen Skeletten. Diese Eigenschaft kann ausgenutzt werden, sodass nach dem Skeletonizing gezielt auf die langgezogenen Skelettlinien hin gesucht wird, und diese dann als Grundlage für weitere Untersuchungen der Maske oder des Bildes hin verwendet werden. Ein weiterer Vorteil ist, dass aus diesen Skelettlinien sehr einfach die Deutrichtung gewonnen werden kann. Nachteile des Skeletonizing ist hauptsächlich die Performance, welche zum Berechnen der Skelette benötigt wird. Zwar findet die Berechnung der Skelette in Echtzeit statt, doch bleibt vor allem bei großer Bildgröße und Auflösung wenig Spielraum für Folgeoperationen. Die Performance kann allerdings noch durch geeignetes Zuschneiden und Herunterskalieren der Bildausschnitte stark verbessert werden.

In der Referenzimplementierung wurde die Skeletonize-Implementierung von Félix Abecassis [7] zum Testen verwendet, und die resultierenden Skelettlinien mithilfe der OpenCV-Funktion findContours() in Konturen umgewandelt. Diese Konturen wurden dann nach Anzahl der enthaltenen Punkte gefiltert, und nur die Konturen betrachtet, welche eine bestimmte Länge übertrafen. Diese Konturen wurden dann mithilfe der Funktion fitLine() in Vektoren umgewandelt. Bei den Armlinien ergab das schon sehr gute Schätzungen der Deutrichtung.

Fig. 7 Gefilterte Vordergrundmaske

Fig. 5 Ergebnis der Skeletonize-Operation

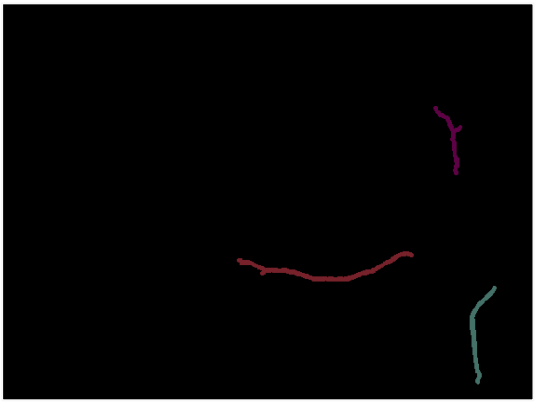


Fig. 9 Aus dem Skelett erkannte Konturen

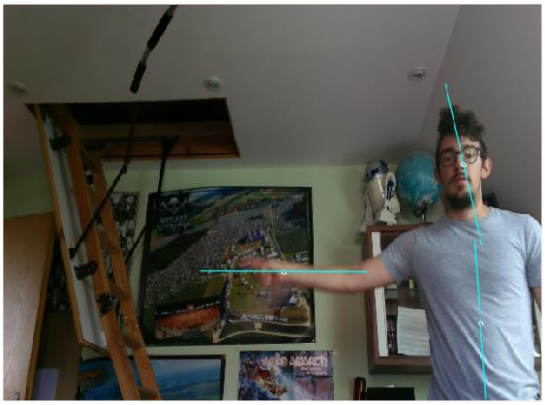


Fig. 6 Durch die Konturen gefittete Linien

Eingrenzung über vereinfachende Annahmen

Mithilfe ein paar weiterer Annahmen kann der auf Deutgesten zu durchsuchende Bereich sehr schnell eingegrenzt werden. Unter der Annahme, dass der Arm der Deutgeste auf dem 2D-Bild nicht in vertikale Richtung zeigt, können Zusammenhangkomponenten vor allem in horizontale Richtung hin auf das Vorhandensein von Armen hin überprüft werden. Dabei wird einfach für jeden Zusammenhangskomponenten (üblicherweise der Körper des Benutzers) spaltenweise von links und rechts außen nach innen die Anzahl der Vordergrundpixel pro Spalte summiert. Ein zu hoher Summenwert deutet darauf hin, dass es sich in der durchsuchten Spalte nicht mehr um einen Arm, oder eine andere dünne Struktur handelt. Wenn der Arm und die Deutrichtung zu vertikal sind, funktioniert diese Technik nicht, deshalb können keine vertikalen Deutgesten erkannt werden. Sollte dies aber gefordert sein, so kann man die Untersuchung nicht nur horizontal, sondern auch vertikal durchführen. Dann wären vor allem diagonale Deutgesten diejenigen, die am ehesten nicht erkannt

werden. Der größte Vorteil dieser Technik ist, dass ohne große Performanceeinbußen eine schnelle Aussortierung von zu großen Bereichen in Komponenten erfolgen kann.

In der Referenzimplementierung wurde dafür auf einige der Matrixoperationen der numpy Bibliothek und Funktionen von OpenCV zurückgegriffen.

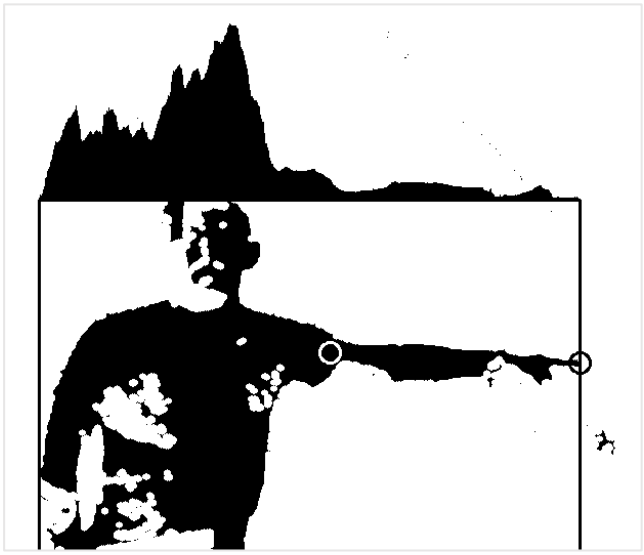


Fig. 7 Visualisierung der horizontalen spaltenweisen Summation

Erkennung der Deutgesten

Nun wurde der zu untersuchende Bildausschnitt schon deutlich reduziert, dennoch müssen immer noch die Deutgesten erkannt werden. Mehrere Ansätze wurden dafür auf ihre Eignung hin getestet. Dabei wurde vor allem bewertet nach den Kriterien Performance, Stabilität, und den Einschränkungen der Erkennung.

Konturbasierte Erkennung

Der Mensch kann Dinge unabhängig von ihrer Rotation erkennen. Da die zweidimensionale Abbildung einer Deutgeste in beliebige Richtungen zeigen kann (abhängig nach Deutziel und Standpunkt des Benutzers), und Richtung und Ursprung die für die Weiterverarbeitung zu ermittelnden Werte sind, muss auch ein Deutgestenerkenner die Deutgeste unabhängig von der Deutrichtung als solche klassifizieren, und die Richtung und den Ursprung bestimmen. Eine Möglichkeit, dies anzugehen, ist das Bild und alle zu untersuchenden Bildteile im Curvature Space zu betrachten [8]. Beim Curvature Space handelt es sich um eine rotationsinvariante Repräsentation von Kurven. Die Konvertierung wandelt Punktefolgen in andere Punktefolgen um. Im zu

betrachtenden Bild werden Konturen extrahiert und dann in den Curvature Space konvertiert. Diese Kurven werden sozusagen „abgeleitet“, wodurch nicht mehr die Orientierung der Kurvenabschnitte im zweidimensionalen Raum betrachtet wird, sondern nur mehr die Winkeländerungen der Kurvenabschnitte den angrenzenden Abschnitten (Siehe Fig. 12 - 14). Die dabei entstehenden neuen Kurven werden nochmals neu gesampelt, und mit einem Parameter Sigma gefiltert. Das Filtern führt dazu, dass kleine Details der Kurven verschwinden, und mehr die grobe Form übrigbleibt, auch die Auswirkungen von Pixelrauschen werden damit entsprechend abgeschwächt (Siehe Fig. 15-17). Die Rotation der gesamten Kurve entspricht einer Verschiebung der neuen Kurve auf der x-Achse (durch die potenziell unterschiedliche Wahl des Startpunktes). Ähnliche Kurven (Also z.B. Deutgesten) ähneln sich auch im Curvature Space (siehe Fig. 6 und 7, größter Unterschied liegt nur in der Verschiebung des Plots auf der x-Achse).

Fig. 8 2D- Plot der Kurven 1 (grün), 2 (blau) und 3 (rot)

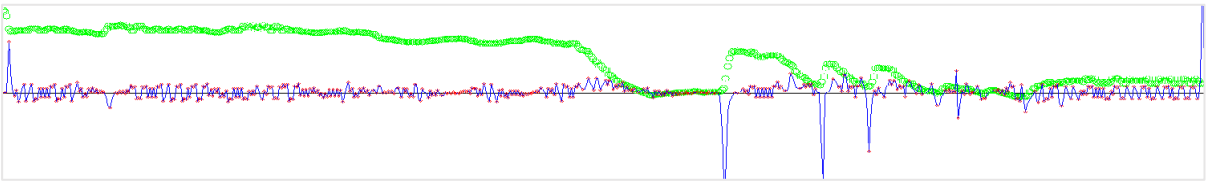


Fig. 9 CSS-Plot der Kurve 1 mit Sigma 1

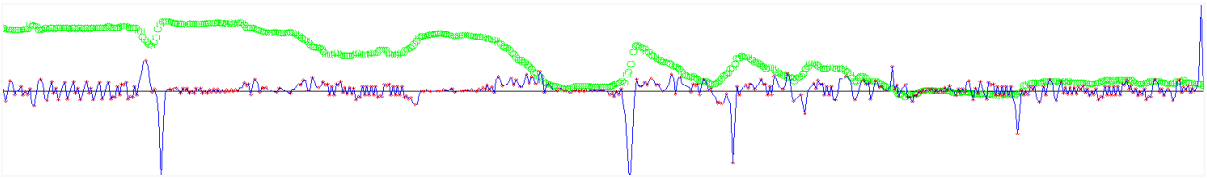


Fig. 10 CSS-Plot der Kurve 2 mit Sigma 1

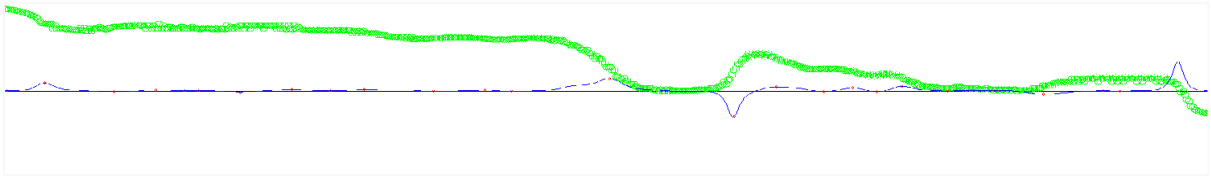


Fig. 11 CSS-Plot der Kurve 1 mit Sigma 9

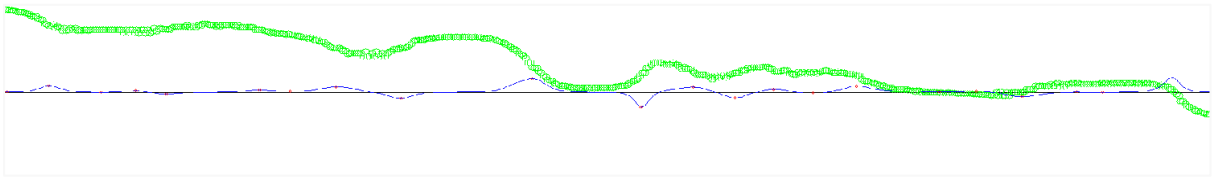


Fig. 12 CSS-Plot der Kurve 2 mit Sigma 9

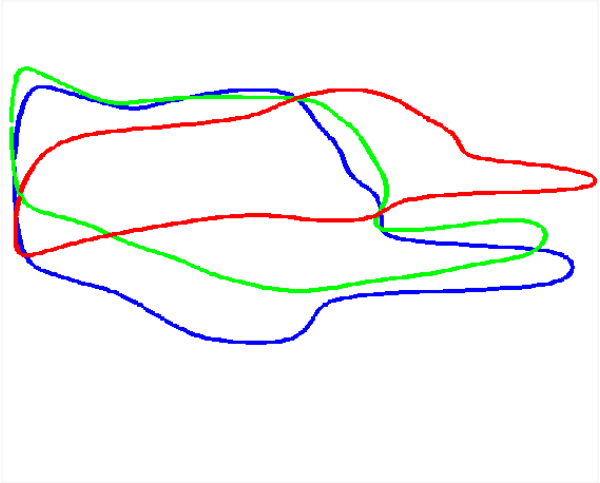


Fig. 13 2D-Plot der Kurven 1, 2 und 3, gefiltert mit Sigma 9

In der Referenzimplementierung wurde zum Testen die OpenCV-Implementierung findContours() zum Extrahieren der Konturen aus dem Ergebnis des Hintergrundsegmentierers, und eine bereits bestehende Implementierung [9] welche sich auf das Paper von Farzin Mokhtarian und Sadegh Abbasi [8] bezieht, zum Umwandeln der Konturen in den Curvature Space sowie zum Filtern der so entstandenen Kurven verwendet,

Nun ist die Rotation als Variable entfernt, und es muss nur mehr die Form der Kurve abgeglichen werden. Dafür gibt es wieder unterschiedliche algorithmische Möglichkeiten: Neuronale Netzwerke, oder andere lernende Algorithmen könnten zu Klassifizierung dieser Kurven verwendet werden. Diese bieten den Vorteil, dass keine weiteren Annahmen bezüglich der Kurven gemacht werden müssen, erfordern aber einen großen Satz an Trainingsdaten. Die Evaluation dieser Ansätze kann in

Folgearbeiten erfolgen. Stattdessen wird versucht, den Vergleich per Kreuzkorrelation durchzuführen. Als Vergleichskurven dienen dabei vorab berechnete Konturkurven, welche unterschiedliche Deutgesten darstellen. Um Kurven unterschiedlicher Größe per Kreuzkorrelation zu vergleichen, müssen beide Kurven unterschiedlich skaliert werden, da sich eine größere Konturkurve eine breitere CSS-Kurve ergibt, und die Kreuzkorrelation nur gut auf Kurven mit ähnlicher Skalierung in x-Richtung funktioniert. Deshalb werden für alle zu vergleichenden Kurven durch Resampling unterschiedlich skalierter Auskangskonturen und Filterung mit verschiedenen Sigma Signaturdatensätze mit jeweils mehreren hundert unterschiedlichen CSS-Kurven erstellt. Diese Kurven werden dann jeweils per Kreuzkorrelation verglichen, und die Übereinstimmung der Teilstücke mit der höchsten Korrelation als Treffer zurückgegeben. Dabei werden dann nur Treffer akzeptiert, welche über einem bestimmten Korrelationswert liegen, um false positives zu vermeiden.

In der Referenzimplementierung erwies sich diese Herangehensweise als ungeeignet: Zuerst einmal liefert der kreuzweise Vergleich der Kurvensignaturdatensätze eine quadratische Laufzeit, welche schon bei einer Kurvengröße von 100 (die Kurven bestehen aus jeweils 100 Punkten) schon Laufzeiten weit abseits der Echtzeit bedingte. Zusätzlich lieferte die Erkennung in der Referenzimplementierung weitgehend nur false positives, der Umriss des gesamten Körpers wurde häufig als Deutgeste nach oben erkannt. Dies ist auch auf zu ungenaue Parametrisierung der Signaturdatensätze zurückzuführen, die maximalen Skalierungsfaktoren und Sigmas waren zu groß gewählt. Dennoch wurde der Ansatz, kurvenbasierte Vergleiche per Kreuzkorrelation durchzuführen, zurückgestellt.

Pixelbasierte Erkennung

Pixelbasierte Erkennung wird schon lange im Feld der Computer Vision verwendet. Vor allem auch in der Stereo Vision, wo gleiche Bildausschnitte unterschiedlicher Kameras einander zugeordnet werden müssen, finden pixelbasierte Algorithmen Anwendung. Dabei ist das Grundprinzip immer sehr ähnlich: Zwei Bildausschnitte werden auf bestimme Charakteristika hin miteinander verglichen, und je nach Wert dieser entschieden, ob eine Übereinstimmung gefunden wird. Eine einfache Vergleichscharakteristik könnte dabei beispielsweise die Summer der quadrierten pixelweisen Differenzen sein (Least Squared Error). Natürlich ist das nicht sehr sinnvoll, diese Methode nimmt keine Rücksicht auf die Formen oder den Inhalt der zu vergleichenden Ausschnitte, und benötigt in diesem Fall auch viele unterschiedliche Vergleiche für mögliche unterschiedliche Ausartungen von Deutgesten. Natürlich gibt es aber bessere Methoden.

Die Technik im Bereich der Feature-Erkennung ist mittlerweile so fortgeschritten, dass selbst kleine Smartphones Gesichter in Echtzeit identifizieren und sogar erkennen können. Die dabei verwendeten Algorithmen wurden analysiert, und auf den Einsatz bei einem Deutgestenerkenner hin getestet.

Der größte Unterschied zwischen der Gesichtserkennung und der Deutgestenerkennung liegt darin, dass zu erkennende Gesichter auf Bildern meist die gleiche Orientierung haben, einerseits zeigen die Gesichter hin zur Kamera, und andererseits stehen sie anatomiebedingt aufrecht. Deutgesten sind in der Orientierung beliebig und haben damit einen deutlichen Nachteil bezogen auf pixelbasierte Verfahren.

Generell sind die pixelbasierten Verfahren sehr anfällig für unterschiedliche Rotation zwischen Bildern. Eine Möglichkeit, dieses Problem zu umgehen ist, nach dem Greedy-Verfahren verschiedene Rotationen zu testen, und diese dann hin zu lokalen Optima zu verfeinern. Alternativ muss auf eine andere Art und Weise sichergestellt werden, dass die zu vergleichenden Bildinhalte die gleiche Ausrichtung haben.

In der Referenzimplementierung wurde dafür auf die schon oben aufgeführten Annahmen zurückgegriffen, dass Deutgesten auf den Bildern fast nur in horizontale Richtung verlaufen. Die Ausschnitte, die der Schritt der Filterung nach dünnen horizontalen Strukturen zurückgibt, enthalten im Idealfall nur noch Arme mit ausgestreckten Fingern. Um die Richtung dieses (ausgestreckten) Armes zu gewinnen, reicht es im Idealfall jeweils einen Punkt auf beiden Seiten des Bildes zu finden, welcher auf dem Arm liegt. Im Falle der Schwarz-Weiß-Maske ist dies relativ einfach, es werden einfach die Centroids (Mittelpunkte) der Vordergrundpunkte des rechten und linken äußeren Bildausschnittes berechnet. Sofern es sich bei dem Inhalt des Bildabschnittes um einen Arm handelt, und nicht zu viel Rauschen vorliegt, werden diese beiden Punkte dann auf der Hand/ auf dem Arm der Deutgeste liegen. Sofern der Arm ausgestreckt ist, bildet die Linie $\overline{v}'$, welche die beiden berechneten Punkte verbindet, genau $\overline{v}$. Mithilfe von $\overline{v}'$ kann nun genauer untersucht werden, ob es sich bei der länglichen Struktur um eine Deutgeste handelt. Dafür wir nun ein Bildausschnitt, welcher durch $\overline{v}'$ und eine Breite b bestimmt wird, gedreht, sodass darauf nun pixelbasierte Erkennung durchgeführt werden kann.

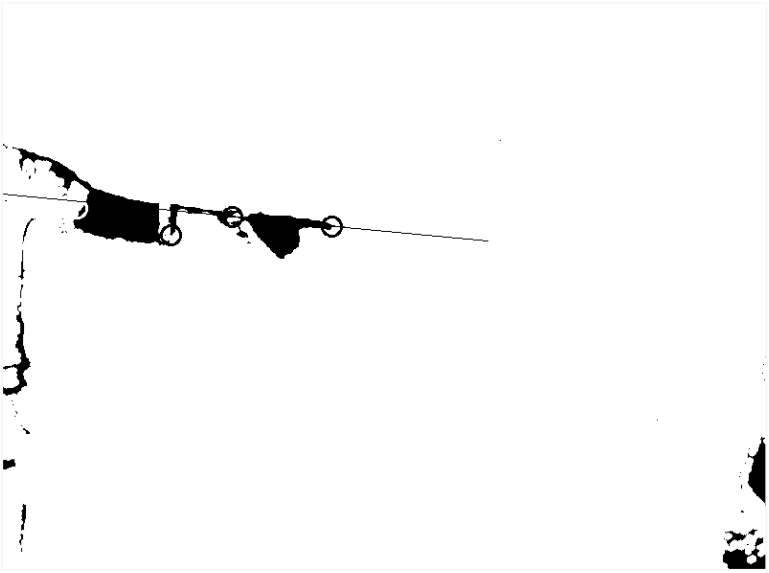


Fig. 15 Schätzung von $\vec{v}'$ mithilfe der zwei Hilfspunkte

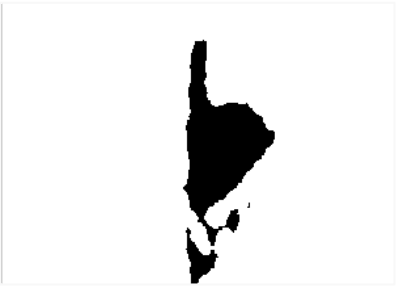


Fig. 15 Daraus generierter Bildausschnitt, rotiert um weiter untersucht zu werden

Dlib HOG Object Detector

Hier wurde der Simple Object Detector der dlib verwendet. Dieser basiert intern auf dem Histogram of Oriented Gradients Verfahren [10], welches basierend auf einer Auswahl von Trainingsbildern ein Histogramm erstellt, welches markante Verlaufsänderungen innerhalb des Bildes codiert. Mithilfe dieses Histogramms kann dann für einen gegebenen Bildbereich schnell bestimmt werden, ob dieser die durch die Trainingsbilder vorgegebenen Merkmale besitzt (ob er den Trainingsbildern ähnelt). Im Falle von Gesichtern wird der Algorithmus mit einer großen Anzahl von Gesichtern enthaltenden Bildern trainiert. Dabei wird für jedes Bild markiert, wo sich ein zu erkennender Bereich (im Falle von Gesichtern eben ein Gesicht) befindet. Mithilfe einer Optimierung wird dann das Histogramm so erstellt und parametrisiert, dass alle Trainingsbilder mit einer festgelegten Wahrscheinlichkeit und Genauigkeit wiedererkannt, gleichzeitig aber in zufällig ausgewählten Negativtestbildern (in diesem Fall ohne Gesichter) keine Übereinstimmung erkannt wird.

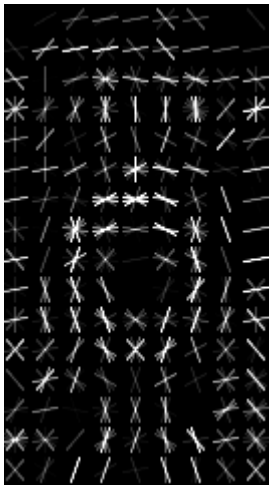


Fig. 16 Visualisierung des Histogramms zur Erkennung von Deutgesten

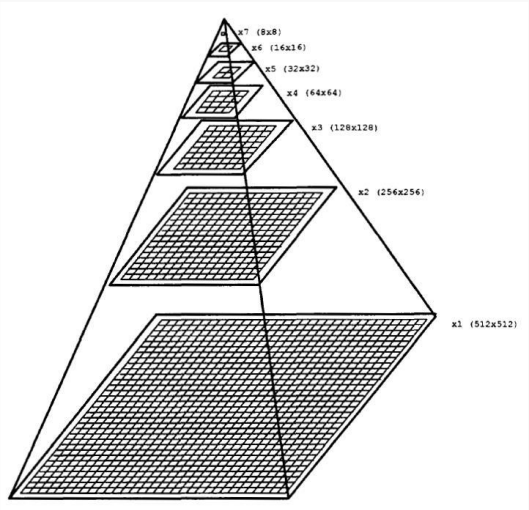


Fig. 17 Möglicher Aufbau einer Bildpyramide [14]

Um für ein gegebenes Bild mithilfe des Histogramms festzustellen, ob z.B. ein Gesicht enthalten ist, muss dieser entsprechende Bildausschnitt gegen das Histogramm verglichen werden, es reicht nicht, einfach das ganze Bild mit dem Histogramm abzugleichen. Um den passenden Bildausschnitt zu bestimmen, wird ein das Histogramm auf eine Bildpyramide angewandt: Zum zu durchsuchenden Bild wird eine Bildpyramide aufgebaut, auf jedem Level wird die Auflösung des Bildes hin bis zu einem Minimum verkleinert. Das Histogramm wird zuerst über die niedrig auflösenden Teile der Bildpyramide angewendet, und dabei jeweils über das Bild geschoben. Falls eine Übereinstimmung auf einem der niedrigeren Level auftritt, wird diese auf den höher auflösenden Levels weiter untersucht. So kann das ganze Bild mit relativ geringem Aufwand komplett durchsucht werden. Zusätzlich können noch Pyramidenlevels mit einer höheren Auflösung als das Ursprungsbild generiert und durchsucht werden, um Features zu erkennen, welche im Originalbild kleiner als das Histogramm sind.

In der Referenzimplementierung wurde der Object Detector der dlib mithilfe eines Teils der Trainingsdaten eines anderen Datensets zur Gestenerkennung trainiert. [11] Dabei waren alle Bilder aus realen Fotos extrahierte Masken von Deutgesten, die alle nach oben zeigen. Für das Training wurden dann alle Bilder noch einmal horizontal gespiegelt, sodass Deutgesten beider Arme erkannt werden können. Dies birgt natürlich den Nachteil, dass eine Erkennung dann nicht mehr so zuverlässig wird, da das zu erkennende nicht mehr so eindeutig definiert ist. Die Alternative dazu ist, ein Histogramm nur mithilfe Deutgesten einer Richtung (z.B. nur rechte Hand) zu generieren und eines nur mit mithilfe der gespiegelten Bilder (oder einfach das berechnete Histogramm zu spiegeln), und dann immer die Erkennung für beide Richtungen parallel laufen zu lassen. Dies wiederum würde die Qualität der Erkennung verbessern, aber auch doppelt so viel Zeit und damit Performance in Anspruch nehmen. Ein positiver Nebeneffekt wäre, dass die Deutgesten dann eindeutig einem rechten oder linken Arm zugeordnet werden könnten.

In der Referenzimplementierung wurde auf erstere Methode zurückgegriffen und ein Symmetrisches Histogramm erzeugt, um die Performance zu verbessern. Im Folgenden die Bilder, die für das

Training verwendet wurden:

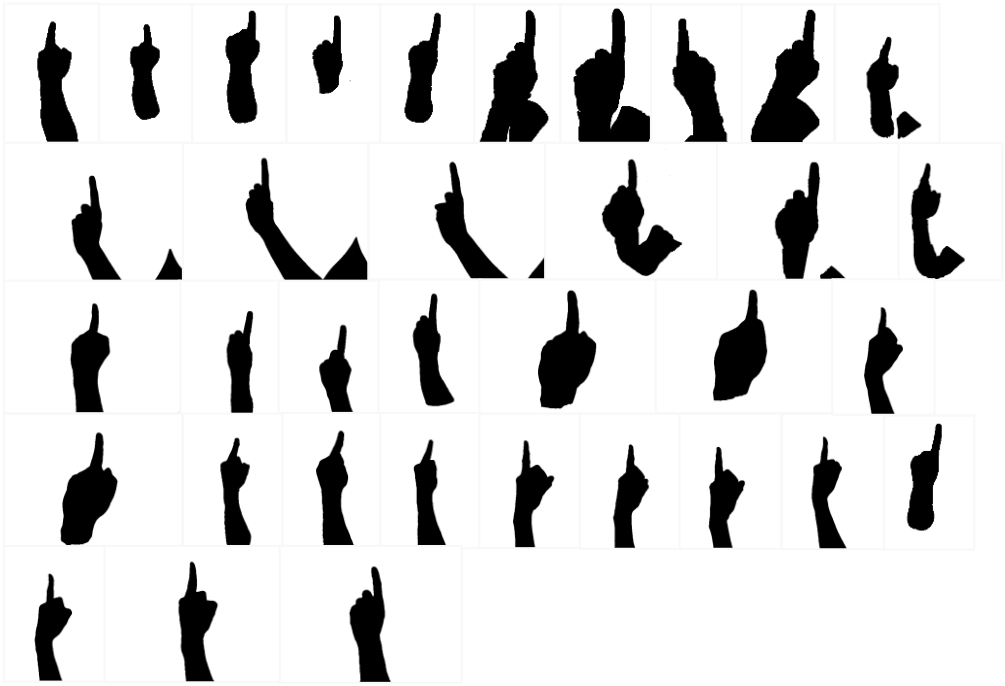


Fig. 18 Die zum Training des Object Detectors verwendeten Rohbilder

Zwischenfazit

Durch die gewählte Kombination aus Vorder-Hintergrund Segmentierung, Filterung der Zusammenhangskomponenten nach langen, dünnen, horizontalen Bereichen, grober Richtungsschätzung dieser und anschließender HOG-basierter Überprüfung der vermuteten Arme werden unter optimalen Voraussetzungen aus einem Bildstrom nun für jedes Bild die enthaltenen Deutgesten sowie deren Richtung mit relativ hoher Wahrscheinlichkeit und Genauigkeit erkannt. Dies geschieht auch der emuliert schwachen Hardware noch in Echtzeit, mit mehreren Berechnungen pro Sekunde.

Um eine Abdeckung aller möglichen Deutrichtungen (außer direkt nach oben) zu erreichen, werden die Bilder von mehreren Kameras verwendet. Es besteht die Möglichkeit, die beiden Kameraperspektiven per Kalibrierung in Perspektive zueinander zu setzen, und Deutgesten, sofern sie von beiden Kameras erfasst werden, zurück in den dreidimensionalen Raum zu mappen. Dies hat allerdings den Nachteil, dass dann die Bildverarbeitungs- und Deutgestenerkennungsprozesse der beiden Kameras nicht mehr unabhängig voneinander wären, und sich damit auch nicht mehr so

einfach parallelisieren lassen würden. Die Parallelisierung als Optimierung wird später noch einmal aufgegriffen.

Jedoch sind auch jetzt schon einige Faktoren bekannt, welche die Erkennung von Deutgesten mit der vorgeschlagenen Vorgehensweise erschweren oder unmöglich machen:

Die Hintergrundsegmentierung, auf der alle folgenden Schritte beruhen, ist sehr anfällig, wenn sich große Bereiche im Bild, vor allem hinter den Benutzern, verändern. Dies passiert allerdings immer, wenn sich Lichtverhältnisse ändern. Dann braucht der Algorithmus erst wieder ein paar Sekunden, bis er sein Hintergrundmodell angepasst hat, und den Benutzer wieder vom Hintergrund unterscheiden kann. Solange eine Unterscheidung nicht möglich ist, ist anstatt der Kontur des Benutzers auf der Maske nur ein großer Zusammenhängender Bereich erkennbar, und damit können auch keine potenziellen ausgestreckten Arme erkannt werden. Dies birgt in einer Umgebung, in der unter anderem Lichter per Deutgeste gesteuert werden können sollen, einen großen Nachteil. Des Weiteren wird ein Benutzer, wenn er lange genug stillsteht, oder eine Deutgeste lange ausführt, irgendwann ins Hintergrundmodell übernommen, sodass diese Deutgeste auch nicht mehr erkannt werden kann.

Wenn die Hintergrundsegmentierung zu großes Rauschen produziert, werden bei der Berechnung der äußeren Armpunkte falsche Pixel mit in die Berechnung einbezogen, und die Position der Centroids und die Richtung der Deutgeste damit verfälscht. Sich bewegende Hintergründe wie Bäume im Wind, Fernseher, Ventilatoren oder Vorhänge äußern sich ähnlich.

Bedingt durch die getroffenen Annahmen können mit diesem Algorithmus auch keine vertikalen Deutgesten erkannt werden. Ein bei einer Deutgeste angewinkelter Arm führt auch dazu, dass die erkannte Deutrichtung nicht der tatsächlichen Richtung entspricht.

Ein weiteres Problem, was nach dem Testen deutlich wurde ist, dass das System nicht sicher entscheiden kann, in welche Richtung $\overline{v}$ weist. Dies artet darin aus, dass im schlimmsten Fall Aktuatoren getriggert werden, die zwar auf $\overline{v}$ liegen, aber auf der anderen Seite des Benutzers.

Da der Simple Object Detector der dlib auf die Vordergrundmaske, ein Schwarz-Weiß-Bild angewendet wird, entfaltet dieser nicht sein volles Potenzial, und liefert verhältnismäßig viele false positives. Da im Schwarz-Weiß-Bild keine Informationen mehr z.B. über die Beschaffenheit und die Farbe der Haut des Benutzers kodiert sind, sind diese bei der Erkennung mit diesem Algorithmus

irrelevant, also auch mit grünen oder bunt gesprenkelten Händen würden Deutgesten theoretisch erkannt werden. Dies birgt jedoch den Nachteil, andere false positives nicht ausgeschlossen werden können.

Interpretation der Deutgesten

Um mit den erkannten Deutgesten und deren Richtungen intelligente Umgebungen zu steuern, ist noch eine Zuordnung einer Deutgeste zu einer bestimmten Aktion notwendig. Um die Steuerung intuitiv zu gestalten, müssen erkannte Deutgesten diejenigen Aktuatoren ansteuern, welche auch das Ziel der Deutgeste sind. Anders formuliert: Wenn der Benutzer auf eine Lampe in der Ecke zeigt, soll auch diese Lampe aktiviert/deaktiviert werden. Gegeben durch die Funktionsweise einer Kamera ist auch die Tatsache, dass ein Objekt T, was $\overline{v}$ im dreidimensionalen Raum schneidet, auch auf der Projektion auf das zweidimensionale Kamerabild $\overline{v}$ schneidet. Dabei ist es egal, ob das Objekt auf dem Kamerabild sichtbar ist oder nicht. Deshalb werden jedem Aktuator i pro Kamera c Triggerkoordinaten T_i^c im zweidimensionalen Raum zugewiesen, mit welchen dann mithilfe der Gleichungen 1) bis 6) überprüft wird, ob dieser Aktuator Ziel der gerade erkannten Deutgeste ist. Anstatt einem Paar von Triggerkoordinaten pro Bild kann auch gleich ein Triggerbereich $\sum T_i^c$ für jeden Aktuator zugewiesen werden, das Prinzip bleibt gleich. Mit einer Bereichsrepräsentation können aber voluminösere Aktuatoren wie Fernseher, oder größere Lichter besser repräsentiert werden. Alternativ wäre es auch möglich, die von mehreren Kameraperspektiven erkannten Deutgesten sowie die Aktuatoren im dreidimensionalen Raum zu betrachten.

In der Referenzimplementierung wurde testweise die einfachere Repräsentierung mit nur einem Paar Koordinaten im zweidimensionalen Raum umgesetzt.

Um die Zuverlässigkeit der Erkennung zu erhöhen und fälschliche (De)Aktivierungen zu vermeiden, muss eine Deutgeste erst für eine bestimmte Zeit gehalten werden, bevor das Auslösen einer Aktion erfolgt. Um Doppel-(De)Aktivierungen zu verhindern muss die Deutgeste erst wieder beendet werden, bevor derselbe Aktuator nochmals (de)aktiviert werden kann.

In der Referenzimplementierung wurde dafür ein Probabilistisches Modell implementiert pro Aktuator eine Triggerwahrscheinlichkeit eingeführt, welcher pro Zeiteinheit in welcher der Aktuator Ziel einer Deutgeste ist, proportional zur Wahrscheinlichkeit mit der er Ziel der Deutgeste ist erhöht wird. Überschreitet dieser Wert ein bestimmtes Level, so wird der Aktuator getriggert.

Erkennung und Zuordnung der Aktuatoren

Bisher wurden Koordinaten T_i^c der Aktuatoren i für jede Kameraposition c per Hand vergeben, jedoch ist an dieser Stelle eine Automatisierung möglich, unter der Annahme, dass das Triggern des Aktuators deutlich in allen Kamerabildern zu sehen ist, und gleichzeitig der Bereich mit der größten Veränderung nach dem Triggern auch das Ziel einer Deutgeste für diesen Aktuator ist. Dies ist z.B. bei Lichtern, Fernsehern Jalousien oder ähnlichen Geräten im Allgemeinen so der Fall. Um also die Koordinaten T_i^c oder Bereiche zu bestimmen, welche die Aktuatoren im zweidimensionalen Raum einnehmen reich es grob gesagt, pro Kameraperspektive die Differenz aus einem Bild vor der Aktivierung und einem Bild nach der Aktivierung eines Aktuators zu bilden, die Bereiche mit der größten Veränderung werden dann als neuer Triggerbereich für den jeweiligen Aktuator verwendet. Wenn stattdessen Triggerkoordinaten verwendet werden sollen, wird beispielsweise einfach der Mittelpunkt des Bereichs berechnet. Dieses Verfahren kann, sofern der Aktuator in den Kamerabildern zu sehen ist, offline, also ohne Zutun des Benutzers geschehen. Sollte ein Aktuator nicht auf dem Kamerabild erkennbar sein, so kann diese manuell gesetzt werden. Alternativ lassen sich die Triggerkoordinaten jedes Aktuators auch durch mehrmaliges Durchführen von Deutgesten darauf von unterschiedlichen Positionen im Raum interpolieren. Dafür wird der wahrscheinlichste Schnittpunkt aller $\vec{v}_i^c$ berechnet. Um ein genaueres Endergebnis zu erhalten, ist ein Durchführen der Deutgesten an unterschiedlichen Positionen im Raum vorteilhaft.

Anbindung der Aktuatoren

Um die Aktuatoren anzusteuern, wurde auf das von Florian Bayer im Rahmen seiner Bachelorarbeit entwickelte Interface zurückgegriffen. Das Interface ermöglicht das einfache Ansteuern von Aktuatoren unterschiedlichster Art mithilfe von einstellbaren Triggern. Die Kommunikation erfolgt per Token gesichertem (aber unverschlüsseltem) URL-Aufruf. Mithilfe des Interfaces können Phillips Hue Lichter an- und ausgeschaltet, die Farbe und -intensität eingestellt, sowie gedimmt werden. Zusätzlich ist das Interface einfach um weitere Smart Actors erweiterbar. Die unsichere Kommunikation ist insoweit unproblematisch, als dass beide Programme auf derselben, vom Netzwerk isolierten Maschine laufen.

Anbindung zum Voice Interface

Florian Bayer's Interface bietet auch die Anbindung an Amazon's Sprachassistenten Alexa. Damit können Deutgesten und Sprachbefehle zur Steuerung von Intelligenten Umgebungen kombiniert

werden. Die Deutgeste agiert dabei immer als räumliche Selektion und Trigger eines Aktuators, die Sprachsteuerung spezifiziert gegebenenfalls die Aktionen, welche mit dem gewählten Aktuator möglich sind, erfasst dann, welche Aktion der Benutzer letztendlich ausgewählt hat, und führt diese aus. Ein konkretes Beispiel könnte sein: Ein Benutzer deutet auf den ausgeschalteten Fernseher im Raum, Alexa fragt, auf welchen Kanal der Fernseher eingeschaltet werden soll. Der Benutzer nennt den Kanal, der Fernseher wird ein- und auf den genannten Kanal geschaltet. Ein weiteres Beispiel: Der Benutzer deutet auf eine eingeschaltete Lampe, Alexa fragt, ob die Lampe ausgeschaltet werden soll, oder die Farbe oder Intensität geändert werden soll. Der Benutzer antwortet, und die Lampe reagiert entsprechend.

Anwendungsmöglichkeiten und Ausblick

Neben den bereits genannten Möglichkeiten, Deutgesten im Kontext von intelligenten Umgebungen zu verwenden bieten sich noch viele weitere, welche aber nicht im Rahmen dieser Arbeit evaluiert werden.

Denkbar ist beispielsweise eine Unterscheidung von unterschiedlichen Deutgesten. Dem Deuten mit zwei Fingern könnte eine andere Bedeutung zukommen, als dem Deuten mit einem Finger. Möglicherweise würde das Zeigen mit zwei Fingern dann ein Aufzählen von kontextabhängigen Optionen für den gewählten Aktuator durch den Sprachassistenten auslösen, während das Zeigen mit nur einem Finger diesen Aktuator nur (de)aktiviert. Auch eine Unterscheidung von weiteren Deutgesten ist denkbar.

Andererseits wären auch immaterielle Aktuatoren denkbar: Deuten auf ein Fenster könnte eine Wettervorhersage durch den Sprachassistenten auslösen. Natürlich können Deutgesten auch als komplett konfigurierbarer Trigger fungieren. Vom Nutzer selbst definierte Gesten auf vom Nutzer definierte Punkte im Raum können theoretisch jede denkbare Art von Interaktion auslösen.

Mögliche Optimierungen und Verbesserungen

Um die Performance, sowie die Robustheit der Erkennung zu verbessern, können einige Verbesserungen und Optimierungen vorgenommen werden, welche im Rahmen dieser Arbeit nicht aus Zeit- oder Technikgründen nicht in Betracht gezogen wurden.

Wie bereits erwähnt können einige Teile der Erkennung parallelisiert werden: Die Bildströme der Kameras können unabhängig voneinander bearbeitet werden, und erlauben so eine bessere

Ausnutzung der gegebenen Hardware, und ermöglichen im Vergleich zur sequenziellen Bearbeitung, wie sie jetzt implementiert ist, eine Beschleunigung von bis zu 100%.

Die Bearbeitung der Einzelbilder kann durch Pipelining ebenfalls stark beschleunigt werden: Zwar wird die Zeit von der Aufnahme des Bildes bis zum Erkennen der Deutgeste nicht beeinflusst, es können jedoch je nach Anzahl der Prozessoren mehrere Kerne auf die Bearbeitung eines Einzelbildes verwendet werden. Die rechenaufwändige Hintergrund-Segmentierung, sowie die pixelbasierte Erkennung eignen sich als Flaschenhälse des Programmes sehr gut zum Pipelining. Ein Kern beschäftigt sich nur mit der Hintergrund-Segmentierung und der Vorselektion, und gibt das Ergebnis einer Berechnung dann weiter an den zweiten Kern, welcher sich um die pixelbasierte Erkennung kümmert. Auf dem Zielsystem, welches 4 Kerne zur Verfügung stellt, können damit (gegeben, es wurden bereits Aufnahmen von 2 Kameras parallel bearbeitet) wieder Beschleunigungen von theoretisch bis zu 100% erreicht werden (realistischerweise allerdings weniger).

Neben der Performance kann auch die Erkennungsgenauigkeit optimiert werden. Wenn der pixelbasierte Erkenner statt auf der Schwarz-Weiß-Maske auf dem echten Kamerabild eingesetzt wird, werden fallen Fehler, welche in den vorhergegangenen Schritten aufgetreten sind (z.B. Rauschen bei der Hintergrundsegmentierung, Verlust von Genauigkeit durch die morphologische Öffnung) nicht mehr so stark ins Gewicht. Dafür müsste aber der Erkenner mit anderen Bilddaten trainiert werden.

Anstatt der horizontalen spaltenweisen Summation, welche zur Eingrenzung des durchsuchenden Bereichs verwendet wird, können komplexere Algorithmen verwendet werden, um die Auftrittswahrscheinlichkeit von false negatives zu verkleinern. Anstatt nur horizontal zu suchen, könnte auch für jeden Zusammenhangskomponenten radial von außen nach innen gesucht werden. Alternativ kann auch ein modifiziertes Skeletonizing zurückgegriffen werden, welches beim iterativen Berechnen nur Skelettlinien anzeigt, wenn eine bestimmte Anzahl von Erosionsiterationen nicht überschritten wurde. Dies würde ebenfalls das Ergebnis auf Bereiche mit einer maximalen Dicke beschränken, und für diese dann die Skelettlinien berechnen.

Mithilfe einer GPU könnten einige der Berechnungen stark beschleunigt werden. Vor allem die pixelbasierte Erkennung, die Hintergrundsegmentierung sowie alle morphologischen Operationen könnten durch den Einsatz einer Grafikkarte parallelisiert werden.

Evaluation von Deutgesten zur Steuerung von Intelligenten Umgebungen

ZIELE DER EVALUATION

Deutgesten sollen im Hinblick auf Ihre Praktikabilität, Intuitivität und Akzeptanz im Kontext der Steuerung von intelligenten Umgebungen evaluiert werden. Besonders interessant sind dabei folgende Fragestellungen:

- Können intelligente Umgebungen mit Deutgesten effektiv gesteuert werden?
- Können intelligente Umgebungen mit dem im Rahmen dieser Arbeit entwickelten System effektiv gesteuert werden?
- Wie effektiv können intelligente Umgebungen mit Deutgesten im Vergleich zu etablierten Technologien wie Fernbedienungen oder Sprachassistenten gesteuert werden?
- Wie praktikabel sind Kombinationen aus Deutgesten mit etablierten Technologien zur Steuerung intelligenter Umgebungen?
- Wie einfach und intuitiv ist die Steuerung von intelligenten Umgebungen per Deutgeste?
- Wie einfach und intuitiv ist die Steuerung von intelligenten Umgebungen mithilfe von kombinierter Deutgestenerkennung und Sprachsteuerung?
- Wird kamerabasierte Deutgestenerkennung von potenziellen Benutzern akzeptiert?

AUFBAU DER EVALUATION

Um die Eignung von Deutgesten zur Steuerung von intelligenten Umgebungen zu evaluieren, wurde eine beispielhafte intelligente Umgebung aufgebaut. Diese beinhaltete mehrere vernetzte intelligente Aktuatoren und Sensoren:

Aktuatoren:

1. Lichtsäule, betrieben mit vier Phillips Hue RGB LEDs
2. Tischlampe, betrieben mit einer Phillips Hue LED
3. Bettlampe, betrieben mit einer Phillips Hue Iris
4. Fernseher mit Amazon Fire TV, gesteuert vom Logitech Harmony Hub

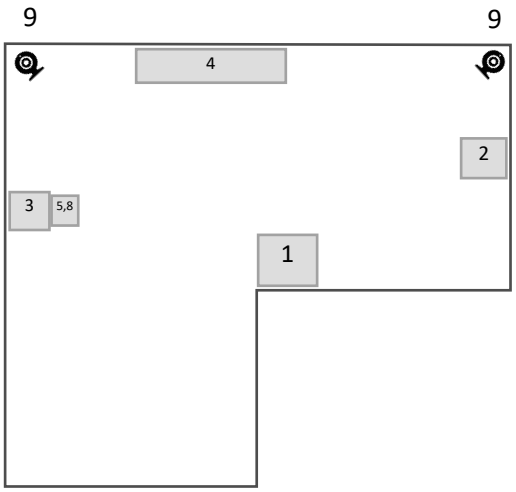


Fig. 19 Schematischer Aufbau der Testumgebung

5. Sprachausgabe durch Amazon Alexa

Sensoren:

6. Fernbedienung für Fernseher und Fire TV
7. Lichtschalter für Lichtsäule und Tischlampe
8. Spracherkennung durch Amazon Alexa
9. Kameras für Deutgestenerkennung

Die Aktuatoren wurden dabei im Raum so angeordnet, dass jeweils eine eindeutige Zuordnung einer Deutgeste möglich ist.

Die Kameras für die Deutgestenerkennung wurden an unterschiedlichen, nicht entgegenliegenden Ecken des Raumes unter der Decke platziert, sodass Deutgesten in Richtung einer Kamera, die perspektivbedingt nicht erkannt werden können, von der jeweils anderen Kamera erfasst werden können.

DURCHFÜHRUNG DER EVALUATION

Die Teilnehmer der Evaluation waren zwischen 19 und 28 Jahre alt, und unterschieden sich teils stark im Hinblick auf ihr Vorwissen und den Erfahrungen mit den beim Test verwendeten Technologien. Einige Teilnehmer hatten schon viel Erfahrung mit Sprachassistenten gesammelt, andere nicht. Mit allen Teilnehmern der Evaluation wurde jeweils dasselbe Szenario auf unterschiedliche Arten durchgespielt: Der Teilnehmer sollte alle Geräte (Bettlampe, Lichtsäule, Tischlampe, Fernseher) im Raum einmal ein, und dann wieder ausschalten, in beliebiger Reihenfolge, auf 4 unterschiedliche Arten:

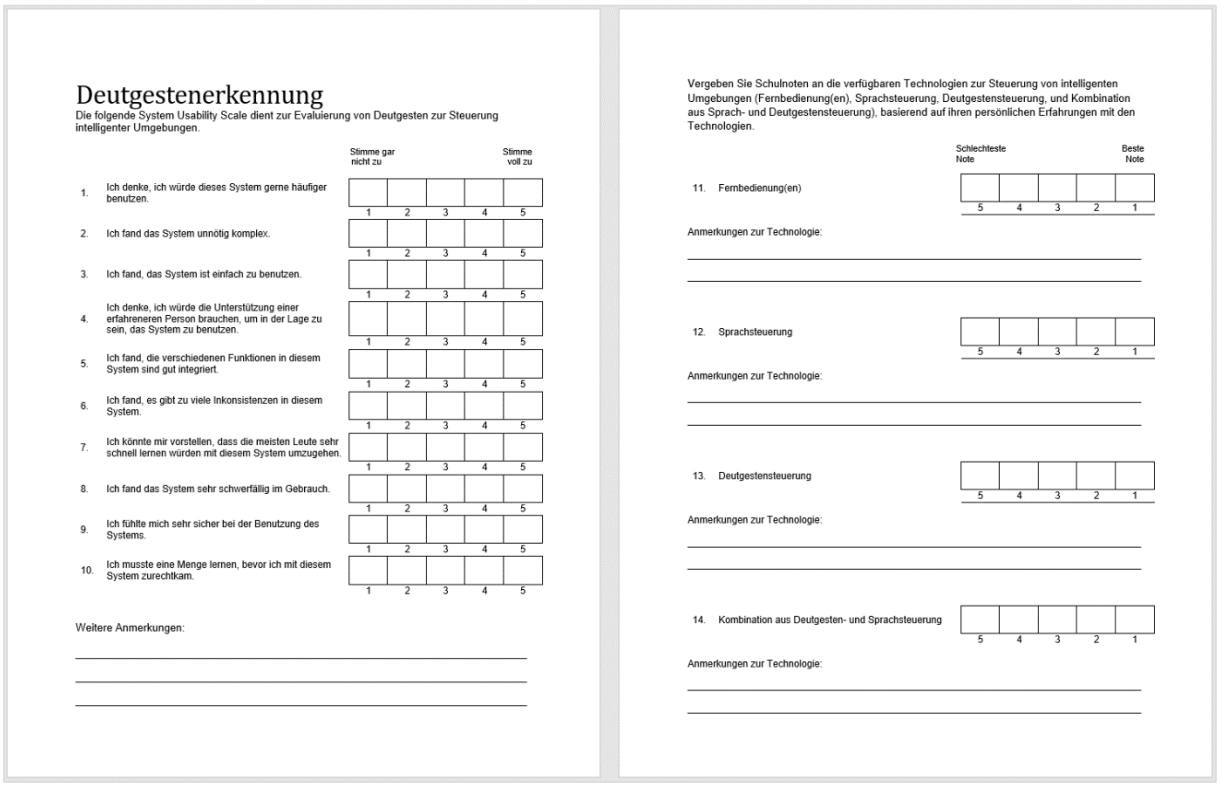
1. Mit Fernbedienungen und den Schaltern der Lampen
2. Mithilfe von Sprachsteuerung
3. Mithilfe von Deutgestensteuerung
4. Mithilfe von Kombinierter Deutgesten- und Sprachsteuerung

Die Reihenfolge der Art und Weise die Aktuatoren an- und wieder ausgeschaltet werden sollten, wurde dabei auch zwischen den Probanden durchgewechselt.

Währenddessen wurden die Zeiten gemessen, welche die Probanden je für einen Durchlauf An- und Ausschalten der Sequenz benötigten.

Um neben Deutgesten als Steuerungskonzept auch den implementierten Deutgestenerkenner zu evaluieren, wurde bei einem Nichtansprechen der Deutgestenerkennung die von Probanden gewünschte Aktion manuell aktiviert, diese false negatives dann aber notiert und damit die Zuverlässigkeit der Implementierung bewertet.

Im Anschluss wurden die Eindrücke und Meinungen der Teilnehmer dann mit einem Fragebogen sowie persönlichen Gesprächen eingeholt:



Deutgestenerkennung

Die folgende System Usability Scale dient zur Evaluierung von Deutgesten zur Steuerung intelligenter Umgebungen.

		Stimme gar nicht zu				Stimme voll zu
1.	Ich denke, ich würde dieses System gerne häufiger benutzen.	1	2	3	4	5
2.	Ich fand das System unnötig komplex.	1	2	3	4	5
3.	Ich fand, das System ist einfach zu benutzen.	1	2	3	4	5
4.	Ich denke, ich würde die Unterstützung einer erfahreneren Person brauchen, um in der Lage zu sein, das System zu benutzen.	1	2	3	4	5
5.	Ich fand, die verschiedenen Funktionen in diesem System sind gut integriert.	1	2	3	4	5
6.	Ich fand, es gibt zu viele Inkonsistenzen in diesem System.	1	2	3	4	5
7.	Ich könnte mir vorstellen, dass die meisten Leute sehr schnell lernen würden mit diesem System umzugehen.	1	2	3	4	5
8.	Ich fand das System sehr schwerfällig im Gebrauch.	1	2	3	4	5
9.	Ich fühlte mich sehr sicher bei der Benutzung des Systems.	1	2	3	4	5
10.	Ich musste eine Menge lernen, bevor ich mit diesem System zurechtkam.	1	2	3	4	5

Weitere Anmerkungen:

Vergeben Sie Schulnoten an die verfügbaren Technologien zur Steuerung von intelligenten Umgebungen (Fernbedienung(en), Sprachsteuerung, Deutgestensteuerung, und Kombination aus Sprach- und Deutgestensteuerung), basierend auf ihren persönlichen Erfahrungen mit den Technologien.

		Schlechteste Note				Beste Note
11.	Fernbedienung(en)	5	4	3	2	1
12.	Sprachsteuerung	5	4	3	2	1
13.	Deutgestensteuerung	5	4	3	2	1
14.	Kombination aus Deutgesten- und Sprachsteuerung	5	4	3	2	1

Anmerkungen zur Technologie:

Fig. 20 Fragebogen

ERGEBNISSE DER EVALUATION

Die Ergebnisse der Probanden unterschieden sich untereinander stark, eine Tendenz ist dennoch zu erkennen.

Ein Durchlauf An- oder Ausschalten mithilfe von Fernbedienungen und den Schaltern der Lampe benötigte durchschnittlich ca. 32 Sekunden. Dies ist zurückzuführen darauf, dass die Probanden

jeweils von Lampe zu Lampe gehen, und die Schalter betätigen mussten. Die Sprachsteuerung hingegen benötigte nur 27 Sekunden. Dabei waren einige Tester deutlich schneller, weil der Sprachassistent das Potenzial hatte alle Lichter mit einem einzigen Kommando auf einmal an- oder auszuschalten. Erschwerend für die Sprachsteuerung kam jedoch hinzu, dass für das Aus- und Anschalten des Fernsehers bedingt durch die Funktionsweise von Amazon's Alexa eine kompliziertere Syntax notwendig war („Alexa, schalte den Fernseher an mit Harmony"). Diese bereitete aber keinem der Tester eine sonderlich große Schwierigkeit. Für einen kompletten Durchlauf mit Deutgesten ohne zusätzliche Sprachsteuerung benötigten die Probanden nur 25 Sekunden. Die Kombination aus Deutgesten- und Sprachsteuerung schnitt hingegen mit 35 Sekunden am schlechtesten ab. Dies ist zurückzuführen darauf, dass aufgrund der gewählten Implementierung eine erkannte Deutgeste hier erst noch den Umweg über die Sprachsteuerung, und damit in die Amazon Cloud machen muss, bevor dann noch eine verbale Bestätigung an den Benutzer zurückgemeldet wird, oder kontextsensitiv noch eine Frage zur Aktion gestellt wird („Welche Farbe soll <Lampe> haben?"). Erst danach werden die Aktuatoren aus- oder angeschaltet.

Die Erkennungsgenauigkeit des Deutgestenerkenners lag bei nur 20%, das heißt die 80% der Aktionen wurden manuell ausgelöst, weil der Deutgestenerkenner sie nicht rechtzeitig erkennen konnte. Fast nur eine Deutgeste auf den Fernseher wurde erkannt. Dies hat mehrere Gründe: Hauptverantwortlich ist der Testaufbau: Durch die Anordnung der Kameras und die Verlängerung per USB wurde auf Kamera 2 nur eine Auflösung von 256 × 144p erreicht. Damit war die Erkennungsrate auf dieser Kamera gleich 0%. Kamera 1 lief auf der gewünschten Auflösung von 800 × 600p, konnte Deutgesten aber dennoch nur bis zur Entfernung von ca. 2 Metern erkennen, alles dahinter wurde aufgrund zu niedriger Auflösung auch nicht mehr erkannt. Nur die Deutgeste auf den Fernseher erfüllte die Bedingung, von Kamera 1 erkannt werden zu können, und im richtigen Abstand von der Kamera stattzufinden.

Die System Usability Scale wurde für das Konzept der Deutgesten auf 80 Punkte ausgewertet, dies entspricht einem „guten" Wert.

Die Probanden bewerteten die Fernbedienung mit einer 2 als „gut", die Sprachsteuerung mit einer 2,5 als „gerade noch gut", die Deutgesten mit einer 2 ebenfalls als „gut", die Kombination aus Sprachsteuerung und Deutgesten mit einer 2,5 als „gerade noch gut".

INTERPRETATION DER ERGEBNISSE

Die Anzahl der Tester war nicht groß genug, um allgemeingültige Schlüsse ziehen zu können. Dennoch deuten die Ergebnisse darauf hin, dass intelligente Umgebungen mit Deutgesten effektiv gesteuert werden können. Im Vergleich aller getesteten Bedienungsarten schnitten die Deutgesten in Anbetracht der Zeit, die für die Bedienung benötigt wird, im Durchschnitt mit am besten ab. Erfahrene Nutzer waren jedoch mit der Sprachsteuerung deutlich schneller, da viele Aktionen auf einmal ausgelöst werden konnten.

Die Evaluation zeigt jedoch auch, dass das im Rahmen dieser Arbeit entwickelte Prototypensystem noch zu viele Schwächen hat, um eine echte Alternative zur Steuerung von intelligenten Umgebungen darzustellen.

Die Kombination aus Sprachsteuerung und Deutgestenerkennung erwies sich im Test als umständlicher als beide Systeme einzeln für sich. Dies bedeutet jedoch nicht, dass eine Einbindung von Deutgestenerkennung in einen Sprachassistenten oder umgekehrt nicht sinnvoll ist, lediglich die hier getestete Umsetzung konnte die Probanden nicht überzeugen.

Die Deutgestensteuerung wurde von den meisten Probanden als sehr intuitiv interpretiert. Auf der System Usability Scale ist diese Tendenz ebenso erkennbar: Die Aussagen 3 und 7 erhielten fast durchgehend volle Zustimmung.

Einige der Tester merkten jedoch auch an, dass sie bei einem kamerabasierten System bedenken hätten, sich dieses ins Eigenheim zu stellen, aus Angst vor einem möglichen Missbrauch der Videodaten.

KONKLUSION

Deutgesten sind eine der frühsten und intuitivsten Kommunikationsformen des Menschen. Ihr Einsatz zur Steuerung von modernen User Interfaces war bislang nur durch die noch nicht abrufbare Rechenleistung und die benötigten Algorithmen beschränkt. Im Zeitalter von 3-dimensionaler Gesichtserkennung in Smartphones wurden diese technischen Hürden nun ausgehebelt. Der im Rahmen dieser Arbeit entwickelte Prototyp zur kamerabasierten Deutgestenerkennung erfüllt nicht alle an ihn gestellten Anforderungen. Die Abstriche, die zugunsten einer guten Performance auf schwacher Hardware gemacht werden mussten, verursachten zahlreiche Probleme, welche den Algorithmus nur unter bestimmten Voraussetzungen zuverlässig funktionieren lassen.

Das Konzept, Deutgesten zur Steuerung von intelligenten Umgebungen zu verwenden, ist aber durchaus nicht abwegig.

Themenverwandte Arbeit

STEFAN NOSOVIC

In der Forschungsarbeit wird versucht, Handgesten mithilfe von Accelerometern und Gyroskopen von Smart Watches durch Pattern Matching zu identifizieren. [12]

Um sich von diesem Forschungsprojekt abzugrenzen, und aufgrund anderer hier gestellter Anforderungen an das System, wurde statt auf Smart Wearables in dieser Arbeit kamerabasierte Deutgestenerkennung untersucht.

PETERS, SEBASTIAN MATTHIAS

In MIBO – „MIBO – A Framework for the Integration of Multimodal Intuitive Controls in Smart Buildings“ [13] wird ein Interface für die Verwaltung von multimodalen Bedienungskonzepten für intelligente Umgebungen vorgestellt.

Eine Anbindung der Deutgestensteuerung an das MIBO Framework könnte, sofern die Technologie der Deutgestenerkennung ausgereift genug ist, erfolgen.

Abbildungsverzeichnis

Referenzen

[1] M. Weiser, „The origins of ubiquitois computing research at PARC in the late 1980s“.

[2] C. Colonnesi, G. J. J. M. Stams, I. Koster und M. J. Noom, „The Relation between Pointing and Language Development: A Meta-Analysis“.

[3] D. A. Leavens, W. D. Hopkins und K. A. Bard, „Understanding the Point of Chimpanzee Pointing, Epigenesis and Ecological Validity“.

[4] K. Duden, Duden, 2017.

[5] „OpenCV,“ https://opencv.org/.

[6] „dlb,“ http://dlib.net/.

[7] F. Abercassis, „OpenCV - Morphological Skeleton,“ http://felix.abecassis.me/2011/09/opencv-morphological-skeleton/.

[8] F. Mokhtarian und S. Abbasi, „Shape similarity retrieval under affine transforms,“ 2002.

[9] Roy, „http://www.morethantechnical.com/,“ 27 12 2012. [Online]. Available: http://www.morethantechnical.com/2012/12/27/2d-curve-matching-in-opencv-w-code/. [Zugriff am 10 9 2017].

[10] B. T. Navneet Dalal, „Histograms of Oriented Gradients for Human Detection“.

[11] M. Kawulok, T. Grzejszczak, J. Nalepa und M. Knyc, „Database for hand gesture recognition,“ http://sun.aei.polsl.pl/~mkawulok/gestures/.

[12] S. Nosovic, „Hand Gesture Recognition using Sensor Data from a Smart Watch,“ https://www1.in.tum.de/lehrstuhl_1/people/536-stefan-nosovic.

[13] S. M. Peters, „MIBO – A Framework for the Integration of Multimodal Intuitive Controls in Smart Buildings,“ http://mediatum.ub.tum.de/?id=1304127.

[14] G. Klinker, „Implementation and Performance of a Complex Vision System on a Systolic Array Machine (Area Qualifier),“ http://campar.in.tum.de/Chair/KlinkerCMU.

[15] F. Bayer und S. Weber, „Verknüpfung von Sprachassistenten mit anderen Steuerungsmethoden in intelligenten Umgebungen“.

[16] J. Kawulok und M. Nalepa, „Fast and accurate hand shape classification“.*Beyond Databases, Architectures, and Structures, S. Kozielski, D. Mrozek, P. Kasprowski, B. Malysiak-Mrozek, and D.*

Kostrzewa, Eds., vol. 424 of Communications in Computer and Information Science, pp. 364-373. Springer, 2014..

[17] T. Grzejszczak, M. Kawulok und A. Galuszka, „Hand landmarks detection and localization in color images“.*Multimedia Tools and Applications, vol. 75, no. 23, pp. 16363-16387, 2016..*

[18] M. Kawulok, J. Kawulok, J. Nalepa und B. Smolka, „Self-adaptive algorithm for segmenting skin regions,“ *EURASIP Journal on Advances in Signal Processing, vol. 2014, no. 170,* 2014.

Anhang

QUELLCODE

```
import imutils
import dlib
import cv2
import numpy as np
import time
import skimage
import codecs

from urllib.request import urlopen
import json

# API Key for Florian Bayer's Trigger Interface
API_KEY =
"BoUANSU7XC0iFDJzYEo1YyNsun8fdhyEvi691Q6Oi5XNyxcn2rdQQQ9b57g8dFCzOxxEdsjln7YMt2Zlj42ZJ11DW8B1C
4pgqNLt7VFkRtWblMmvFdkf4bJQ0aHC4WJK"
# A decoder for utf-8 as python3 treats the response as a byte stream
reader = codecs.getreader("utf-8")

# Now let's use the detector as you would in a normal application.  First we
# will load it from disk.
detector = dlib.simple_object_detector("detector.svm")

# Video capture source
cap = (cv2.VideoCapture(0), cv2.VideoCapture(1))

# Reduce capture quality to avoid bandwith problems
cap[0].set(cv2.CAP_PROP_FRAME_WIDTH , 400);
cap[0].set(cv2.CAP_PROP_FRAME_HEIGHT , 300);
cap[1].set(cv2.CAP_PROP_FRAME_WIDTH , 400);
cap[1].set(cv2.CAP_PROP_FRAME_HEIGHT , 300);

# Background substractor
fgbg = (cv2.createBackgroundSubtractorMOG2(500, 16.0, 1),
cv2.createBackgroundSubtractorMOG2(500, 16.0, 1))
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (7, 7))

win_det = dlib.image_window()
win_det.set_image(detector)

win = dlib.image_window()

# Minimum and Maximum thickness of the arm
max_height = 100
min_height = 2
# Minimum length the arm segment must have
min_width = 100
# Search region length for start and end points
sr_len = 3
# Shoulder offset, distance to shoulder to make the accuracy better
sr_offset = 60
# Scan offset to include more of the arm in case components got disconnected
sc_offset = 20
# Trigger Radius
tr_radius = 200
# Trigger time
tr_time = 2

# Vector of
# points which represent the center of the light seperate for all frames
```

```
# timestamps when they were triggered the last time
# values stating how triggered they are
# the respective triggers
actors = [((0, 0), (0, 0)), 0, 0, "Bottom Right"),
          (((800, 0), (800, 0)), 0, 0, "Bottom Left")]


# Function which triggers actors via Florian Bayer's Trigger interface (taken from his
#  implementation)
def triggerAction(actionId):
    responseString = urlopen(http://smater.diebayers.de/?f=function&func id=
                             +str(actionId)+"&key="+str(API_KEY))
    jsonDecode = json.load(reader(responseString))
    responseType = jsonDecode["responce"]["type"];
    if responseType == "SUCCESS":
        return 1
    else:
        return 0

# Function that gets a line defined by a starting point and a direction point and
# accordingly updates what is triggered
def applyPG(point1, point2):
    for k, (centers, then, value, trigger) in enumerate(actors):
        # Subtract time since last time
        now = time.time()
        delta = int(now - then)
        value = max(value - delta * 10, 0)

        # Calculate distance between line and point
        point3 = centers[cur_cam]
        x, y = point3
        # Check if the point is on the right side of the line
        if(np.sign(point2[0] - point1[0]) == np.sign(point2[0] - x)):
            d = np.linalg.norm(np.cross(point2 - point1, point1 - point3))
                / np.linalg.norm(point2 - point1)
            if d < tr_radius:
                # Add to the trigger value
                value += (tr_radius - d) / 2
                # Check if it is triggered
                if value > 100 * tr time:
                    print (trigger + " triggered ")
                    # Trigger the light
                    triggerAction(k)
                    value = 0

        actors[k] = (centers, now, value, trigger)


# Function that gets image part, and possible arm start and end points and checks
# via Objectdetector if that is indeed an arm pointing at something
def checkForArm(img, point1, point2):
    # point1 should be left, point2 right
    if point1[0] > point2[0]:
        point1, point2 = point2, point1


    # dir always pointing right
    dir = np.subtract(point2, point1)
    vlen = np.linalg.norm(dir)
    dir_norm = (dir[0] / vlen, dir[1] / vlen)

    # Extend Points outwards
    point1 = np.add(point1, np.multiply(dir_norm, -sc_offset))
    point2 = np.add(point2, np.multiply(dir_norm, sc_offset))

    # Readjust len and dir
    dir = np.subtract(point2, point1)
    vlen = np.linalg.norm(dir)

    # Orth always pointing up
```

```
    orth = (- dir[1] / vlen, dir[0] / vlen)
    corner_src_1 = np.add(point1, np.multiply(orth , max_height))
    corner_src_2 = np.add(point2, np.multiply(orth, max_height))
    corner_src_3 = np.subtract(point2, np.multiply(orth, max_height))
    corner_src_4 = np.subtract(point1, np.multiply(orth, max_height))
    corners_src = np.float32([corner_src_1, corner_src_2, corner_src_3, corner_src_4])

    corner_dest_1 = (2 * max_height, 0)
    corner_dest_2 = (2 * max_height, vlen)
    corner_dest_3 = (0, vlen)
    corner_dest_4 = (0, 0)
    corners_dest = np.float32([corner_dest_1, corner_dest_2, corner_dest_3, corner_dest_4])

    M = cv2.getPerspectiveTransform(corners_src, corners_dest)
    warped = cv2.warpPerspective(img, M, (2 * max_height, int(vlen)))
    warped = imutils.resize(warped, width = 300)
    warped = cv2.threshold(warped, 155, 255, cv2.THRESH_BINARY_INV)[1]

    rects = detector(warped)


    if len(rects) > 0:
        # We detected something pointing upward, which from point2 towards point1
        cv2.line(img, (int(point2[0]),
                 int(point2[1])),
                 (int(point1[0] - 10*dir[0]),
                 int(point1[1] - 10*dir[1])),
                 (255, 255, 255), 1)
        applyPG(point2, point1)
        cv2.imshow("Detection", warped)

    # TODO Sometimes detects wrong way around, so not exclusive this time
    # else:

    warped = cv2.flip(warped, 0)
    rects = detector(warped)

    if len(rects) > 0:
        # We detected something pointing downward, so from point1 towards point2
        cv2.line(img, (int(point1[0]), int(point1[1])), (int(point2[0] + dir[0]),
                 int(point2[1] + dir[1])), (255, 255, 255), 1)
        applyPG(point1, point2)
        cv2.imshow("Detection", warped)

cur_cam = 1

# Work loop
while True:
    # Hard coded to work with two cameras
    cur_cam = (cur_cam + 1) % 2
    ret, image = cap[cur_cam].read()

    # Downsize the image for performance
    image = imutils.resize(image, width=800)

    # Update the background model
    fgmask = fgbg[cur_cam].apply(image)

    # Remove some of the noise
    fgmask = cv2.morphologyEx(fgmask, cv2.MORPH_CLOSE, kernel)

    # Include the detected shadows (gray) into the background
    ret, fgmask = cv2.threshold(fgmask,250,255,cv2.THRESH_BINARY)

    # Detect the connected components
    output = cv2.connectedComponentsWithStats(fgmask, 4, cv2.CV_16U)

    # The first cell is the number of labels
    num_labels = output[0]
    # The second cell is the label matrix
```

```
labels = output[1]
# The third cell is the stat matrix
stats = output[2]
# The fourth cell is the centroid matrix
centroids = output[3]

for k, d in enumerate(stats):
    x1 = d[cv2.CC_STAT_LEFT]
    y1 = d[cv2.CC_STAT_TOP]
    w = d[cv2.CC_STAT_WIDTH]
    h = d[cv2.CC_STAT_HEIGHT]
    x2 = x1 + w
    y2 = y1 + h

    if d[cv2.CC_STAT_AREA] > 1000 and w < 700 and w > 10:
        # Filter out stuff that does not belong to the component
        layer_mask = labels[y1:y2, x1:x2]
        mask = np.ones((x2-x1, y2-y1), cv2.)
        mask[layer_mask != k] = 0

        subimage = cv2.bitwise_and(fgmask[y1:y2, x1:x2], fgmask[y1:y2, x1:x2], mask)

        # Project upon horizontal axis
        projection = cv2.reduce(subimage, 0, cv2.REDUCE_SUM, dtype=cv2.CV_32S)[0]

        # Search left to right
        for i, v in enumerate(projection):
            v /= 255
            if v > max_height or v < min_height:
                # End of arm detected
                if i > min_width:
                    # Arm segment from left to right long enough
                    M = cv2.moments(subimage[0:h, 0:sr_len], 0)
                    point1 = (int(M['m10'] / M['m00']) + x1,
                              int(M['m01'] / M['m00']) + y1)
                    M = cv2.moments(subimage[0:h, i-sr_len-1-sr_offset:i-1-sr_offset], 0)
                    point2 = (int(M['m10'] / M['m00']) + x1 + i - sr_offset,
                              int(M['m01'] / M['m00']) + y1)

                    checkForArm(fgmask, point1, point2)

                    # Visualisation on the mask
                    cv2.circle(fgmask, point1, 10, (255, 255, 255), 2)
                    cv2.circle(fgmask, point2, 10, (0, 0, 0), 2)
                    # cv2.rectangle(fgmask, (x1, y1), (x1 + i, y2), (255, 255, 255), 2)
                break

        else:
            # Complete subimage was arm, direction unknown
            M = cv2.moments(subimage[0:h, 0:sr_len], 0)
            point1 = (int(M['m10'] / M['m00']) + x1, int(M['m01'] / M['m00']) + y1)
            M = cv2.moments(subimage[0:h, w - sr_len:w], 0)
            point2 = (int(M['m10'] / M['m00']) + x2, int(M['m01'] / M['m00']) + y1)

            checkForArm(fgmask, point1, point2)

            # Visualisation
            cv2.circle(fgmask, point1, 10, (255, 255, 255), 2)
            cv2.circle(fgmask, point2, 10, (255, 255, 255), 2)
            # cv2.rectangle(fgmask, (x1, y1), (x2, y2), (255, 255, 255), 2)

        # Search right to left
        for i, v in enumerate(reversed(projection)):
            v /= 255
            if v > max_height or v < min_height:
                # End of arm detected
                if i > min_width:
                    # Arm segment from right to left long enough
                    M = cv2.moments(subimage[0:h, w - sr_len:w], 0)
                    point1 = (int(M['m10'] / M['m00']) + x2,
```

```
                        int(M['m01'] / M['m00']) + y1)
                    M = cv2.moments(subimage[0:h, w - i + sr_offset:w - i
                                             + sr_len + sr_offset], 0)
                    point2 = (int(M['m10'] / M['m00']) + x2 - i + sr_offset,
                              int(M['m01'] / M['m00']) + y1)

                    checkForArm(fgmask, point1, point2)

                    # Visualisation
                    cv2.circle(fgmask, point1, 10, (255, 255, 255), 2)
                    cv2.circle(fgmask, point2, 10, (0, 0, 0), 2)
                break

        # Display the different densities
        #for i, v in enumerate(projection):
            #cv2.line(fgmask, (x1 + i, y1), (x1 + i, y1 - int((v/ 512))), (255, 255 ,255))

        # cv2.rectangle(fgmask, (x1, y1), (x2, y2), (255, 255, 255), 2)


ret, fgmask = cv2.threshold(fgmask,250,255,cv2.THRESH_BINARY_INV)

cv2.imshow("Output" + str(cur_cam), fgmask)

k = cv2.waitKey(30) & 0xff
if k == 27:
    break
```