Bauhaus-Universität Weimar
Fakultät Medien
Studiengang Medieninformatik

Robustes Hand- und Finger-Tracking

Eine Methode zur Optimierung von Tracking auf Multi-Touch Tabletops

Bachelorarbeit

Moritz Loos
Geboren am 03.01.1994 in Weimar

Matrikelnummer 112385

1. Gutachter: Junior-Prof. Dr. Florian Echtler
2. Gutachter: Prof. Dr. Bernd Fröhlich

Datum der Abgabe: 31.03.2016

Erklärung

Hiermit versichere ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Weimar, den 31.03.2016

.....
Moritz Loos

Zusammenfassung

Mit der heutigen Zeit haben sich immer mehr Touch-Geräte etabliert und werden täglich von einer breiten Masse der Bevölkerung genutzt. Dabei steigen die Anforderungen an Touch-Systeme mit zunehmender Größe der Geräte immer weiter an. Um Touch-Interaktionen auch auf großen Geräten stabil zu realisieren, gibt es bislang jedoch keine vollständig akzeptierte Lösung.

Im Rahmen dieser Arbeit wurde ein System zur Erkennung und stabilen Zuordnung von Touch-Eingaben bei diffusen Infrarot-Beleuchtungssetups implementiert. Dabei wurde ein neuer Ansatz entwickelt, welcher die Handzuordnung aus dem Kamerabild extrahiert und für das Tracking der Finger berücksichtigt. Hierzu werden eine Reihe von existierenden Trackingmethoden diskutiert und ihre Effizienz während einer Studie evaluiert.

Inhaltsverzeichnis

Kapitel 1

Einführung und Motivation

Im Verlauf der rasanten Technikentwicklung haben sich immer mehr „Touch“-Geräte etabliert und dominieren den heutigen Elektronikmarkt. Mittlerweile sind Tablets und Notebooks mit Touch-Displays keine Seltenheit mehr. So ist u.a. das Smartphone kaum noch aus dem Alltag wegzudenken und weit verbreitete Gerätebeispiele wie das „iPhone“ von Apple gelten als Vorreiter in diesem Bereich. Darüber hinaus gibt es zahlreiche weitere technische Geräte, welche sich über Touch-Berührungen steuern lassen. Hierzu gehören Fernbedienungen, Computer Zeigegeräte, MP3-Player, Kameras und TVs, um nur einige Beispiele zu nennen.

Mit zunehmender Anzahl unterschiedlicher Geräte mit verschiedenen Voraussetzungen, steigt auch die Anzahl an neuen Erkennungstechnologien für Interaktionen, die über Touch-Eingabe arbeiten. Es gibt dazu Technologien, die über elektrische Impulse den Fingerspitzen-Kontakt ausmachen, sogenannte kapazitive Displays, aber auch druckempfindliche Displays (Resistive Touchscreens), welche über Drucksensoren die Fingerposition bestimmen. Darüber hinaus gibt es Varianten, die beispielsweise über Infrarot-Layer, Induktionsstrom oder über Bildanalyse die Berührungspunkte erkennen.
Die Touch-Erkennungstechniken haben alle eine Gemeinsamkeit: Sie messen die Position der Berührungen und tracken anschließend, sodass keine Interaktion einzelner Finger verloren geht. Tracking heißt, dass man über die gesamte Dauer der Berührung der Finger die jeweilige Position verfolgt und richtig zuordnet.

Bei Interaktionen mit kleineren Geräten, wie Smartphone oder Tablets, wird dies Erfahrungsgemäß auch zuverlässig umgesetzt. Jedoch sind diese, alleine schon durch die geringe Größe, limitiert. Das Gerät wird vorrangig nur von einem Nutzer bedient und muss auch nur einfache Interaktionen, wie

standardmäßig ein-, zwei- oder drei-Finger Gesten, ermöglichen. Zudem wird die Bewegungsgeschwindigkeit mit der gezielte Aktionen durchgeführt werden, durch die geringe Display Größe und hohen Auflösung, eingegrenzt.

Die Problematik der Zuordnung und Erkennung der Finger taucht erst bei immer größer werdenden Setups auf. Mit den heutigen Anwendungen, vor allem im professionellen Bereich, wächst die Forderung nach großen Interaktionsräumen wie große Tischdisplays, den sogenannten Tabletops.
Große Setups bieten mehr Platz und ermöglichen somit gleichzeitig die Interaktion von mehreren Nutzern. Dadurch steigt auch implizit die Anforderung an das System. So muss die Software Interaktionen mehrerer Nutzer erkennen und geeigneter Weise voneinander unterscheiden können. Darüber hinaus müssen schnelle Bewegungen zuverlässig den Nutzern und den Aktionen zugeordnet werden.

Durch eine größere Touchfläche werden zudem viel komplexere Bewegungen möglich. Anstatt nur im Browser zu scrollen oder Bilder zu skalieren, wird über Touch-Interaktionen beispielsweise auch durch 3D Szenen navigiert oder Objekte manipuliert. Eine höhere Anzahl an Freiheitsgraden, führt dazu, dass die Anforderung an die Gesten steigen. Das heißt aufgrund der komplexeren Anwendungen müssen neben den bekannten ein-, zwei-, drei-Finger Gesten auch 1-Hand oder 2-Hand Gesten unterstützt werden und das bei schnelleren Bewegungen.

Im Verlaufe der Arbeiten zu diesem Thema im Rahmen des Studiums habe ich viel mit einem Multi-Touch-Tisch gearbeitet und die Limitierungen aktuell verfügbarer Systeme ausgewertet. Es wurden nicht immer alle Interaktionen erkannt und richtig verfolgt. Dies hat natürlich große Auswirkung auf die darauf aufbauende Anwendung, da eine falsche Zuordnung zu unerwünschten Ergebnissen führt, wie z. B. die Auswahl falscher Gesten. Um dieses unerwünschte Verhalten einzugrenzen, muss eine entsprechende Software ('Touch-Treiber') dementsprechend verlässlich Finger erkennen und zuverlässig zuordnen.

Mit der vorliegenden Arbeit will ich diese Problematik aufgreifen und sowohl Erkennung, als auch die Verfolgung der Finger stabilisieren. Die Arbeit beinhaltet eine Systementwicklung für ein stabiles Erkennen und robustes Zuordnen der Finger. Dabei liegt der Schwerpunkt der Arbeit auf dem Tracking. Außerdem werden Konzepte für eine weitere Auseinandersetzungen mit diesem Thema vorgestellt.

Im Nachfolgenden Abschnitt 1.1 gehe ich zuerst auf die grundlegenden Anfor-

derungen eines Multi-Touch Tabletops ein, um anschließend zur Problematik des Trackings im Absatz 1.2 zu kommen.

Im Anschluss beschreibe ich bekannte Lösungsansätze im Kapitel Verwandte Arbeiten und Lösungsansätze auf S. 11 und gehe ausführlich auf meinen neuen Lösungsansatz im Kapitel3 [Eigener Lösungsansatz] auf S. 25 ein.

Das Kapitel 4 beschäftigt sich mit der Softwareumsetzung.

Mein neuer Lösungsansatz wurde durch eine Studie auf die Effektivität und Robustheit geprüft. Zusätzlich wurden die bekannten Tracking-Methoden verglichen. Die Ergebnisse werden auf der Seite 25 im Kapitel [Vergleichsstudie von Tracking Methoden] ausführlich ausgewertet.

1.1 Grundlagen

In diesem Kapitel beschreibe ich zum besseren Verständnis die Anforderungen an Hard- und Software, um Multi-Touch Lösungen für Tabletops zu realisieren.

1.1.1 Hardware

Um die Positionen der Finger bestimmen zu können, wird ein entsprechender Sensor benötigt. Dafür gibt es verschiedenste Herangehensweisen: Von kapazitiven zu resistiven oder induktiven Displays, bis hin zu optischen Systemen. Die Beschreibung aller Technologien würde das Ausmaß dieser Bachelorarbeit übersteigen, weshalb ich nachfolgend nur auf die von mir gewählte optische Variante eingehen werde.

1.1.1.1 Diffuse Illumination (DI)

Die Diffuse Illumination (engl. diffused illumination) ist ein Verfahren, bei dem Berührungspunkte mithilfe einer Kamera bestimmt werden. Wie in folgender Abbildung 1 schematisch dargestellt ist, wird bei der Diffusen Illumination Infrarot Licht verwendet, um die Region über dem Display auszuleuchten und Berührungen im Kamerabild zu verstärken.

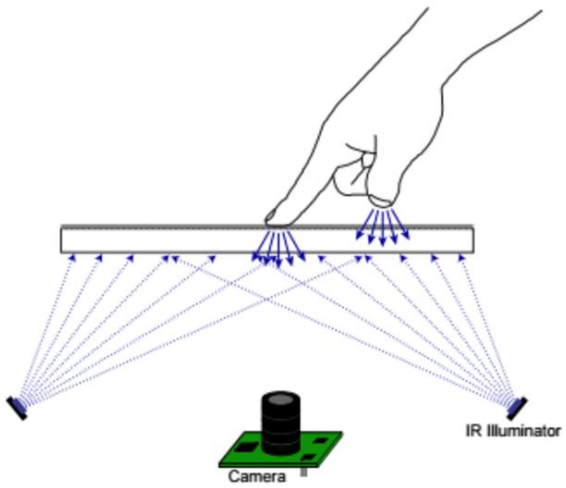


Abbildung 1: Schematische Darstellung der Diffusen Illumination.

Dabei strahlt das Licht von Unten durch die Unterseite des Displays und wird an aufliegenden Objekten, wie beispielsweise einer Hand, reflektiert. Die Reflexionen werden über eine Kamera mit Infrarotfilter aufgezeichnet. Dabei ist das reflektierte Licht von näheren Objekten, wie aufliegende Fingerspitzen, intensiver als das von entfernteren Objekten. Das Intensitätsbild der Kamera (siehe Abbildung 2) wird anschließend verwendet, um Finger zu erkennen. Eine gleichmäßige Ausleuchtung ist hier folglich von Vorteil.

Abbildung 2: Beispiel eines Kamerabildes eines Diffusen Illumination Setups.

Hingegen anderer Touch-Technologien bietet das komplette Kamerabild den

Vorteil, dass keinerlei Informationen verloren gehen, sodass zusätzlich auch Objekte wie z. B Marker erkannt werden könnten.

1.1.1.2 Aufbau

Stickert et. al haben im Rahmen der Entwicklung dreidimensionaler Navigationstechniken das gleiche diffuse Infrarot-Beleuchtungssetup 3 verwendet. Die nachfolgende Abbildung 3 verdeutlicht den Aufbau.

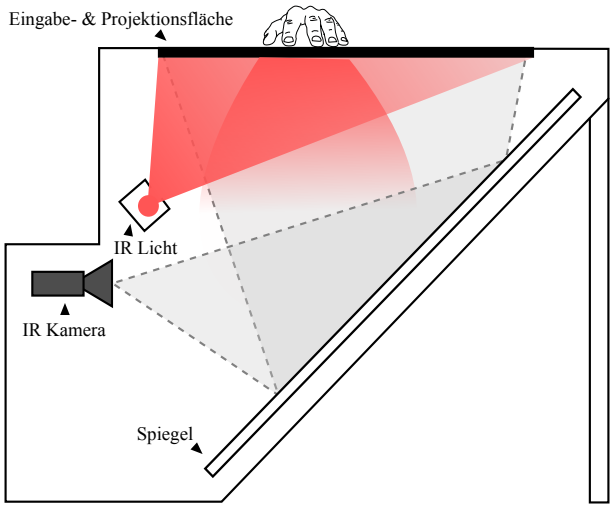


Abbildung 3: Das verwendeten Hardware Setup zum Tracking von Touch-Eingaben.

Im Tisch befindet sich ein Projektor, welcher mit 360Hz über einen Spiegel Stereobilder für bis zu 3 Nutzer auf die Milchglas-Oberfläche projizieren kann. Damit jeder Nutzer ein für seine Position korrektes Stereobildpaar erhält, wird auf ein fest installiertes Infrarot-Tracking-System und Shutter-Brillen zurückgegriffen. Das Tracking-System ermittelt kontinuierlich die Kopfposition und Kopforientierung von bis zu drei Nutzern und sorgt somit für ein perspektivisch korrektes Bild. Die Shutter-Brillen öffnen sich, sobald der Beamer das Bild des jeweiligen Nutzers sendet und schließt, sobald das eigene Bild durch das von einem anderen Nutzer abgelöst wird [Kulik et al., 2011].

Für die Touch-Erkennung befindet sich in dem Tisch eine PointGrey Grasshopper3 Kamera mit eingebauten Infrarot Filter. Die Kamera hat eine Auflösung von 1920 x 1200 (vgl. [PointGrey, 2015]) und kommt bei halber Auflösung auf 163FPS. Für eine annähernd gleichmäßige Ausleuchtung sind 11 Infrarot LEDs

mit Diffusor an die Innenseite des Tisches angebracht, welche teilweise über den Spiegel, aber auch direkt von Unten, die Bildfläche des Tisches beleuchtet.

1.1.2 Software

Was muss eine Touch-Software leisten? In diesem Kapitel werden die Grundbausteine des optischen Tracking-Systems für Multi-Touch-Tischdisplays erläutert, die zum Verständnis der in dieser Arbeit entwickelten Erweiterungen nötig sind. Eine genaue Erklärung zur Umsetzung dieser Schritte wird im Kapitel [Eigener Lösungsansatz] auf Seite 25 genauer erläutert.

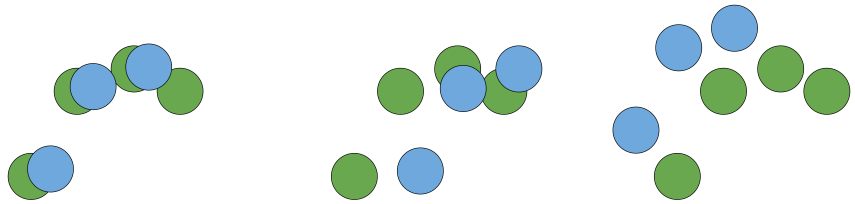
Im Allgemeinen gibt es vier Schritte:

1. Bild Kalibrierung
2. Erkennung
3. Tracking
4. Senden

Als erstes muss die Software gewährleisten, dass das Bild der Kamera empfangen wird und für die weiteren Schritte aufbereitet wird. Anschließend muss das Bild ausgewertet werden, sodass Informationen über die erfassten Finger gesammelt werden können und im nächsten Schritt das Tracking erfolgen kann. Um auf die Berührungseingaben unmittelbar eingehen zu können, muss dieser Prozess in Echtzeit ablaufen und die erkannten Informationen prompt zur entsprechenden Anwendung weitergeleitet werden.

1.2 Problembeschreibung

Das Tracking lässt sich als ein Korrespondenzproblem betrachten. Es sollen die korrekten Verbindungen zwischen den Punkten einer Gruppe zu den Punkten einer anderen Gruppe gefunden werden (siehe Abbildung 4). Bei der linken Abbildung (a) ist die Zuordnung noch Trivial, bei der Abbildung in der Mitte (b) wird es schon schwieriger.



(a) (b) (c)

Abbildung 4: Beispiele für mögliche Punktkonfigurationen zum tracken.

Die Digitalisierung von analoge Bewegungen hängt immer stark von der jeweiligen Abtastrate ab. Je größer die Abtastrate, desto geringer der Abstand zum letzten Frame und desto genauer die Bewegung. Dieser Zusammenhang ist besonders für das Tracking wichtig.
Wie in der Abbildung (a) zu (b) deutlich wird, erleichtern geringere Abstände die Zuordnung.
Das bedeutet die Problematik des reinen Positions-Trackings steht im direkten Zusammenhang mit der Geschwindigkeit mit welcher neue Blobs (verweis auf Abschnitt 2.2.1) erkannt werden, aber auch an der Geschwindigkeit, mit der der Nutzer seine Hand bewegt. Je schneller die Hand bewegt wird, desto weiter weg sind die erkannten Blobs im nächsten Frame und desto schwieriger das Tracking.
Das Tracking wird zudem durch die Bewegungsrichtung der Finger beeinflusst. Eine Bewegung in Richtung anderer Finger ist schwieriger, als eine Bewegung in entgegengesetzter Richtung, da dabei die Distanzen zu anderen Punkten verringert werden. Die Abbildung 4(c) zeigt eine Verschiebung der blauen Punkte, anstatt nach rechts, wie bei (b) nach oben links. Die zurückgelegte Distanz ist die gleiche, aber das Tracking schon deutlich einfacher.
Neben der Distanz und Bewegungsrichtung wird das Tracking mit steigender Anzahl an Fingern erschwert. Je größer die Anzahl der Finger in unmittelbarer Nähe desto schwieriger wird es die korrekte Zuordnung zu finden.
Die zentrale Problematik des Trackings ist demnach trotz großer Abstände zwischen den Punkten und unabhängig von der Anzahl, Punkte von Frame zu Frame korrekt zuzuordnen.

Kapitel 2

Verwandte Arbeiten und Lösungsansätze

In diesem Kapitel sind ausgewählte Verfahren und Algorithmen beschrieben, die im Zusammenhang mit Touch-Software und Diffuser Illumination bereits existieren. Dabei gehe ich zuerst auf die Bild Kalibrierung ein. Danach komme ich in chronologischer Reihenfolge zu Erkennung und Tracking.

2.1 Bild Kalibrierung

Das Bild muss zu Beginn von der Kamera angefordert und erstmal an die physikalischen Restriktionen angepasst werden. Das heißt, die Kamera muss so eingerichtet werden, dass sie senkrecht von unten auf die Tischplatte schaut und ausschließlich den Touch-Bereich aufnimmt. Da dies physikalisch nicht immer möglich ist, muss dies von der Software angepasst werden.

Um die exakte Position von Fingern zu bestimmen, muss das Kamerabild so genau wie möglich an die Realität angepasst werden. Das heißt, es muss sowohl die Krümmung des Objektives, sowie die perspektivische Verzerrung herausgerechnet werden und anschließend nur die relevante Touch-Ebene herausschneiden.

Für die Bild-Kalibrierung bietet die Open Source Computer Vision Bibliothek OpenCV viele Methoden an. Die Krümmung des Objektives kann mithilfe eines Schachbrettmusters und der Methode cv::calibrateCamera bestimmt und herausgerechnet werden.

Zur perspektivischen Verzerrung eignet sich die Methode

cv::getPerspectiveTransform von OpenCV. Diese liefert eine Homography-Matrix, die das Bild zu definierten Punkten verzerrt und somit bei korrekter Parametrisierung entzerrt.

2.1.1 Hintergrundsubtraktion

Die Hintergrundsubtraktion (engl. Background Substraction) ist ein essentieller Schritt für die Touch-Erkennung. Hierbei wird der Hintergrund herausgeschnitten, um Veränderungen hervorzuheben.
Dafür muss vorerst ein Hintergrund Bild I_{min}, das heißt ein Bild ohne jeglichen Gegenständen, auf dem Touch-Bereich gespeichert werden.

$$I_{sub} = I - I_{min} \tag{2.1}$$

Anschließend wird dieses Hintergrundbild in jedem Frame vom aktuellen Bild I subtrahiert und übrig bleiben nur noch Veränderungen I_{sub} (siehe Abbildung A.37(b)).

2.2 Erkennung

Das folgende Kapitel widmet sich der Bestimmung von aufliegenden Fingern.

2.2.1 Binarisierung der Bilder

Für die Erkennung von Fingerpositionen wird das Bild typischerweise binarisiert. Das heißt, dass das Bild über einen definierten Grenzwert (Threshold) in helle und dunkle Bildteile getrennt wird (siehe Abbildung 5).

Abbildung 5: Erkennung der Fingerpositionen über Binarisierung.

Da die aufliegenden Finger am nächsten zur Touch-Ebene sind, befinden sie sich auch am nächsten zur Lichtquelle und sind somit am hellsten. Da das eigene System bekannt ist und man die Helligkeit von aufliegenden Fingerspitzen kennt, wird der Schwellenwert so definiert, dass die aufliegenden Finger hervorgehoben werden und der Rest ignoriert wird. Die so hervorgehobenen Fingerspitzen können daraufhin zu Blobs zusammengefasst werden.
Blobs sind voneinander abgetrennte Strukturen im Bild. Die Definition von Blobs unterscheidet sich in der Literatur stark. Die einen definieren die Fingerspitzen als Blobs, die Anderen sehen jegliche Strukturen, wie eine ganze Hand oder auch ein abgelegter Gegenstand, als Blob. Für mich sind Blobs nur Fingerspitzen.
Diese können über einen einfachen Detektor gefunden und positioniert werden. Dabei werden weiße Pixel im Bild gruppiert und das Zentrum als Position verwendet. Das quelloffene Computer Vision Framework OpenCV bietet dafür beispielsweise die cv::SimpleBlobDetector Methode an. Da die Finger und somit die Blobs eine bestimmte Struktur besitzen, kann die Methode entsprechend parametrisiert werden, um beispielsweise nur runde Blobs einer bestimmten Größe zu erkennen.

Eine weitere Methode, um die Position der Fingerspitzen zu bestimmen oder die Erkennung zu stabilisieren, ist das Anwenden eines Hochpass-Filters. Durch den Hochpass-Filter werden Konturen erkannt und hervorgehoben. Da die Finger direkt auf der Oberfläche liegen, sind diese deutlich schärfer als die dahinterliegende Hand und können dementsprechend über den Hochpassfilter lokalisiert werden.

2.2.2 MSER

Der MSER-Algorithmus (engl. maximal stable regions) geht auf Matas et al. [Matas et al., 2004] zurück, wobei ursprünglich eigenschaftsbezogene Korrespondenzen zwischen Stereobildpaaren gefunden werden sollen.
Ewerling et al. nutzen diese Erkenntnise zur Erkennung von Berührungspunkten und stellen diesen für die Verwendung in einer Touch-Software vor [Ewerling et al., 2012].

Der MSER-Algorithmus teilt das Intensitätsbild in sogennante stabile Extremal Regions (ER) auf. Diese Extremal Regions sind geschlossene Pixelregionen, welche je nach Stabilitätskriterium entweder eine höhere oder niedrigere Farbintensität als umliegende Pixel im Bild besitzen. Diese Regionen werden mithilfe des MSER-Algorithmus extrahiert und als hierarchische Struktur, welche als ComponentTree (Komponentenbaum) bezeichnet wird, gespeichert.

Stabile Regionen werden über Fünf Parameter definiert:

- Delta
- MinArea
- MaxArea
- MaxVariation
- MinDiversity

Der Parameter “Delta” beschreibt die Inkrementierungsstufe, bei denen der Algorithmus schaut, ob Regionen stabil sind. Je größer Delta, desto weniger stabile Regionen gibt es. Beispielweise wird bei Delta 1, jeder Graustufe und bei Delta 10 nur jeder zehnte Grauwert nach stabilen Regionen gesucht. Um die Regionen in ihrer Größe zu definieren, gibt es die Parameter MinArea und MaxArea. Die stabilen Regionen dürfen nicht kleiner als der Wert “MinArea” sein und dürfen eine bestimmte Größe “MaxArea” nicht überschreiten.
Die Variation beschreibt die relative Größe zu dem Vater. Je größer die maximale Variation, desto mehr stabile Regionen gibt es.
Zuletzt gibt es noch den Parameter “MinDiversity”. Dieser Parameter ist dafür da Regionen, welche sich nur in wenigen Pixeln unterscheiden auszusortieren - Je größer dieser Wert, desto weniger stabile Regionen gibt es.

Die Struktur des Komponentenbaumes lässt sich hervorragend für die Erkennung von Fingern nutzen. In folgender Abbildung 6 sind die ER, als approximierte Ellipsen, im Kamerabild dargestellt. Zugehörig zu den gefundenen ER

befindet sich auf der rechten Seite der Abbildung der visualisierte Component-Tree.

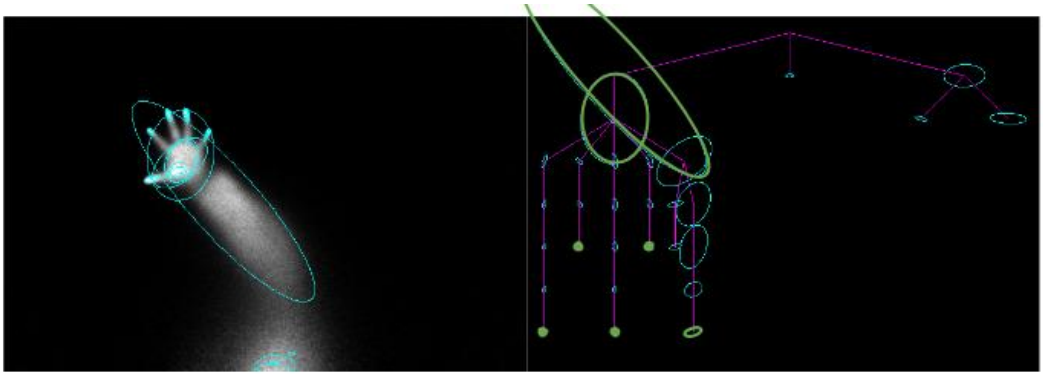


Abbildung 6: Aufbau des Komponentenbaum (rechts) mit zugehörigem Intensitätbildes (links).

Dabei lassen sich sowohl Fingerspitzen, aber auch wie Ewerling at al. vorgestellt haben, Hand- und Armregionen (grün umrandet) erkennen. Um die Visualisierung übersichtlich zu gestalten, wurden die Parameter des MSER-Algorithmus genau an dieses Bild angepasst, so dass nur die relevanten stabilen Regionen zum Vorschein kommen.

Um den MSER-Algorithmus (maximal stable extremal regions) bildlich zu beschreiben, kann man sich das monochrome Bild, welches man von der Kamera bekommt, als Tiefenkarte vorstellen. Man behält die Bildebene bei und definiert den Grauwert jedes einzelnen Pixels als Höhe. So erhält man eine schematische 3D-Struktur des Bildes (siehe Abbildung 7).

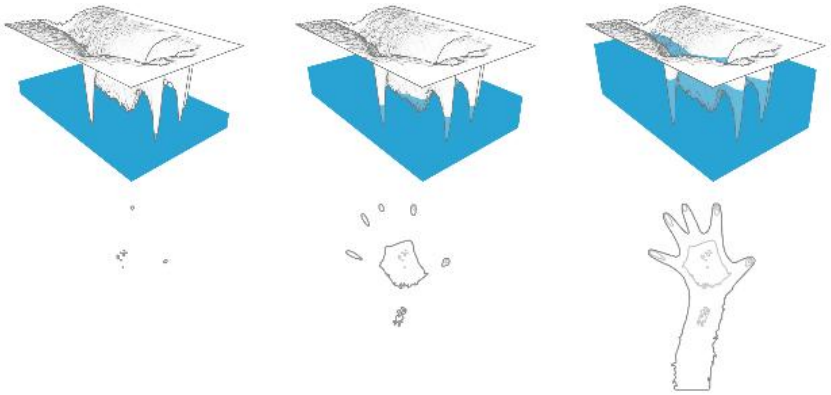


Abbildung 7: Schematische Wirkungsweise des MSER Algorithmus. Siehe [Ewerling et al., 2012]

Wird diese Tiefenkarte gedanklich mit Wasser gefüllt, kommen nach und nach immer mehr Bildteile zusammen und bilden eine gemeinsame Struktur. Zuerst füllen sich die Fingerspitzen, dann quillt das Wasser über und verbindet sich mit der Handfläche. Zuletzt steigt das Wasser Richtung Arm an.
Für ein 8-Bit Bild, das heißt mit 256 verschiedenen Graustufen, steigt das Wasser 255-mal und für jeden Schritt kommen größere Regionen hinzu. Diese Hierarchie wird im Komponentenbaum gespeichert und beschreibt genau die Beziehung der einzelnen Regionen.

Eine sehr effiziente Implementierung dieses Algorithmus wurde von Nister et al. vorgestellt [Nistér and Stewénius, 2008]. Aber auch OpenCV bietet eine Implementierung an, welche aber vorrangig für Texterkennung ausgelegt ist.

2.3 Tracking

In dem Kapitel beschreibe ich den Vorgang des Trackings und welche Lösungsvorschläge bzw. Implementationen es bereits gibt.

Unter dem Begriff Tracking verstehe ich die Zuordnung von Fingern über mehrere Frames. Beim Aufsetzten eines Fingers wird diesem eine ID zugewiesen. Das Ziel eines robusten Trackings ist es diese ID für die gesamte Zeit, die der Finger den Tisch berührt, beizubehalten. Das heißt, es muss unterschieden werden, ob ein neuer Finger hinzukommt, ein aktueller Finger aktualisiert oder ein Finger gelöscht werden muss.

In den folgenden Abschnitten werde ich einige Herangehensweisen erläutern, wie man Blobs von Frame zu Frame tracken kann. Zum besseren Verständnis werde ich die neu erkannten Blobs "Kandidaten" nennen und bereits bekannte Blobs weiterhin als Blobs bzw. Finger bezeichnen.

2.3.1 Kleinste-Distanzen-Tracker (KDT)

Eine triviale Methode ist der Kleinste-Distanzen-Tracker (englisch: minimal distance tracker), welcher in der Arbeit von Wang et al. vorgestellt wird [Wang et al., 2008].
Es wird mit einem bereits vorhandenen Finger angefangen und nach dem nächst gelegenen Kandidaten gesucht. Überschreitet die Distanz zu diesem eine definierte maximale Distanz nicht, wird der Finger geupdatet und der Kandidat aus der Liste gelöscht. Auf dieser Weise wird jeder Finger nach und nach abgearbeitet. Am Ende werden alle nicht geupdateten Finger gelöscht und alle Kandidaten, welche nicht zugeordnet werden können, als neue Finger hinzugefügt.
Solch ein Algorithmus wird als gierig, sogenannte Greedy-Algorithmen, bezeichnet, da sie den anderen Fingern potentielle Lösungen wegnehmen.
Anstatt mit einem zufälligen Finger anzufangen und unkontrolliertes Verhalten zu riskieren, ist es von Vorteil mit dem Finger mit der geringsten Distanz zu einem Kandidat zu beginnen und so Schritt für Schritt weiter vorzugehen.

2.3.2 K-nächste-Nachbarn-Klassifikation (KNN)

Der wohl verbreitetste Algorithmus für Touch-Tracking ist die K-nächste-Nachbarn-Klassifikation (englisch: k-nearest neighbor). K-nächste-Nachbarn gehört zu den einfachsten Maschinelles Lernen Algorithmen und sucht nach den nächsten k Nachbarn.

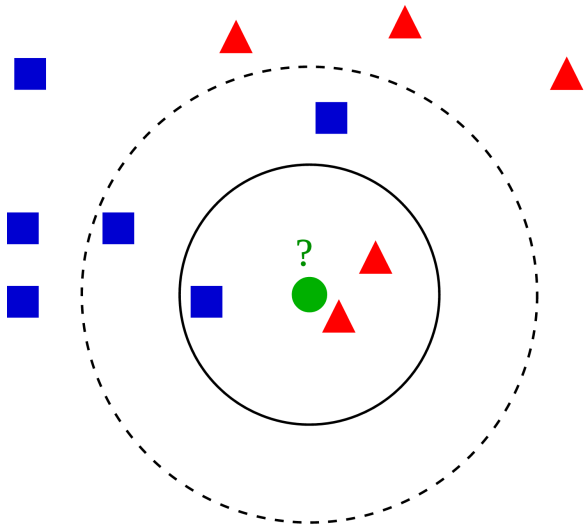


Abbildung 8: Darstellung der KNN Klassifizierung [KNN,].

Angenommen es gibt nur zwei verschiedene Klassen rotes Dreieck und blaues Viereck (siehe Abbildung 8) und ein neuer Kandidat (grüner Kreis) muss einer dieser Klassen zugeordnet werden. So berücksichtigt dieser Algorithmus bei der Klassenzuordnung die k nächsten Nachbarn. Beispielweise für K = 3 befinden sich zwei rote Dreiecke und ein blaues Viereck in der Liste von potentiellen Klassen. Dementsprechend würde der grüne Kreis zu den roten Dreiecken zugeordnet werden.
Anders verhält sich das für K = 5. Hier sind es drei blaue und nur zwei rote Dreiecke, also wird der grüne Kreis den blauen Vierecken zugeordnet.

Dies lässt sich einfach auf das Blob-Tracking Problem übersetzen. Nur das man diesmal für jeden Kandidaten eine variable Anzahl an möglichen Klassen hat. Jeder Kandidat könnte einem aktiven Finger (eine Klasse) zugeordnet werden.
Für K=1 funktioniert KNN auf den ersten Blick ganz ähnlich wie der kleinste Distanzen Tracker. Auch hier wird versucht, jedem Kandidaten den nächsten Finger zuzuordnen. Nur wird er diesmal nicht aus der Liste aller Kandidaten gelöscht, sodass auch mehreren Kandidaten dieselben Fingern zugeordnet werden können. Da am Ende entschieden wird, welcher Finger mit welchem Kandidaten geupdatet wird, kann präferiert vorgegangen und zuerst der Finger mit der kleinsten Distanz aktualisiert werden.

KNN findet keine korrekte Lösung mehr, sobald die Kandidaten nicht am nächsten zu Ihren entsprechenden Fingern sind. Dies hängt stark vor der Ge-

schwindigkeit und der Abtastrate ab, aber auch von der Bewegungsrichtung. Bewegt man sich in Richtung anderer Punkte, wird die Distanz zu diesen immer geringer. Dieses Verhalten lässt sich bei Seitwärtsbewegung mehrerer Finger feststellen. Dadurch erfolgt eine Verschiebung der Zuordnung, wie es in Abbildung 9 deutlich wird.

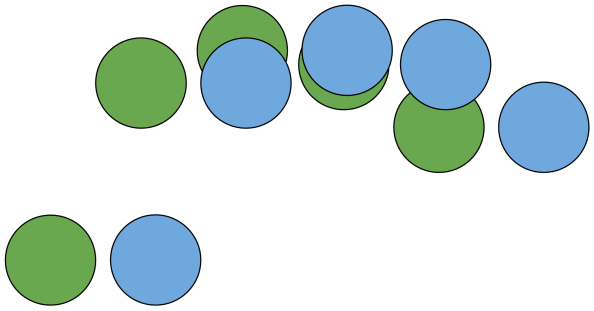


Abbildung 9: Verschiebung der Zuordnungen durch eine Seitwärtsbewegung der Punkte.

Eine Verbesserung des Trackings lässt sich mit der vermuteten Position des Fingers im nächsten Frame erreichen (vgl. [Schöning et al., 2010]). Anstatt die Kandidaten mit der Position aller aktiven Finger zu vergleichen, wird die vermutete Position der Finger für diesen Frame verwendet. Dieses Verhalten gilt natürlich auch für den kleinsten Distanzen Tracker. Um die vermutete Position im nächsten Frame zu bestimmen, bietet sich entweder die triviale Variante an, einfach die letzte Bewegung erneut drauf zu rechnen oder die Verwendung von effektiveren Methoden wie die Doppelte Exponentielle Glättung oder der Kalman Filter [LaViola, 2003].

Problematisch wird das Verwenden der vorhergesagten Position bei abrupt endenden Bewegungen. Die vermutete Position wandert weiter und die Distanz zum eigentlichen Kandidaten wächst und das Tracking wird wieder schwieriger.

2.3.3 Global-Tracker

Eine weitere Methode, welche von einigen Touch-Erkennungs Softwares verwendet wird, verfolgt den Ansatz nicht jeden Blob einzeln mit allen Kandidaten zu vergleichen, sondern zu versuchen das Problem global zu lösen.
Diese Methode ist der in dieser Arbeit von mir Global-Tracker genannte Ansatz. Dieser vergleicht alle aktiven Finger mit allen Kandidaten und sucht die Zuordnung mit den meisten Updates und der geringsten Gesamtdistanz. Die

Idee dahinter ist, dass man den Algorithmus dazu drängt, möglichst viele aktive Finger zu aktualisieren. Also sucht der Algorithmus unter allen möglichen Verbindungen zwischen Finger und Kandidaten die naheliegendste heraus.

Dabei ist es notwendig Bedingungen zu definieren. Einerseits darf die Verbindung von Finger zu Kandidat den definierten maximalen Updateradius nicht überschreiten, andererseits sind die Verbindungen von Fingern zu Kandidaten eindeutig. Einem Finger können nicht mehrere Kandidaten zugeordnet werden. Hier spricht man von einer injektiven Abbildung. Jedem Element einer Menge A wird genau ein Element einer anderen Menge B zugeordnet.
Das Problem bei dem Algorithmus ist, dass alle möglichen Varianten der Verbindungen zwischen Finger und Kandidaten rasant wachsen und der benötigte Rechenaufwand darunter stark leidet. Das Anwachsen der Anzahl der injektiven Abbildungen mit der Eigenschaft $|B| <= |A|$ wird folgendermaßen beschrieben.

$$\frac{|A|!}{(|A| - |B|)!} \tag{2.2}$$

Angenommen es sind fünf Finger aktiv und fünf Kandidaten innerhalb des maximalen Updateradius müssen zugeordnet werden. Dann besitzt der erste Finger fünf potentielle Verbindungen, der zweite nur noch vier, da ein Kandidat bereits vom ersten Finger belegt ist und der dritte kann nur noch drei Kandidaten zugeordnet werden. Diese Permutation Schrittfolge wird solange weitergeführt, bis alle Kombinationen betrachtet wurden. So kommt man auf eine Anzahl von $5! = 120$ möglichen Kombinationen. Die Anzahl der Kombinationen wächst exponentiell an. Bei zehn Fingern und zehn Kandidaten sind das bereits $10! = 3628800$.
Um die Anzahl an Vergleiche zu vermindern ist es ratsam, die möglichen Verbindungen eines Fingers in jedem Permutationsschritt auf eine bestimmte Maximalanzahl, beispielsweise die nächsten zwei Kandidaten zu beschränken. Dadurch wird der Suchraum verringert, aber die naheliegendsten Verbindungen bleiben erhalten. Ein Vergleich der Laufzeiten der beiden Varianten ist in Kapitel 5.4.2 aufgeführt.

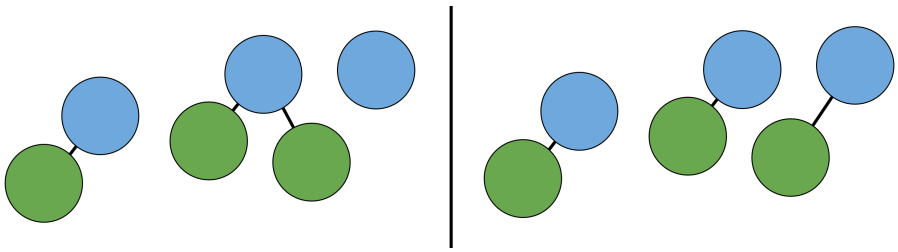
2.3.4 Diskussion

Die Schwierigkeit Blobs mit zunehmender Distanz und steigender Anzahl zu tracken ist nicht bestreitbar. Je größer die Distanz bzw. näher die Finger zu-

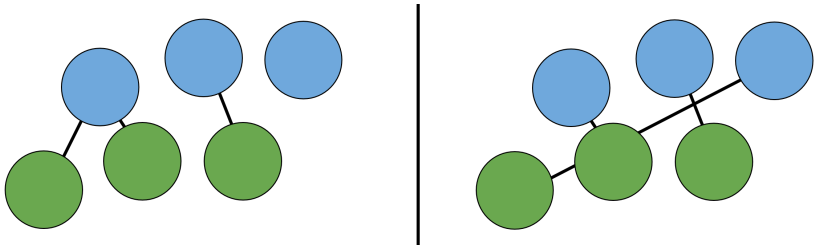
einander, desto fehleranfälliger wird das bisher beschriebene Tracking.

Mithilfe der vorhergesagten Position, wird der Abstand zwischen Kandidat und Finger in der Regel verringert und das Tracking vereinfacht. Natürlich ist die Vermutung aber nur ein Versuch der Annäherung an die Wirklichkeit und garantiert keine korrekte Lösung. Bei abrupten Bewegungen lässt sich keine vorhergesagte Position bestimmen und die Distanz zu den entsprechenden Kandidaten wächst im Gegensatz zur originalen Position. Beide Varianten haben ihre Vorteile: Mit der vermuteten Position werden schnellere aufbauende Bewegungen möglich und mit der originalen Position langsamere aber dafür abrupte Bewegungen. Entweder man limitiert die möglichen Bewegungen auf einen der genannten Fälle oder betrachtet beide Positionen unabhängig voneinander und wählt die Zuordnung mit dem besseren Ergebnis aus.
Hier stellt sich die Frage, wie man das bessere Ergebnis erkennen kann. Das wichtigste Bestreben einer Touch-Software sollte es sein, so viele Punkte wie möglich zu updaten. Aufliegende Finger sollen schließlich auch über den gesamten Zeitraum verfolgt werden können. Deshalb sollte die Zuordnung mit der höheren Anzahl an Aktualisierungen priorisiert werden. Um zwischen unterschiedlichen Lösungsvorschlägen mit gleicher Anzahl an Updates zu unterscheiden, wird, wie beim Global-Tracker auch, anschließend die mit der geringsten Gesamtdistanz aus den Verbindungen ausgewählt.

Diese Art der Betrachtung wird jedoch besonders für den Greedy-Algorithmus gefährlich. Bei KNN kann es vorkommen, dass einem Finger zwei Kandidaten zugeordnet werden, bei dem minimalen Distanzen Tracker nicht. So wird bei KNN nur mit dem nähere Kandidaten geupdatet und ein neuer Finger für den anderen Kandidaten erstellt. Der gierige Algorithmus hingegen wird anschließend wieder nach dem nächsten Finger suchen und insgesamt zwar in der Regel mehr Finger aktualisieren aber dafür auch mit erhöhter Wahrscheinlichkeit falsch. Folgende Darstellung 10 zeigt das unterschiedliche Verhalten bei KNN (Links) und dem minimalen Distanzen Tracker (Rechts) bei jeweils gleichen Positionierungen 2.10(a) und 2.10(b) der aktiven Finger und Kandidaten. In der Darstellung sind die aktiven Finger grün und die Kandidaten blau visualisiert. Dieses Farbschema wird über die gesamte Arbeit beibehalten.
Im oberen Teilabschnitt 2.10(a) kommt der Greedy-Algorithmus auf die bessere Zuordnung, aber wie im unteren Bereich 2.10(b) zu sehen ist, führt diese Vorgehensweise gleichzeitig zu großen unerwünschten Sprüngen der Finger.



(a) Fehlerhafte Zuordnung der Punkte über KNN (links) und korrekt Zuordnung über KDT auf der rechten Seite.



(b) Fehlerhafte Zuordnung der Punkte über KNN (links) und falsche Zuordnung der Punkte über KDT (rechts)

Abbildung 10: Vergleich der Zuordnungen von KNN und KDT bei ausgewählten Fingerpositionen.

Nur der Globale-Tracker kommt bei den oberen Beispielen auf das korrekte Ergebnis. Er ist vergleichsweise weniger stark von dem Abstand zwischen Finger und zugehörigen Kandidaten abhängig, dafür bleibt hier erhöhter Rechenaufwand nicht aus.

2.4 Software Varianten

In diesem Kapitel erwähne ich Referenz-Software, welche auf Basis von Diffuser Illumination arbeiten und Touch-Punkte erkennen und tracken.

Die Touch-Erkennung durch Diffuse Illumination ist nicht sehr verbreitet. Das benötigte Setup nimmt zu viel Platz ein, ist zu stark lichtabhängig und das Erkennen der Punkte ist fehleranfällig. Dennoch gibt es einige Open Souce Software Lösungen, die das Einrichten eines solchen Setups stark vereinfachen und die nötige Softwarestruktur bereitstellen.

2.4.1 Community Core Vision (CCV)

Die Community Core Vision ist ein Open Source Projekt der NUI Group Community. Die aktuelle und von mir beschriebene Version ist die Version 1.5.

Früher hieß die Software tbeta, wurde aber mit ihrer Veröffentlichung jedoch umbenannt. CCV ist plattformunabhängig und basiert auf OpenFrameworks. Bereits von Haus aus wird eine Reihe von Kameras unterstützt, aber es ist auch möglich, ein Video als Input einzuladen.
Mithilfe eines Hochpass Filters wird das Bild binarisiert und die Blobs erkannt. Ein optionaler Amplify-Filter wird angeboten, um den Kontrast des Bildes anzupassen und somit die Erkennung zu verbessern.
Das anschließende Tracking erfolgt über den einfachen KNN mit vermuteter Position.
Ansonsten bietet die Community Core Vision viele zusätzliche Funktionen, wie das Kalibrieren des Tisches, um die Punkte genau zu positionieren und unterstützt für große Hardwarelösungen auch mehrere Kameras gleichzeitig.

2.4.2 reacTIVision

ReacTIVision wurde von Martin Kaltenbrunner und Ross Bencina an der Pompeu Fabra Universität in Barcelona entwickelt. Hierbei handelt es sich um ein, ebenfalls plattformunabhängiges, Computer Vision Framework, welches vorrangig das Tracken von definierten Markern unterstützt. Seit der Version 1.4 dient es auch dem Finger-Tracking [Kaltenbrunner, 2009]. Dabei werden die Blobs, wie auch bei der CCV, über ein binarisiertes Bild erkannt und anschließend getrackt.
Da das Framework aber noch immer in erste Linie auf das Erkennen und Tracken von Markern, sogenannten “Fiducial Symbols” ausgelegt ist, ist die Verwendung von reacTIVision als robuste Touch-Tracking Software eher ungeeignet. Nach Qian Liu gestaltet sich das Einrichten der Software um sowohl Fiducial Marker, als auch annähernd stabil Finger zu tracken, als sehr schwierig und eignet sich dadurch weniger für reine Touch-Anwendungen [Liu, 2010].

2.4.3 TouchLib

TouchLib ist eine Multi-Touch Bibliothek und stammt ebenfalls von der NUI Group Community. Die Bibliothek wurde in C++ implementiert und läuft vorrangig unter Windows [Touchlib,].
Mithilfe der TouchLib Bibliothek kann der User relativ einfach seine eigene Touch-Applikation bauen.
Dabei wird das Erkennen von Blobs unterstützt und das Tracken erfolgt über die Minimierung der globalen Distanz. Um das Performance Problem dabei zu lösen, wird die Anzahl der möglichen Verbindungen zu anderen Blobs anzahlabhängig limitiert.
Folgende Übersicht zeigt die Limitierung der Verbindung Anhand der Anzahl

an aktiven Fingern (cursize).

- if (cursize <= 4) → lasse 4 Verbindungen zu
- if (cursize <= 6) → lasse 3 Verbindungen zu
- if (cursize <= 10) → lasse 2 Verbindungen zu
- sonst → lasse 1 Verbindung zu

Das heißt, sobald mehr als zehn Finger aktiv sind, wird nur noch über die geringste Distanz zugeordnet.

2.4.4 Microsoft PixelSense

Das wohl bekannteste Touch-Display, welches mit Infrarot Bildern arbeitet und zum Verkauf angeboten wird, ist der Multi-Touch-Tisch PixelSense von Microsoft. Dieser kam 2007 auf den Markt und wurde damals noch unter dem Namen Microsoft Surface bekannt. Mit der Veröffentlichung der zweiten Generation im Jahr 2012, dem SUR40, wurde der Touch-Computer in Microsoft PixelSense umbenannt.
Die Berührungs-Erkennung wird über mehrere Infrarot LED-Lichtquellen und fünf Kameras realisiert und kann laut Microsoft bis zu 52 Objekte gleichzeitig erfassen. Der Microsoft-Evangelist Lutsch sagt dazu: „Wir geben 52 als Limit an, weil das System bei mehr Objekten langsamer wird. Theoretisch sind aber mehr Objekte gleichzeitig möglich – auch wenn so viele Leute gar nicht um den Tisch passen“ [ct:, 2008].
Die Blob-Erkennung erfolgt über die Umrisse von Objekten. Dabei wird nicht unterschieden, ob es ein Finger, eine komplette Hand oder nur der Umriss einer abgestellten Flasche ist. Zusätzlich können über sogenannte Tags Objekte identifiziert und lokalisiert werden.
Bei informellen Experimenten mit dem Microsoft PixelSense Interaktionstisch am Lehrstuhl für Mobile Media waren Probleme der Fingererkennung und -verfolgung zu beobachten.

Kapitel 3

Eigener Lösungsansatz

Im Folgenden möchte ich einen neuen Ansatz beschreiben, um Fingerbewegungen zu tracken.
Die Hauptmotivation das Trackingproblem zu lösen, ziehe ich aus der Erkenntnis, dass die mir bekannten Systeme viele zusätzliche Informationen vernachlässigen. So wird von den Systemen bisher nur die Position der Finger betrachtet.

Nachfolgend ist die gleiche Positionierung der Finger des linken und mittleren Bild vom Anfang (Abbildung 4 S. 10) dargestellt. Nur wird diesmal zusätzlich das Kamerabild betrachtet.

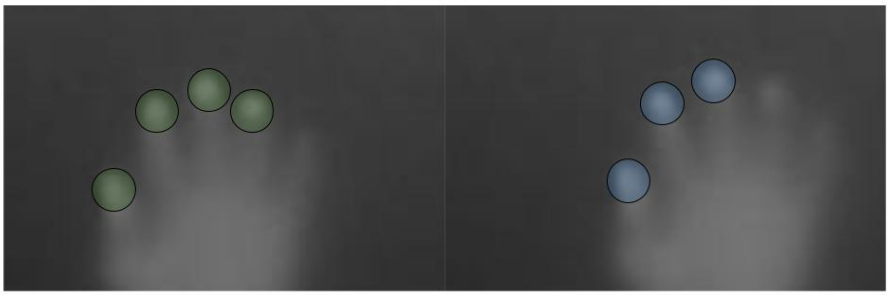


Abbildung 11: Vereinfachtes Tracking durch Bildinformation.

Visuell betrachtet ist die richtige Zuordnung nun auch bei großer Distanz kein Problem mehr, da unser Gehirn instinktiv nicht nur die Positionen der Finger zuordnet, sondern weitaus mehr Informationen nutzt. Diese Herangehensweise versuche ich in meiner Arbeit zu reproduzieren.

Finger sind an anatomische Bedingungen geknüpft, welche sich hervorragend

nutzen lassen, um die Erkennung und die Zuordnung zu stabilisieren. Wieso wird nur die Position der Finger getrackt? Finger haben noch andere Eigenschaften, wie Richtung, Handzugehörigkeit bzw. die Relation zu den anderen Fingern, die hervorragend zum Stabilisieren des Trackings beitragen können. Ein Finger wird immer derselben Hand zugehören. Des Weiteren wird sich die Position zu den anderen Fingern von Frame zu Frame nur bedingt ändern.

Um genau diese Handzugehörigkeit der Kandidaten herauszufinden, mache ich mir die Informationen aus dem Kamerabild zu Nutze und fasse die erkannten Kandidaten zu sogenannten Clustern zusammen.

3.1 Clustering-Methoden

Clustering beschreibt das Gruppieren von Kandidaten zu einer Hand. Dabei habe ich unterschiedliche Ansätze ausgearbeitet. Grundsätzlich lässt sich die Handzugehörigkeit zum Beispiel über die Fingerrichtung erkennen. Entdeckt man die Fingerspitzen und spannt entlang der Fingerrichtung eine Linie auf, so gehören Linien, welche sich im Rahmen einer bestimmten maximalen Distanz kreuzen, zu einer Hand (siehe Bild 12).

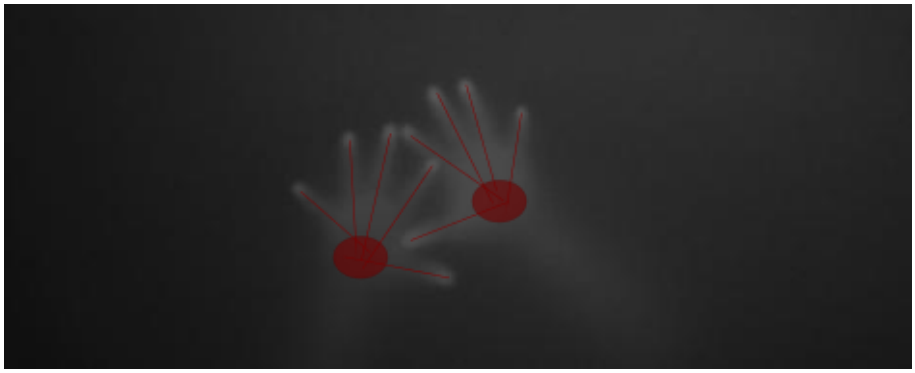


Abbildung 12: Handzuordnung über Fingerrichtung.

Um die Fingerrichtung zu bestimmen wird sich der Helligkeitsverlauf des Bildes zunutze gemacht. Folgend beschreibe ich bekannte Herangehensweisen, mit denen man die Fingerrichtung bestimmen kann und werde anschließend zwei weitere Varianten von mir vorstellen.

3.1.1 Kontur

In der Veröffentlichung [Dang and André, 2011] wird die Erkennung der Fingerrichtung über die Kontur der Blobs realisiert: Zuerst wird das Bild binarisiert und anschließend die Richtung der ellipsenförmigen Konturen (siehe Abbildung13) berechnet. Aus der Hauptdiagonalen einer Ellipse lassen sich zwei potentielle Richtungen bestimmen. Für die Fingerrichtung wird die Version mit dem geringeren Helligkeitsabfall verwendet.

Abbildung 13: Handzuordnung über die Kontur der Finger (Siehe [Dang and André, 2011]).

Wie auch Dang et al. feststellten, ist es nicht ratsam, sich nur auf ellipsenförmige Konturen zu verlassen [Dang and André, 2011]. Im binarisierten Bild kommt es häufig zu kreisförmigen Konturen, welche auftreten, sobald die Finger die Tischplatte fast senkrecht berühren oder der Threshold für diesen Finger zu klein ist und deshalb nur die reine kreisförmige Fingerspitze hervorgehoben wird.
Um den Fall von kreisförmigen Konturen abzudecken, beschreiben Dang et al. auch eine andere Herangehensweise über die äußere Kontur der Finger (siehe Abb. 14).

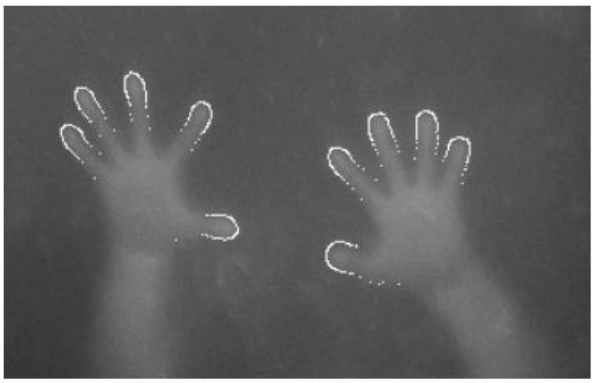


Abbildung 14: Fingerrichtungsbestimmung über äußere Kontur nach Dang et al.

Dabei wird zuerst die äußere Kontur bestimmt, anschließend die annähernd parallelen Linien der Finger erkannt und der Durchschnitt der Winkel dieser Linien als Fingerrichtung verwendet.
Dafür müssen die Hände allerdings stark gespreizt sein, sodass die äußere Kontur der Finger auch sichtbar ist. Die Autoren [Dang and André, 2011] sprechen von einer Erfolgsrate von 70-80 % .

3.1.2 Gradient

Eine weitere potentielle und bisher noch nicht getestete Möglichkeit von mir ist es, die Fingerrichtung zu bestimmen, indem man von erkannten Blobs die Richtung mit dem geringsten Helligkeitsabfall sucht. Oder algorithmischer ausgedrückt: Suche in diskreten Schritten radial um den Finger den, zum vorherigen Punkt nächsten, dunkleren Pixel und bestimme die Richtung durch den Mittelwert aller bestimmten Punkte.
Der Nachteil dieser Methode ist, dass sie sehr stark von der Abtastrate abhängt. Im Bild 15 ist die Problematik der Abtastrate dargestellt.

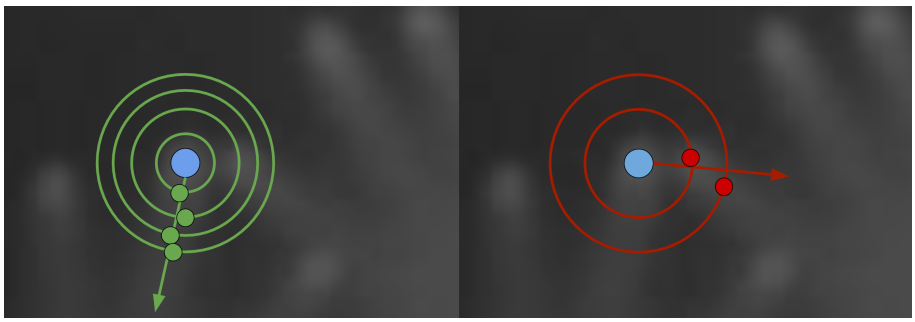


Abbildung 15: Bestimmung der Fingerrichtung in radialen Schritten über geringsten Helligkeitsabfall. Links mit ausreichender Abtastrate. Rechts zu kleine Abtastrate und falscher Bestimmung der Fingerrichtung.

Ist die Abtastrate nicht groß genug (siehe rechtes Bild), gehen schnell Informationen verloren. In dem Beispiel wird der dunkle Teil zwischen zwei nahen Fingern ignoriert und deshalb die Fingerrichtung in Richtung des anderen Fingers vermutet. Verringert man die Abtastrate, wird man jedoch die korrekte Richtung erkennen, muss aber mit erhöhter Prozessierzeit rechnen.

3.1.3 Komponentenbaum

Eine weitere Möglichkeit, welche implizit ebenfalls den Helligkeitsabfall und somit die Fingerrichtung betrachtet, ist die Verwendung des Komponentenbaumes vom MSER. Man kann jede Region anhand seiner Helligkeit, seiner Größe, sowie dem Verhältnis zu den Kinder bzw. Vaterknoten zu potentiellen Fingern, Händen oder Armen klassifizieren.
Traversiert man den klassifizierten Komponentenbaum, müsse es über jeden Finger theoretisch eine Vaterregion geben, welche auf die Handfläche verweist oder in deren Nähe liegt. Über jeder Hand bzw. Handfläche muss es auch eine größere Region geben, die den Arm referenziert.
So kann die Richtung der erkannten Fingerspitzen über deren Vaterregion ermittelt und die Handzugehörigkeit über die hierarchische Struktur des Komponentenbaums bestimmt werden.

3.2 Hand-Tracking

Durch das Clustering wird auch das Tracking deutlich einfacher. In diesem Kapitel beschreibe ich den Zusammenhang von Clustering und Tracking und wie man dies miteinander verbinden kann.

Angenommen die erkannte Handzuordnung der Kandidaten stimmt, muss man nicht mehr jeden Finger mit jedem Kandidaten des neuen Frames vergleichen, sondern es reicht, die aktuellen Hände mit den neu ankommenden Clustern zu vergleichen. Dies verringert den Suchraum vom Tracking deutlich. Anstatt alle Finger mit allen Kandidaten zu vergleichen, reicht es die jeweiligen Finger mit den jeweiligen Kandidaten des Clusters zu tracken.
Die Hand - Die Cluster Zuordnung kann unproblematisch über die kleinste Distanz bestimmt werden. Hände sind rein anatomisch zwangsweise größer als Finger und können nicht so dicht zusammen liegen, dass eine Zuordnung problematisch wäre.
Der Ansatz dahinter ist schematisch in folgender Abbildung dargestellt.

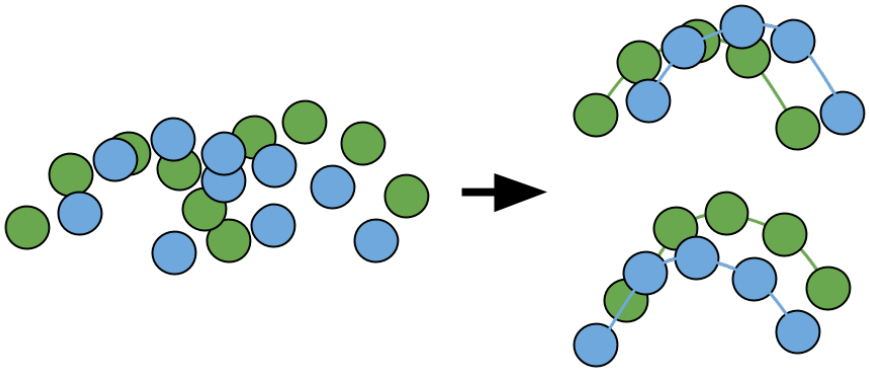


Abbildung 16: Vergleich von positionsgebundenem Tracking-Ansatz (links) mit der des Hand-Tracking Ansatzes (rechts).

Auf der linken Seite sieht man das bisherige Szenario, bei dem zehn Positionen mit zehn weiteren verglichen werden. Da ist es sogar für uns Menschen nur schwer möglich, die richtige Zuordnung zu finden. Betrachtet man diese stattdessen separat im Verbund (rechte Seite), wird es deutlich einfacher, die Zuordnung der Finger zu bestimmen.

Anhand des Bildes kann man das Potential der Methode gut erkennen. Diese Erkenntnis eröffnet ein ganz neues Spektrum an Tracking Varianten. Betrachtet man die Hand als Graph, so kann diese als Wurzelknoten und darunter liegende Finger als Kinderknoten erkannt werden. So lassen sich Graph-Tracking-Methoden anwenden. Stellt man sich dagegen die Hand als approximiertes Skelett vor, so ist es sinnvoll die Skelett-Tracking-Methoden anzuwenden. Da sich Tracking-Methoden stark von Erkennungsfehlern beeinflussen lassen und weder die Fingerrichtung, noch die Handfläche sich wirklich zuverlässig erkennen lassen, greife ich nur auf die stabile Fingerposition zu und betrachte die

Handstruktur als verbundene Punktwolke.

Ein weiterer Vorteil ganze Handstrukturen miteinander zu vergleichen ist, dass man die neue Position der Hand kennt und somit die Richtung, in die sich die Hand bewegt hat, abschätzen lässt. Transformiert man alle Finger dieser Hand in diese Richtung, garantiert dies bei korrekter Erkennung der Cluster Position eine deutliche Verminderung der Distanzen zu den jeweiligen Kandidaten. Dies ist deutlich effektiver, als eine nur vermutete Position der Finger zu verwenden, da sie auch bei abruptem Richtungswechsel eine korrekte Translation garantiert.

Als eine mögliche Variante, Handstrukturen zu tracken, bediene ich mich dem Iterative Closest Point Algorithmus, welchen ich in folgendem Kapitel genauer beschreiben werden.

3.2.1 Iterative Closest Point Algorithmus (ICP)

Ein Algorithmus, der im Bereich der Registrierung von Punktwolken in der Literatur immer wieder auftaucht, ist der Iterative Closest Point Algorithmus. Der Algorithmus kommt ursprünglich aus der 3D Rekonstruktion und geht auf [Besl and McKay, 1992] zurück. Das Ziel des Algorithmus ist es, die beste Transformation zweier Punktwolken zu finden, um die Distanz zwischen den Punkt-Verbindungen zu minimieren und so eine Punktwolke in eine Referenzpunktwolke einzupassen.

Dieser Ansatz lässt sich auf das vorliegende Tracking-Problem übertragen. Wir betrachten sowohl die Finger der Hand, als auch die Kandidaten des aktuellen Clusters als verbundene Punktwolke und ermitteln die beste Transformation, um die eine Punktwolke so genau wie möglich auf die andere zu transformieren. Anschließend ist die Zuordnung trivial und kann über die geringste Distanz erfolgen.

Ein Vergleich vom Tracking-Verhalten einmal mit Transformation der Punktwolke und einmal ohne Transformation ist in folgender Veranschaulichung 17 dargestellt.

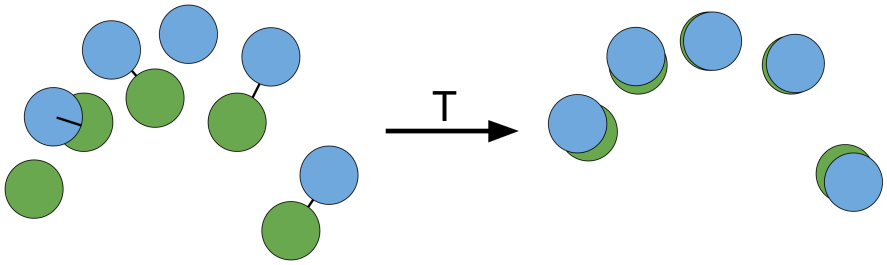


Abbildung 17: Erleichtertes Tracking vor (links) und nach (rechts) einer erfolgreichen ICP Transformierung.

Im linken Teilbereich des Bildes sieht man die fälschliche Zuordnung über KNN und daneben die triviale Zuordnung nach der erfolgreichen Transformation.

Da sich die Finger zwangsläufig mit der Hand mit bewegen und sich die Relation der Finger von Frame zu Frame nicht stark ändert, kann man die Hand als "ridgid body"-Objekt betrachten und nur nach der Translation und Rotation suchen. Dadurch können affine Transformationen, wie Skalierung oder Verzerrung vernachlässigt werden und das Ergebnis wird dadurch robuster.

Die Abfolge von ICP lässt sich in folgende Punkte unterteilen:

1. Suche für jeden Punkt der einen Punktwolke den nächste Punkt der anderen Punktwolke (KNN)

2. Berechne die Transformation, die den quadratischen Abstand zwischen den korrespondierenden Punkten minimiert

3. Transformation der Punktwolke

4. Wiederhole Schritt 1-3 bis zur "optimalen" Lösung

Ignoriert man eine mögliche Rotation der Hand, ist die Realisierung des ICP Algorithmus sehr einfach. Man akkumuliert den Richtungsvektor jedes einzelnen Punktpaares und dividiert anschließend durch die Gesamtzahl der Paarungen, um den durchschnittlichen Richtungsvektor zu bestimmen. Anschließend wird die gesamte Punktwolke um diesen Vektor verschoben.
Bei einer ausgeprägten Rotationsbewegung scheitert diese Form der Translation jedoch.
Um die optimale ridgid Transformation zu bestimmen, muss die Rotation R und die Translation t bestimmt werden, sodass die Punktwolke A der aktiven Hand so gut wie möglich auf die Punktwolke B des Clusters passt.

$$B = R * A + t \tag{3.1}$$

Dafür müssen folgende Punkte abgearbeitet werden (Vgl. [Ho,]):

1. Das Finden der Zentroiden beider Punktwolken
2. Die Verschiebung der Punktwolken in den Ursprung
3. Die Bestimmung der Rotation
4. Die Bestimmung der Translation

Um Rotation und Translation zu bestimmen, werden zunächst die Zentroiden beider Punktwolken festgelegt und jede Punktwolke verschoben, sodass sich dieser Zentroid im Ursprung befindet.
Die Zentroiden sind die Mittelpunkte der Wolken und lassen sich folgendermaßen berechnen.

$$Zentroid_A = \frac{1}{N} \sum_{i=1}^{N} P_A^i \tag{3.2}$$

$$Zentroid_B = \frac{1}{N} \sum_{i=1}^{N} P_B^i \tag{3.3}$$

P_A und P_B sind die Punkte der jeweiligen Punkwolken A und B. Anschließend transformiert man diese zum Ursprung und sucht die Rotation R, wie es in folgender Abbildung dargestellt ist.

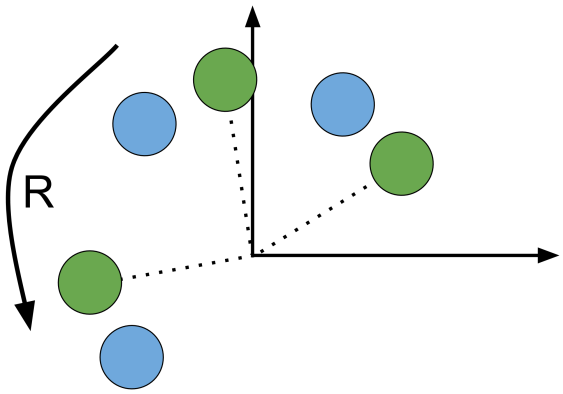


Abbildung 18: Bestimmung der Rotation zweier Punktwolken im Ursprung (vgl. [Ho,]).

Um die bestmögliche Rotation von A auf B zu bestimmen, lösen wir die Formel 3.4 mit Hilfe der Singulärwertzerlegung (SVD).

$$B = R * A \tag{3.4}$$

Die SVD zerlegt eine Matrix E in ein Produkt aus drei speziellen Matrizen, sodass sich die Singulärwerte von E ablesen lassen (siehe 3.5).

$$\begin{aligned} [U, S, V] &= SVD(E) \\ E &= USV^T \end{aligned} \tag{3.5}$$

Damit lässt sich z. B. die Lösung mit dem kleinsten quadratischen Fehler eines linearen Gleichungssystems berechnen. Um dies auf unsere Problematik anzuwenden, bestimmen wir vorerst die Kovarianzmatrix H, welche wir mit Hilfe der SVD zerlegen. Durch die Singulärwertzerlegung der Matrix H lässt sich die Rotation folgendermaßen ermitteln.

$$\begin{aligned} H &= \sum_{[i=1]}^{N} (P_A^i - Zentroid_A)(P_B^i - Zentroid_B)^T \\ [U, S, V] &= SVD(H) \\ R &= V * U^T \end{aligned} \tag{3.6}$$

Im Anschluss muss nur doch die Translation t gelöst werden, indem man den Zentroiden von A um den berechneten Winkel rotieren lässt und anschließend den Richtungsvektor zum Zentroiden von B ermittelt (Formel 3.7).

$$t = Zentroid_B - (R * Zentroid_A) \tag{3.7}$$

Weiterführend können wir jeden Punkt der zu transformierenden Punktwolke mit der Transformations Matrix T multiplizieren und erhalten den neuen transformierten Punkt. Die Transformationsmatrix T setzt sich aus R und t zusammen. R ist eine 2x2 Rotationsmatrix und t ein 2x1 Translationsvektor. Konkatenieren wir R und t miteinander erhalten wir für T = [R— t] eine 2x3 Matrix. Um Rotation und Translation miteinander zu verbinden und die Multiplikation in einer Operation zu gewährleisten, müssen wir in homogenen Koordinaten rechnen. Das heißt, dass sowohl der 2D Punkt, als auch unsere Transformations Matrix, eine weitere Zeile benötigen. So erhalten wir einen 2D Punkt in homogenen Koordinaten, folglich einen 3x1 Vektor. Um diesen problemlos mit der Matrix T (3x3) zu multiplizieren, müssen wir ihn vorerst noch transponieren. Das Ergebnis ist ein transformierter 1x3 Vektor, welcher sich wieder in euklidische Koordinaten umwandelt lässt.

$$P^T_{A_{neu}} = P^T_A * T \tag{3.8}$$

Kapitel 4

Softwareumsetzung

Da es keine öffentliche Softwarelösung gibt, welche die in der Arbeit gestellten Anforderungen erfüllt, galt es eine eigene Software zu entwickeln und den Hand-Tracking Ansatz umzusetzen. Die in dieser Arbeit eingeführte und als 'Iterative-Hand-Tracking' benannte Software ist in C++11 geschrieben und nutzt OpenCV 3.0. Die Benutzeroberfläche wurde mithilfe von QT 4.8 umgesetzt. Die Software wird aktuell nur unter Linux verwendet.

4.1 Programmablauf (Pipeline)

In jedem Frame wird eine Software-Pipeline durchlaufen. Dieser Prozess ist in der nachfolgenden Abbildung vereinfacht schematisch dargestellt und wird vorerst grob beschrieben und im Abschnitt 4.3 und 4.4 in seinen Schritten detailliert erläutert.

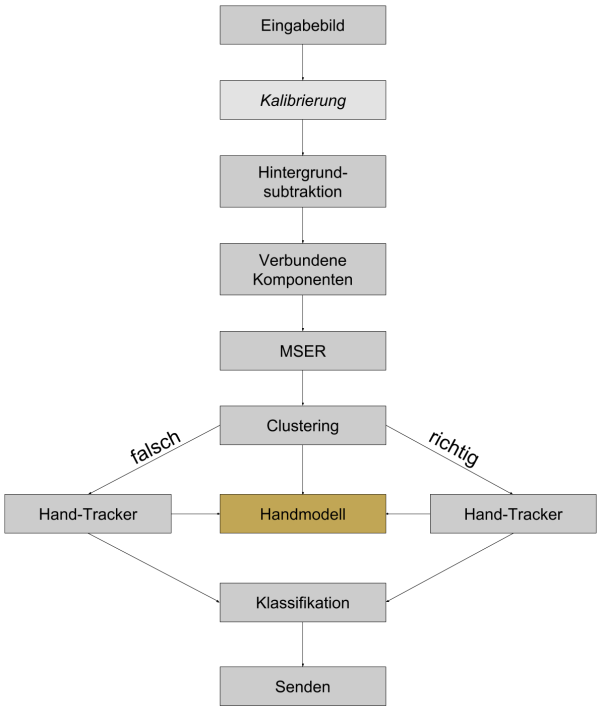


Abbildung 19: Programmablauf in jedem Frame.

Alle Bildverarbeitungsschritte sind im Appendix auf Seite 85 visuell aufgeführt.

Als Eingabebild kommen alle von OpenCV 3.0 unterstützen Kameras (cv::VideoCapture), sowie die PointGrey Grasshopper 3 Kamera (siehe Abb. A.37(a)), ein Video File in gängigen Formaten, als auch ein Ordner mit Bildern in Frage. Kommt das Bild von einer Kamera muss dieses erst kalibriert werden. Bilderfolgen oder Videos müssen in kalibrierter Form eingeladen werden. Das kalibrierte Bild wird für den weiteren Verlauf vorbereitet (Hintergrundsubtraktion (siehe Abb. A.37(b)), Verbundene Komponenten (siehe Abb. A.37(c))), um anschließend über den MSER Algorithmus einen Komponentenbaum zu erstellen (siehe Abb. A.37(d)) und Finger und deren Handzugehörigkeit (Clustering (siehe Abb. A.37(e))) zu ermitteln. Im Anschluss werden die Cluster mithilfe des Hand-Tracking Ansatzes getrackt und zur Klassifikation weitergegeben, wo sowohl die Hände als auch Finger kategorisiert werden (siehe Abb. A.37(f)), um letztendlich an die Anwendung geschickt zu werden.

Das Diagramm 20 zeigt die benötigte Zeit eines jeden Pipelinschrittes.

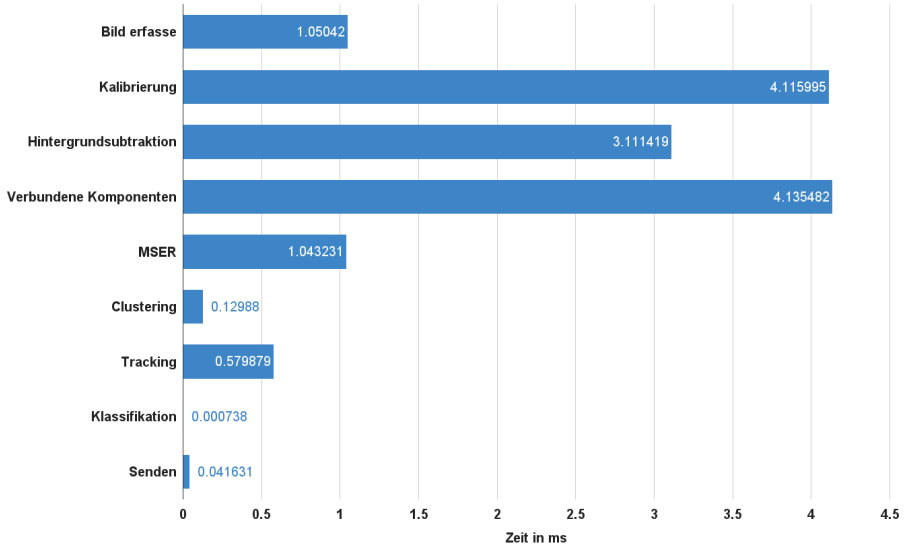


Abbildung 20: Benötigte Zeit die jeder Schritt der Pipeline (Abb. 19) durchschnittlich benötigt.

Dabei wurde die jeweilige Rechendauer eine Minute lang gemessen und der Durchschnitt berechnet. Während der gesamten Zeit wurden zwei Hände mit jeweils fünf Fingern bei 60Hz bewegt. Die derzeitige verwendete Softwarekonfiguration wird in den nachfolgenden Kapiteln beschrieben.
Deutlich zu erkennen ist, dass die meiste Zeit für alle Bildverarbeitungsprozesse, wie Kalibrierung, Hintergrundsubtraktion und Verbundene Komponenten benötigen wird. Die Kameraansteuerung und der MSER-Algorithmus benötigen jeweils ca. 1 ms. Die anschließende Auswertung der extrahierten Informationen im Clustering, sowie das Tracken der Hände benötigt nur ein Bruchteil der benötigten Zeit für die Bildverarbeitung. Die Klassifizierung und das Senden der Hände kann mit durchschnittlich 467 und 21981 Nanosekunden vernachlässigt werden.
Die benötigte Gesamtzeit aller Pipelineprozesse beträgt pro Frame im Durchschnitt 14,02 ms. Somit sind beim aktuellen Stand bei zehn aufliegenden Fingern theoretische 71,32 FPS möglich.

4.2 Klassendiagramm

Der Aufbau der verwendeten Klassen wird im Klassendiagramm deutlich. Zur Übersichtlichkeit befindet sich dieses im Anhang der Arbeit.
Dabei lehnt sich die Struktur stark an den oben beschriebenen Ablauf an. Die

Klasse ImageProcessing hält eine Referenz auf jede in der Pipeline erwähnte Klasse und die Pipeline wird Schritt für Schritt abgearbeitet.

4.3 Aufbau (Hauptbestandteile)

4.3.1 M(S)ER

Ich nutzte die Implementierung des MSER mit linearer Zeit von David Nistér und Henrik Stewénius. Dabei wurde der Code an meine Ansprüche angepasst. Anstatt nur die stabilen Regionen zu verwenden und die hierarchische Struktur zu ignorieren, habe ich den Code so angepasst, sodass ein Pointer auf dem Wurzelelement des Komponentenbaumes zurückgegeben wird. Dabei wurden zur weiteren Betrachtung auch die instabilen Regionen extrahiert. Dies vereinfacht die anschließende Erkennung, da der komplette Komponentenbaum weniger anfällig gegenüber Veränderungen ist.

4.3.2 Handmodell

Ich betrachte eine Hand als anatomisches Objekt, mit bestimmten Bedingungen, welche es festzulegen gilt. Bevor ich zu den Themen Clustering und Tracking komme, beschreibe ich mein Handkonstrukt, um anschließen darauf aufbauen zu können.

Meine Software verwendet für die Klassifizierung von Fingern, Händen und Armen, sowie für das Clustern, ein Handmodell, welches ihre physikalischen und anatomischen Verhältnisse beschreibt. Außerdem wird im Handmodell die maximale Bewegungsgeschwindigkeit für das Tracking definiert.

Alle Werte sind in Zentimetern angegeben und werden beim Programmstart über die Tischgröße, welche in einer Konfigurationsdatei definiert ist, und die Auflösung des Bildes, in Pixel Werte umgerechnet. So sind sie dynamisch auf jedes System übertragbar. Alle Werte wurden von mir persönlich festgelegt und entsprechen keiner biologischen Studie.

Die folgende Tabelle beschreibt alle Werte und erklärt, wie meine Software diese interpretiert und verwendet.

Parameter	Erklärung	Wert
ArmAreaMin	Beschreibt die minimale Fläche für Arme in cm2.	85
ArmAreaMax	Beschreibt die maximale Fläche für Arme in cm2.	100
HandAreaMin	Beschreibt die minimale Größe von Handflächen in cm2. Diese muss sehr klein gewählt werden, da bereits über dem Touch-Bereich schwebende Hände bzw. Handflächen erkannt werden sollen.	
HandAreaMax	Beschreibt die maximale Größe von Handflächen in cm2.	85
FingerAreaMax	Beschreibt die maximale Fläche für Finger in cm2. Eine minimale Fläche ist für Finger nicht vorgesehen. Dieser Wert repräsentiert nur die Fingerspitze und ist deshalb klein gewählt.	1
H2A_DistanceMax	Beschreibt die maximale Distanz zwischen Hand und Arm. Wird für die Arm Zuordnung zur Hand benötigt.	15
H2H_DistanceMin	Beschreibt den minimalen Abstand zwischen zwei Händen. Nur wenn diese Distanz überschritten wird, wird eine neue Hand erkannt.	5
F2H_DistanceMin	Beschreibt den minimalen Abstand zwischen Finger und zugehörigen Hand. Da für eine sehr steil aufliegende Hand der zweidimensionale Abstand sehr gering werden kann, ist dieser Wert klein gewählt.	2
F2H_DistanceMax	Beschreibt den maximalen Abstand zwischen Finger und Hand. Dieser Wert wird größer gewählt als anatomisch möglich, da die Handfläche manchmal nicht richtig erkannt wird. Dieser Wert wird verwendet, um zu entscheiden, ob das gefundene Cluster einer anatomischen Hand entsprechen könnte.	10
F2F_DistanceMin	Beschreibt den minimalen Abstand zwischen Fingern. Wird dieser Abstand unterschritten werden diese Regionen zusammengefasst und nur ein Finger erkannt. Dieser Wert ist nötig, da es vorkommen kann, dass zwei Fingerregionen unter einem Finger existieren. Diese werden durch diesen Wert zusammengefasst.	0,5

Tabelle 4.1: Beschreibung der Werte meines definierten Handmodells

Parameter	Erklärung	Wert
F2F_DistanceMax	Beschreibt den minimalen Abstand zwischen zwei Fingern einer Hand. Dieser Wert wird nur dazu verwendet, um zu schauen, ob das erkannte Cluster einer realen Hand entsprechen könnte.	23
F2F_AngleMin	Beschreibt den minimalen Winkel zwischen zwei Fingern zu der Hand. Dient nur dazu, die Richtigkeit eines Clusters zu bestimmen.	1
HandSpeedMaxPerS	Beschreibt die maximale Geschwindigkeit einer Hand in cm pro Sekunde. Dieser Wert ist sehr hoch gewählt, da mit falsch erkannten Handregionen gerechnet werden muss. Der Wert wird automatisch auf die aktuelle Framerate runtergerechnet.	500
HandRotationMaxPerS	Beschreibt die maximale Rotation einer Hand in Grad pro Sekunde. Zwar ist es nicht möglich, seine Hand innerhalb einer Sekunde um 400° zu drehen, aber da dieser Wert automatisch auf die Framerate heruntergerechnet wird, entspricht das bei 60 Hz einer maximal Rotation von $\tilde{7}$° pro Frame. Dieser Wert wird verwendet, um den ICP Algorithmus robuster zu gestalten und unmögliche Rotationen herauszufiltern.	400
BlobSpeedMaxPerS	Beschreibt die maximale Geschwindigkeit von Fingern. Dieser ist etwas geringer als der HandSpeedMaxPerS Parameter, da die Fingerregionen deutlich stabiler sind. Dieser Wert wird ebenfalls automatisch auf die aktuelle Framerate heruntergerechnet.	300
BlobDiffRelationPerS	Beschreibt die maximale Abweichung der Relation der Finger zueinander pro Sekunde. Dieser Wert muss aufgrund von Skalierungsgesten, bei denen sich die Relation stark ändert, relativ hochangesetzt werden. Auch hier wird automatisch auf die aktuelle Framerate heruntergerechnet.	100

Tabelle 4.2: Beschreibung der Werte meines definierten Handmodells

4.3.3 Clustering

Beim Clustering wird der komplette Komponentenbaum, das heißt auch die instabilen Regionen, vom MSER Algorithmus traversiert und Fingern, Händen oder Armen zugeordnet.
Dafür wird auf die definierten Werte im Handmodell und die einzustellenden Parameter in der Benutzeroberfläche zurückgegriffen.
Als erstes werden Finger im Komponentenbaum und anschließend unabhängig voneinander Handflächen mit zugehörigem Arm bestimmt.
Für mich definiere ich Fingerregionen, die eine, im Handmodell festgelegte, maximal Fläche besitzen und durch ihren starken Helligkeitsabfall viele ähnliche Elternknoten besitzen.

Folgende Werte können für das Erkennen der Finger eingestellt werden:

- min parents
- max variation
- min level

Als Finger kommen nur Blattknoten in Frage, welche eine bestimmte Anzahl (min parents) an ähnlichen (max variation) Elternknoten besitzen. Der Parameter "max variation" beschreibt das Verhältnis der Größe zum Elternknoten. Dieser Wert wird in Pixel angegeben und ist dementsprechend für jedes System anzupassen. Beispielsweise bedeutet ein Wert von 10, dass die Elternregion maximal 10 Pixel größer sein darf, damit die Region als Finger erkannt werden kann.
Der letzte Parameter (min level) definiert den Grauwert, den diese Region überschreiten muss, um als Fingerspitze erkannt zu werden.

Um Hände zu erkennen, gibt es nur einen einzustellenden Parameter:

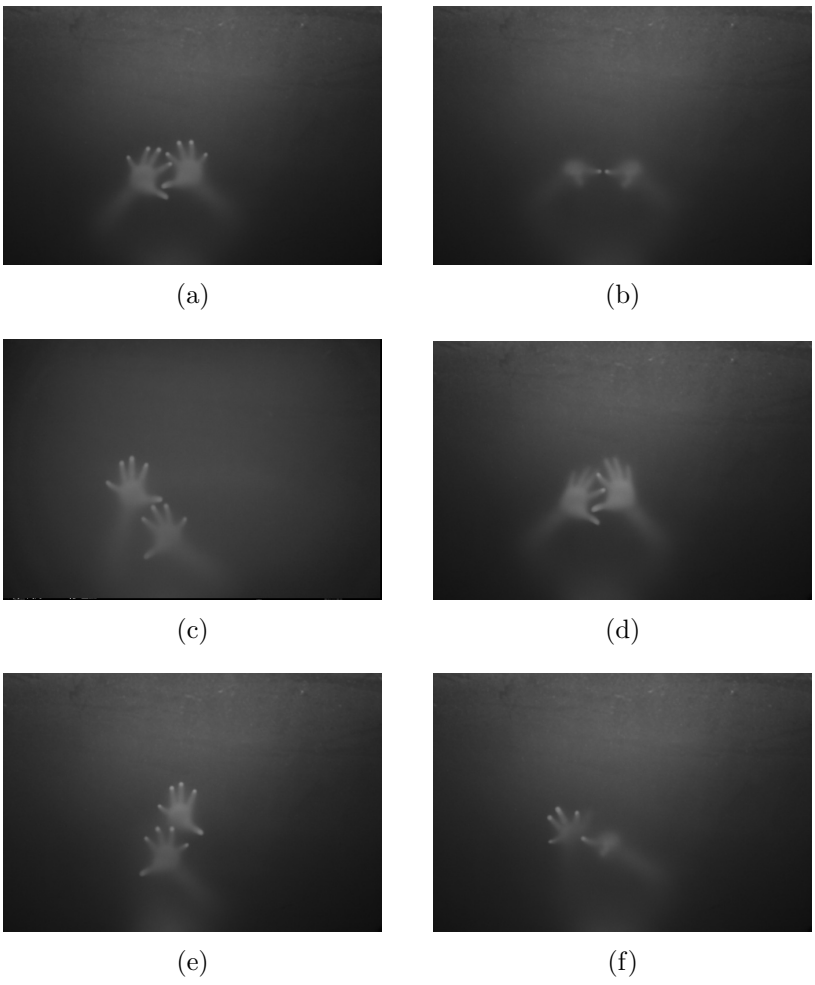
- max variation

Dieser beschreibt, genau wie beim Finger, die relative Größe zum Elternknoten in Pixeln. Da die Handfläche durch die ausgewogene Helligkeit und den starken Helligkeitsabfall am Rand besonders viele Regionen hat, kann man mit diesem Parameter und der definierten Größe im Handmodell relativ zuverlässig Hände erkennen. Fälschliche Regionen, die als Handflächen erkannt werden, wie beispielsweise Ellbogen, stören wenig, da diesen keine Finger zugeordnet werden.

Anschließend wird über jeder Hand eine Armregion gesucht und die zugehörigen Finger über die Baumstruktur zugeordnet und daraus ein “Cluster” erstellt. Dabei kann es passieren, dass nicht alle Finger zugeordnet werden können, da sich diese auch mal direkt unter einer größeren Vaterregion der erkannten Armregion befinden. Ist dies der Fall, wird der nicht zugeordnete Blob zu dem Cluster mit der nächsten Handfläche zugeordnet.

Am Ende wird die Vertrauenswürdigkeit oder besser gesagt die Richtigkeit des Clusters geprüft. Dafür kommen die definierten Werte im Handmodell zum Einsatz. Hier wird geprüft, ob das gefundene Cluster anatomisch einer Hand entsprechen kann.
Diese Überprüfung wird für das anschließende Tracking benötigt.

Folgende Beispiele (siehe Abb. 21) können noch korrekt geclustert werden.



(a) (b) (c) (d) (e) (f)

Abbildung 21: Bildbeispiele, welche noch geclustert werden können

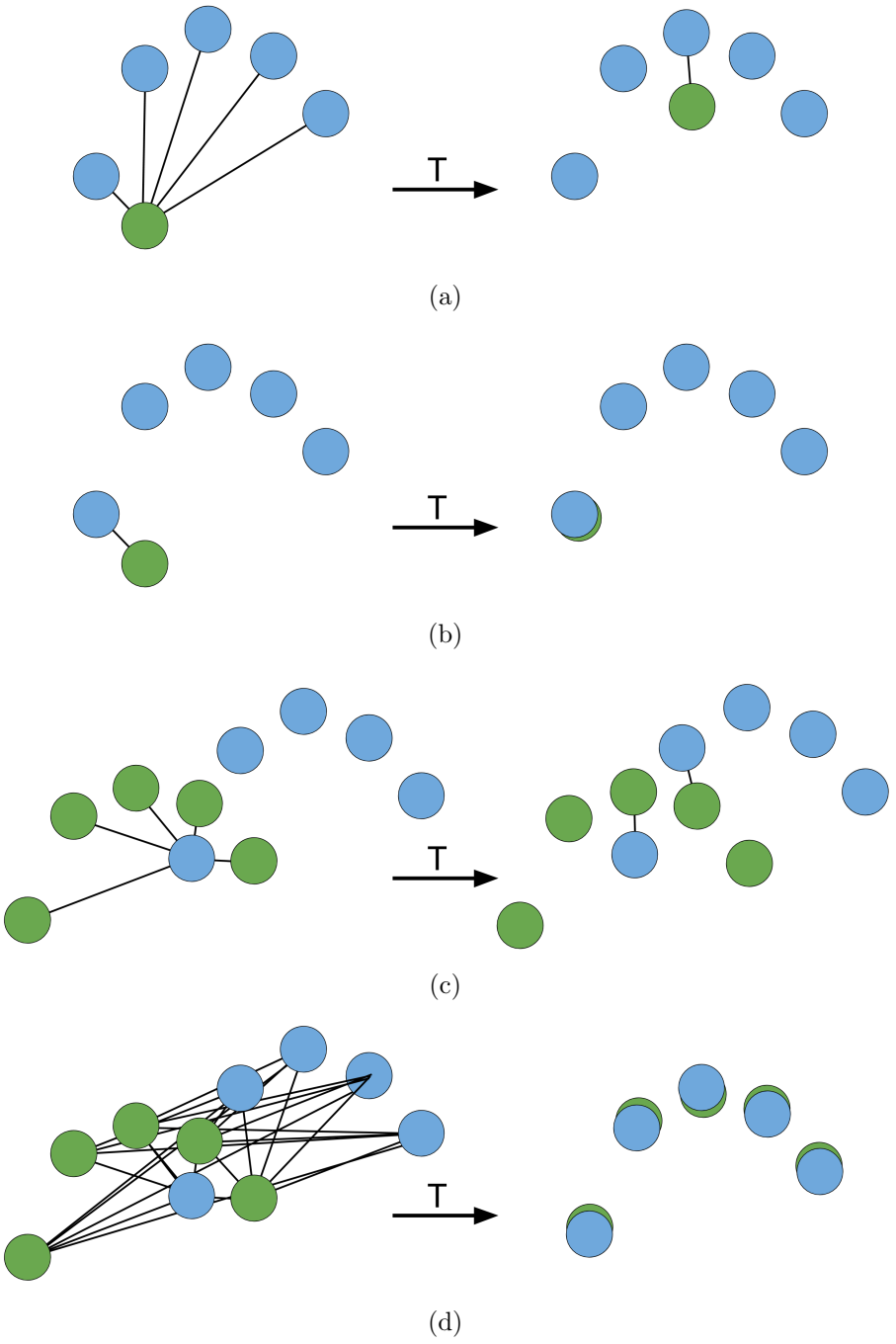
Werden die Hände noch weiter zusammengeführt, verschmilzt das Cluster und wird als falsch weitergegeben. Nur beim letzten Beispiel 4.21(f) verschmilzt zwar das Cluster, wird dennoch als korrektes Cluster weitergegeben, da die Fingerpositionen meinem definierten Handmodell entsprechen. Letzteres stellt folglich eine Fehlinterpretation dar.

4.3.4 Hand-Tracking

Beim Hand-Tracking wird jedem ankommenden Cluster eine aktuelle Hand zugeordnet und die jeweiligen Finger, als Punktwolke betrachtet, über ICP getrackt.

Standardgemäß geht beim ICP Algorithmus jeder Finger der Hand auch nur

die nächste Verbindung zum Kandidaten ein. Anderes Verhalten erreicht man, wenn mehrere Verbindungen zugelassen werden, sodass jeder Finger die nächsten zwei oder gar alle Verbindungen zu den Kandidaten eingehen kann. Dies führt zu unterschiedlichen Transformationen und somit unterschiedlichen Tracking-Ergebnissen. Folgende Beispiele verdeutlichen die Unterschiede bei einer oder allen möglichen Verbindungen. Die grünen Punkte entsprechen wieder den aktuellen Fingern und die blauen den neu hinzukommenden Kandidaten. Auf der linken Seite sind die vom ICP eingegangenen Verbindungen und auf der rechten Seite die nach erfolgreicher Transformation gefundene Zuordnung über KNN dargestellt.



(a)

(b)

(c)

(d)

Abbildung 22: Schematischer Vergleich des ICP Verhaltens mit einer und allen zugelassenen Verbindungen von Fingern zu Kandidaten.

Jede Variante hat ihre Vorteile. Tracke ich nur einen Finger (4.22(a) und 4.22(b)), ist es sinnvoll nur die nächste Verbindung zum Kandidaten zu ver-

wenden (4.22(b)), da ansonsten die Transformation zu sehr von den anderen Kandidaten beeinflusst wird und falsch verschoben und getrackt wird (siehe Bild 4.22(a)).
Bleiben aber alle Finger erhalten, ist es ratsam alle Verbindungen zuzulassen, damit die Transformation auch bei sehr schnellen Bewegungen korrekte Ergebnisse garantiert. Die Teilbilder 4.22(c) und 4.22(d) verdeutlichen diesen Sachverhalt. Bei nur einer Verbindung und fünf Fingern transformiert sich die Punktwolke der Finger (Grün), sodass sie nur um den einen verbundenen blauen Kandidaten liegt und die Zuordnung ist unbestimmt. In Bild 4.22(d) liegen die Punktwolken dagegen in der Gesamtheit optimal übereinander.

Um das bestmögliche Ergebnis zu erzielen, wende ich den ICP Algorithmus mit jeweils einer, zwei und allen Verbindungen an. Am Ende wähle ich die naheliegendste Zuordnung über die Anzahl der Updates und der geringsten Gesamtdistanz aus.

Um die Berührungspunkte zu updaten, betrachte ich sowohl die zurückgelegte Distanz, als auch die Relation zu den anderen Fingern der Hand. Da der ICP Algorithmus die Relation der Finger erhält, lässt es sich über die im Handmodell festgelegte maximale Abweichung der Finger von Frame zu Frame gut abschätzen, wie Finger zugeordnet werden müssen. Es ist unwahrscheinlich, dass bei ähnlicher Relation und einer gleichen Anzahl von aktuellen Fingern und neuen Kandidaten, in einem definierten maximalen Radius, nicht alle Finger geupdatet werden.

Dennoch bleibt nicht auszuschließen, dass der Fall eintreten kann, dass sowohl die Relation, als auch die Distanz in den vorgegeben Bereichen liegen aber dennoch die Wahl der meisten Updates nicht richtig ist. Leider lässt sich dies über meinen Ansatz nicht verhindern. Folgend ist ein Spezialfall vorgestellt, bei dem meine Strategie schief geht. Dies kann beispielsweise durch Erkennungs-Fehler hervorgerufen worden sein.

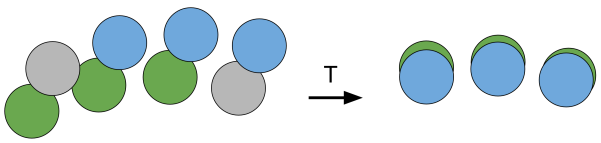


Abbildung 23: Falsches Transformations- und anschließendem Trackingverhalten (rechts) bei gleichzeitigem auf- und absetzen eines Fingers (Grau).

Das Szenario zeigt Zeige-, Mittel, Ring- und kleinen Finger einer rechten Hand in ähnlicher Relation zueinander. Kommt es zu einem gleichzeitigem auf- und absetzen eines Fingers (Grau) oder zu ähnlichem Verhalten durch Erkennungs-Fehler, so führt die anschließende Transformation zu einem falschen Trackingverhalten (vgl. Abbildung 23 rechts). Theoretisch müsste der kleine Finger neu hinzugefügt werden, der Zeigefinger gelöscht und nur Mittel- und Ringfinger geupdatet werden. Da aber alle Finger dieselbe Relation zueinander haben, werden alle Finger (Grün) so transformiert, dass sämtlichen Kandidaten (Blau) geupdatet werden. Dies führt zu einem unerwarteten Sprung der Finger.

Während des Trackings wird unterschieden, ob es sich um ein anatomisch korrektes oder um ein vermutlich falsches Cluster handelt. Diese Information hat Einfluss auf die weitere Abfolge des Trackings.
Wird die Richtigkeit eines ankommenden Clusters bestätigt und dieser erfolgreich über die geringste Distanz einer Hand zugeordnet, werden die jeweiligen Finger in die Bewegungsrichtung der Hand transformiert und über den ICP Algorithmus in dreifacher Ausführung, das heißt mit ein, zwei und allen Verbindungen, getrackt.

Andersherum wird die Richtigkeit widerlegt, wenn zwei oder mehr Hände zu nah zusammen sind und nicht mehr korrekt geclustert werden können. Dann wird über den einfachen ICP, das heiß mit nur einer Verbindung zu jedem Kandidaten, getrackt. Hier wird nur die nächste Verbindung zugelassen, da alle Verbindungen zu allen Kandidaten einen fälschlichen Einfluss auf die Transformation haben.
Dies hat zur Folge, dass das Tracking bei falschen Clustern instabiler wird, da mit zunehmender Geschwindigkeit auch die Initialvermutung der nächsten Kandidaten fehleranfällig wird. Da man für gewöhnlich keine allzu schnellen Bewegungen mit nahen Händen vollzieht, ist diese Limitierung im Allgemeinen aber kaum spürbar. Sobald Hände zu dicht zusammen kommen, wird man erfahrungsgemäß langsamer und geht Berührungen der Hände aus dem Weg. Zudem werden aufliegende Hände in unmittelbarer Nähe in der Regel nicht in die gleiche Richtung bewegt.

Bei falschen Kandidaten, also denen falscher Cluster, liegt bis dato keine Informationen zur Handzugehörigkeit vor und meine Software unterstützt bisher nur Finger mit Handzugehörigkeit. So werden in einem solchen Fall während des Trackings keine neuen Blobs hinzugefügt. Diese Restriktion hat den großen Vorteil, dass fälschlich erkannte Blobs direkt unter der Handfläche vom Tracking ignoriert werden, da das zugehörige Cluster dem Handmodell (min Abstand Handfläche - Finger) nicht entspricht. Die korrekten Finger werden ordnungs-

gemäß getrackt, alle fälschlich erkannten bleiben vom Tracking unberücksichtigt. Dies stabilisiert die Erkennung.

Man hat beim Tracking immer wieder mit falscher Erkennung zu kämpfen und muss versuchen, diese soweit wie möglich zu kompensieren. Um die Auswirkung von Rauschen der Erkennung möglichst im Tracking zu ignorieren, werden Blobs erst hinzugefügt, sobald sie mehr als ein Frame lang aufliegen bzw. erst gelöscht, wenn sie über mehrere Frames hinweg nicht mehr getrackt werden konnten.

Man muss beachten, dass es trotz anatomischer Korrektheit, zu falschen Clustern und dadurch auch zu verfälschten Zuordnungen kommen kann. Um die Auswirkungen dieses Verhaltens zu kontrollieren, werden nach erfolgreichem Tracking die Hände auf ihre Kohärenz zum (vereinfachten) Handmodell geprüft. Kommt es zu Unstimmigkeiten, werden die Finger der jeweiligen Hand zurückgesetzt und wiederum das dafür verantwortliche Cluster als falsch erachtet und erneut getrackt. Dadurch wird realisiert, dass nur die Position der einzelnen Finger geupdatet und nicht die Handzugehörigkeit durch falsche Cluster beeinflusst wird.

Eine wichtige Frage ist: Wie geht man mit den IDs der Finger um, wenn sich ihre Handzugehörigkeit ändert? Theoretisch sollte man davon ausgehen, dass sich die Handzugehörigkeit eines Fingers nicht ändern darf, was sich aufgrund von falschen Clustern nicht immer gewährleisten lässt. Wird nun eine neue Hand erkannt, sollte diese theoretisch auch neue Finger mit neuen IDs bekommen. Für Anwendungen, welche die Handzuordnung verwenden und Interaktionen über Hände steuern, wäre dies auch genau der richtige Ansatz. Eine neue Hand löst auch eine neue Geste aus und dementsprechend werden alle Finger als neu eingestuft.

Andersherum ist es bei Anwendungen, die nur die Position der Finger verwenden. Ändert sich plötzlich die ID eines Fingers, da dieser intern einer neuen Hand zugeordnet worden ist, wird auch die ausgeführte Geste unterbrochen und es kommt zu fehlerhaften Interaktionen.
Um beide Systeme zu unterstützen, werden die IDs der Finger in meiner Version möglichst beibehalten und nur die Handzuordnungen geändert. Dies erfordert die regelmäßige Abfrage der Handzuordnung in der entsprechenden Anwendung für Handgesten.

4.4 Grundbestandteile

Im nachfolgenden Kapitel sind weitere verwendeten Operationen beschrieben, welche zum stabilisieren der Erkennung beitragen oder Informationen zur Verwendung der Software enthalten.

4.4.1 Bild-Vorbereitung

Da die Kamera ein Rohbild liefert, muss dieses in jedem Frame vorerst kalibriert werden. Diese Kalibrierung teile ich zum Verständnis in zwei Schritte auf:

1. Rektifizierung
2. Perspektivische Korrektur

Anschließend muss das Bild drei weitere Operationen durchlaufen:

1. Hintergrundsubtraktion
2. Helligkeitsnormalisierung
3. Verbundene Komponenten

Videodateien oder Bilderfolgen werden bisher nur in bereits kalibrierter Form unterstützt.

4.4.1.1 Rektifizierung

Bei der Rektifizierung wird die Krümmung des Objektives aus dem Bild herausgerechnet. Dafür muss die Krümmung des Objektives und daraus die Kalibrierungsmatrix bestimmt werden. Die Bestimmung der Kalibrierungsmatrix erfolgt durch OpenCV mithilfe eines Schachbrettmusters (siehe Bild) und wird über den Kalibrierungsaufruf in meiner Software gestartet.

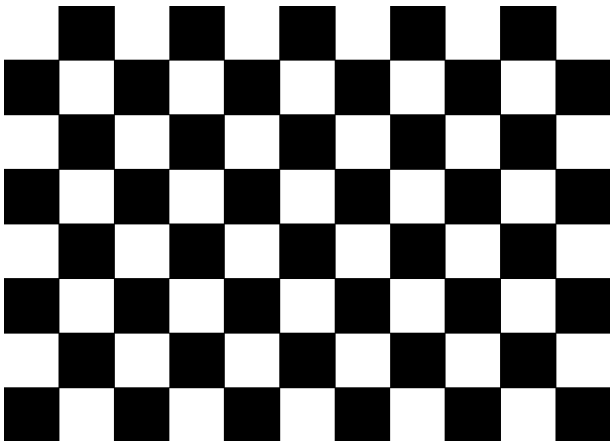


Abbildung 24: Transformationen

Dabei werden mehrere Bilder mit unterschiedlichen Positionen des Schachbrettmusters aufgenommen und durch die OpenCV Funktion cv::findChessboardCorners werden die Positionen der Ecken bestimmt. Wurden genügend Muster erkannt, wird die Kalibrierungsmatrix durch die Methode cv::calibrateCamera bestimmt und gespeichert. Somit können die Bilder ordnungsgemäß begradigt werden (siehe Abbildung A.37(b)).

4.4.1.2 Perspektivische Korrektur

Der zweite Schritt ist die perspektivische Korrektur. Da die Kamera in der Regel nicht so platziert werden kann, dass sie senkrecht auf die Tischplatte schaut, stehen die Kanten des Displays nicht parallel bzw. senkrecht zueinander. Um dies zu beheben werden die vier Ecken des Displays im Bild lokalisiert und symmetrisch zueinander transformiert.
Dafür muss im rektifizierten Bild manuell ein weißes Viereck mit den jeweiligen Ecken des Touch-Bereiches aufgespannt und abgespeichert werden. Dies ist mit einem Bildbearbeitungsprogramm schnell erreicht. Anschließend wird dieses bearbeitete Bild über meine Software geladen und die Ecken des weißen Vierecks automatisch erkannt und begradigt.
Da letztendlich nur die Displayregion, das heißt der eigentliche Touch-Bereich, interessant ist, wird dieser im Anschluss herausgeschnitten und das Bild zum nächsten Schritt der Pipeline weitergereicht.

4.4.1.3 Hintergrundsubtraktion und Helligkeitsnormalisierung

Neben der Hintergrundsubtraktion führe ich auch eine Helligkeits-Normalisierung durch. Dafür wird neben dem Hintergrundbild I_{min} auch ein Maximumbild

I_{max} aufgenommen, welches die maximale Helligkeit an jeder Position des Displays für Touch-Eingaben speichert. Um den bestmöglichen Touch-Input zu simulieren, beweget man eine Hand bei laufender Kamera entlang der Gesamtfläche des Displays, sodass jeder Bereich des Displays einmal bedeckt wird [Holzammer et al., 2009]. Während dieses Vorgangs speichert meine Software automatisch die maximale Helligkeit an jeder Position.
Ein schnelleres Verfahren, mit Erfahrungsgemäß besserem Ergebnis, ist es ein infrarotreflektierendes Material auf die gesamte Tischoberfläche zu legen. Sehr gute Erfahrungen wurden dabei von Ewerling et al. mit schwarzen Moltonstoff gemacht [Ewerling et al., 2012]. Dies kann ich bestätigen.
Dividiert man das vom Hintergrund subtrahierte Bild mit dem Maximalbild, so erhält man für die aufliegenden Fingerspitzen bei beiden Bildern einen ähnlichen Wert und garantiert durch die anschließende Multiplikation mit 255, dass das Bild an den Fingerspitzen für die Erkennung ausreichend hell ist. Durch das Quadrieren erfolgt ein schneller Helligkeitsabfall, was die Erkennung vereinfacht.

$$I_{opt} = (\frac{I - I_{min}}{I_{max}})^2 * 255 \tag{4.1}$$

Dieser Vorgang wird parallelisiert, indem die Zeilen des Bildes auf alle Verfügbaren Prozessorkerne aufgeteilt werden.

4.4.1.4 Verbundene Komponenten

Die verbundenen Komponenten (engl. Connected Components) beschreiben ist eine Vorsortierung des Bildes von zusammengehörenden Objekten. Diese werden aus dem hintergrundsubtrahierten Bild über einen Threshold-Parameter bestimmt. Alle Regionen im Bild, die nicht durch diesen Threshold-Helligkeits-Parameter unterbrochen werden, gehören zusammen.
Die verbundenen Komponenten bieten zwei Vorteile. Einerseits werden dadurch die Bereiche im Bild, mit genügend Veränderungen, lokalisiert. Andererseits beschreiben die verbundenen Komponenten bereits zusammengehörende Objekte, wie bestenfalls Hände, und können im Anschluss separat betrachtet werden. Dies erhöht die Performance, da zum einem nicht das gesamte Bild, sondern nur Teile vom MSER prozessiert werden müssen und zum anderen können die Verbundenen Komponenten unabhängig voneinander parallel betrachtet werden.

4.4.2 Klassifikation

Nach dem Tracken der Hände werden diese klassifiziert.
Während der Klassifikation wird die Richtung des Armes bestimmt. Dabei wird die Armregion als Ellipse approximiert und die Richtung der Ellipse mit der geringeren Anzahl an aufliegenden Fingern bestimmt.
Die Unterscheidung zwischen linker und rechter Hand erfolgt über den Winkel zwischen Daumen und Zeigefinger zu Daumen und kleinem Finger.

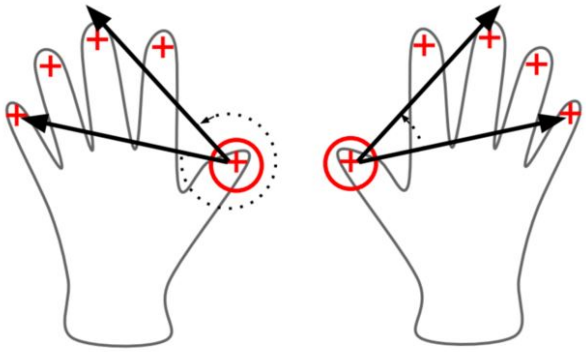


Abbildung 25: Hand Klassifizierung in linke und rechte Hand erfolgt über den Winkel zwischen Daumen und Zeigefinder zu Daumen und kleinem Finger.

Ist dieser Größer als 180° handelt es sich um eine linke Hand, ist er kleiner wird die Hand als rechte Hand interpretiert (siehe Bild). Die Klassifizierung wird für jede Hand nur einmal bestimmt und anschließend für die restliche Dauer beibehalten. Ist die Zuordnung zu links und rechts möglich, werden auch die einzelnen Finger anhand ihrer Reihenfolge als Daumen, Zeigefinger, Mittelfinger, Ringfinger und kleinem Finger eingeteilt. Bisher ist eine Klassifizierung nur bei einer vollständigen Hand möglich. Dabei wird zuerst der Daumen, als Finger mit der größten Distanz zu allen anderen Fingern, bestimmt. Anschließend werden die anderen Finger mit Hilfe des Abstandes zum Daumen festgelegt. Hierbei orientiert sich das System an der Reihenfolge, die vom Daumen wegführt (Zeigefinger, Mittelfinger, Ringfinger und Kleinem Finger). Diese Zuordnung ist für den Ringfinger und kleinem Finger fehleranfällig, funktioniert aber für Zeige- und Mittelfinger gut. Die Klassifizierung orientiert sich an der Lösung von Ewerling at al. [Ewerling et al., 2012].

4.4.3 Senden

Mein Touch-Treiber sendet alle Finger und Hand-Informationen über TUIO [TUIO,] und muss in der entsprechenden Anwendung empfangen und interpretiert werden. TUIO ist ein Netzwerk basiertes Textprotokoll, welches sich auf Open Sound Control (OSC) stützt und von Martin Kaltenbrunner entwickelt worden ist (vgl. [Kaltenbrunner et al., 2005]). Mein Touch-Server sendet OSC Nachrichten mit dem Präfix `/tuio/` zu einer spezifischen Netzwerkadresse über das UDP Protokoll. Um die Kompatibilität zu gewährleisten, wird die TUIO 1.1 Spezifikation verwendet und die Positionen von Fingern als /tuio/2Dcur Nachrichten gesendet. Um zusätzlich Hände und Finger-Informationen zu senden, benutzen wir den in TUIO 2 eingeführten Namensraum `/tuiox/`.

4.4.3.1 Finger

Die Nachricht der Finger besteht aus folgenden fünf Werten.

```
/tuiox/finger s_id x_pos y_pos hand_id class
```

Als erstes die Session ID (`s_id`) von dem Finger, welche der mir zugeordneten Blob ID entspricht. Danach folgen die X und Y Koordinaten der Position. Die Position ist an die Genauigkeit des MSER-Algorithmus und an die Auflösung der Bilder gebunden. Um den Einfluss kleinerer Ungenauigkeiten der Erkennung zu verringern der von Cassiez et al. vorgestellte 1€ Filter verwendet [Casiez et al., 2012]. Der vierte Wert ist die entsprechende Hand ID und der letzte Wert entspricht der zugeordneten Finger- Klasse (Unbekannt, Daumen, Zeige-, Mittel-, Ring-, Kleiner-Finger). Dieser wird als Interger-Wert von 0 (Unbekannt) bis 5 (kleiner Finger) kodiert.

4.4.3.2 Hand

```
/tuiox/hand s_id x_pos y_pos f1_id f2_id f3_id f4_id f5_id
            class arm_major arm_minor arm_incl
```

Die Hand besitzt eine Session ID (`s_id`), sowie die Position (`x_pos`, `y_pos`) und alle fünf Finger IDs (Session IDs der Finger). Besitzt die Hand weniger als fünf Finger, werden die restlichen Finger IDs mit -1 beschrieben. Nach den Finger IDs wird die Klasse (unbekannt, links, rechts) in Form von Integer-Werten (0, 1, 2) übertragen und anschließend die Arm-Ellipse kodiert. Diese wird mithilfe von drei Parametern beschrieben. Zuerst die große Hauptdiagonale (`arm_major`), dann die kleine Hauptdiagonale (`arm_minor`) und am Ende

die Neigung (`arm_incl`) der Arm-Ellipse.

4.4.4 Benutzeroberfläche

Die Benutzeroberfläche ist in QT 4.8 geschrieben und soll die Verwendung des Programms vereinfachen. Die Oberfläche ist an die GUI der Community Core Vision Software angelehnt.

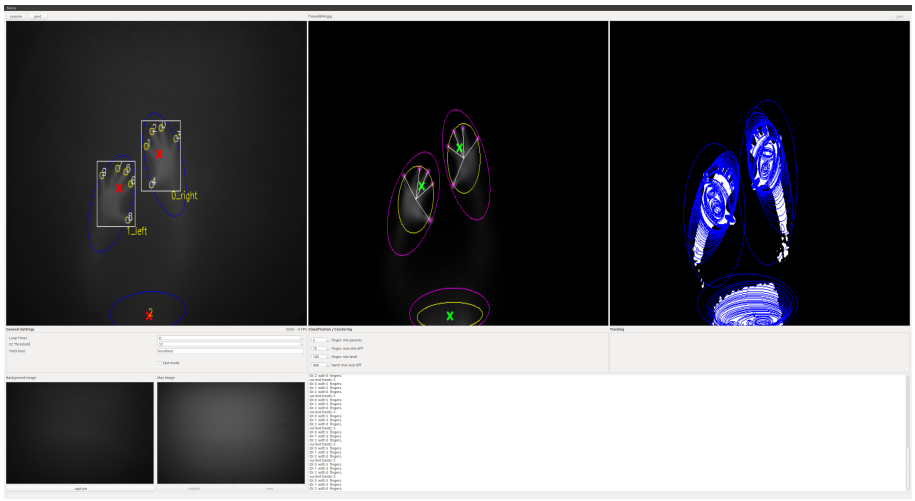


Abbildung 26: Benutzeroberfläche

Über die Menüleiste kann man das Eingabegerät (Kamera, Bildordner, Videodatei) auswählen und man erhält zusätzlich die Möglichkeit, die Kamera zu kalibrieren. Darunter wird links das aktuelle Bild mit getrackten Händen, das hintergrundsubtrahierte Bild mit gefundenen Clustern in der Mitte und die lokalisierten verbundenen Komponenten mit gefundenem Komponentenbaum ganz rechts, gezeigt. Unten links werden die Min- und Max-Bilder angezeigt, mit der Option jeweils ein neues aufzunehmen.
Unter den allgemeinen Einstellungen lässt sich der Connected Component Threshold einstellen. Zusätzlich bietet die Oberfläche hier die Möglichkeit, den TUIO Host, an den die Handdaten geschickt werden, zu bestimmen.
Um eine Änderung der Geschwindigkeit feststellen zu können, wird sowohl die benötigte Zeit in Millisekunden pro Frame, als auch die Anzahl der Frames pro Sekunde (FPS), über den allgemeinen Einstellungen angezeigt.
Die wohl wichtigste Funktion bietet die Aktionsbox “fast mode”. Ist diese Option aktiviert, werden jegliche Visualisierungen übersprungen und dadurch wichtige Prozessierzeit eingespart, wodurch die Software deutlich schneller läuft.

Diese Option sollte stets verwendet werden, um das bestmögliche Tracking-Ergebnis zu erzielen.
Aus Test- und Entwicklungszwecken gibt es die Option, die Kamera zu pausieren oder während einer Bilder- oder Videofolge über den Knopf "next" einen Frame weiter zu gehen. Außerdem lässt sich zu diesem Zweck auch die Prozessierrate im Feld "Loop Timer" einstellen. Hier wird die Zeit in Millisekunden angegeben, die vergehen muss, bevor der nächste Frame abgearbeitet wird. Steht der Wert auf 20 wird beispielsweise alle 20 Millisekunden die Pipeline durchlaufen. Als Standarteinstellung sollte dieser Wert auf 0 stehen.
Um ein Kamerastream aufzuzeichnen bietet die Software oben rechts den Save-Button. Durch das Aktivieren wird man dazu aufgefordert, einen Speicherort für die zu speichernden Bilder auszuwählen. Zum Beenden genügt ein erneutes Klicken auf den Knopf.
Das Textfeld unten rechts zeigt die Anzahl aktuell getrackter Hände und Finger an. Dies ist besonders nützlich für den "fast-mode", da die entsprechende Visualisierung fehlt.
Die für die Erkennung von Fingern und Händen notwendigen Parameter müssen in der Mitte für jedes Setup neu angepasst werden. Anhand der Parameter wird der Komponentenbaum vom MSER durchgegangen und die einzelnen Regionen Finger oder Händen zugeordnet.

Kapitel 5

Vergleichsstudie von Tracking Methoden

In diesem Kapitel werden folgende die Tracking Methoden verglichen: Minimaler-Distanzen-Tracker, KNN, Global-Tracker und der hier entwickelte ICP Ansatz auf ihre Effizienz überprüft.

5.1 Fragestellung und Hypothese

Die nachfolgende Studie untersucht die Stärken und Schwächen des neuen Tracking Ansatzes und vergleicht mit bekannten Tracking Methoden. Dabei sind sowohl die Effektivität, als auch auf die benötigte Rechenzeit berücksichtigt.

Wichtig dabei ist besonders die Robustheit des neuen Ansatzes und das Aufzeigen, dass der ICP Ansatz bessere Ergebnisse als Vergleichsvarianten liefert.

5.2 Aufbau

Um alle Methoden miteinander vergleichen zu können und die Testbedingungen zu kontrollieren, habe ich unterschiedliche Bilderfolgen mit konstanten Geschwindigkeiten aufgenommen und die benötigte Zeit, sowie die zurückgelegte Distanz der jeweiligen Hände gespeichert.
Mit diesen Informationen kann ich jede Geschwindigkeit und Framerate der einzelnen Bilderfolgen simulieren und somit die Effektivität bei unterschiedlichen Frameraten der Tracking Methoden vergleichen. Über die Halbierung der benötigten Zeit simuliere ich eine doppelte Framerate, was gleichbedeutend mit einer Verdoppelung der Handbewegungs-Geschwindigkeit ist.

Um die Tracking-Methoden auf ihre Effektivität zu testen, wird die Framerate Stück für Stück künstlich heruntergesetzt, indem ich Bilder auslasse. Dies vergrößert den Abstand der erkannten Finger und erschwert das Tracking.

Für den Vergleich zähle ich die Fehlerrate für die gesamte Bilderfolge mit jeder Methode. Diese beschreibt die Anzahl an falsch getrackten Fingern, wobei unterschieden wird, ob der Finger eine neue oder eine falsche ID zugeordnet bekommt. Folgefehler werden berücksichtigt.

Da jede Bilderfolge eine andere Problematik anspricht und es schwierig ist, die Bedingungen, wie Geschwindigkeit der Bewegung und die Relation der Finger, mit unseren Mitteln zwischen den einzelnen Bilderfolgen konstant zu halten, steht jede Bilderfolge für sich. Um Vergleiche darüber hinaus zwischen den Bilderfolgen aufstellen zu können, bräuchte man ein kontrollierteres Testsetup, bei dem beispielsweise eine anatomische Nachbildung einer Hand motorisiert über die Tischoberfläche wandert. Dies ist nicht gegeben.

Für die Bilderfolgen habe ich bei 60 Hz meine Hand mit konstanter Geschwindigkeit bewegt und unterschiedliche Gesten simuliert. Dabei wurden folgende Bilderfolgen aufgenommen:

- Translations-Bewegung einer Hand
- Rotations-Bewegung einer Hand
- Zwei Hände zueinander und wieder voneinander weg bewegt
- Zwei Hände eng zusammen aufgelegt und voneinander weg bewegt
- Zwei Hände mit jeweils 3 Fingern zueinander und voneinander weg bewegt
- Einen Finger zum Mittelpunkte einer anderen Hand hin und wieder weg bewegen

Die korrekte Erkennung von Fingern und Handflächen ist in jeder Bilderfolge gegeben.
Um möglichst viele Methoden miteinander zu vergleichen, teile ich die Teststudie in zwei Teile auf. Im ersten Teil wähle ich eine einfache Bilderfolge (Translations-Bewegung einer Hand), um verschiedenste Varianten der Tracking Methoden zu prüfen.

- KNN

- KNN (vorhergesagte Position)
- KNN (Kombiniert)
- Minimale-Distanzen-Tracker
- Minimale-Distanzen-Tracker (vorhergesagte Position)
- Global-Tracker
- Global-Tracker (optimiert)
- ICP mit einer Verbindung
- ICP mit einer Verbindung (falsches Cluster)
- ICP mit allen Verbindungen
- ICP mit allen Verbindungen (falsches Cluster)
- Beobachtung: Community Core Vision Version 1.5

Dabei verwende ich beim KNN einmal die vorhergesagte und einmal die aktuelle Position, sowie die Kombination beider über die Anzahl der Updates und der geringeren Gesamtdistanz. Der Minimale-Distanzen-Tracker wird zudem mit originalen und vorhergesagten Position getestet. Der Global-Tracker wird in normaler und in optimierter Variante verglichen. Beim ICP tracke ich mit einer Verbindung, sowie mit allen Verbindungen und simuliere das Verhalten bei falschen Clustern.

Im zweiten Teil beziehe ich mich nur auf die wichtigsten Tracking-Ansätze ICP, optimierter Global-Tracker und die Kombination der beiden KNN Varianten, sowie der Kombination vom Minimalen-Distanzen-Tracker äquivalent zur kombinierten KNN Variante und vergleiche diese mithilfe der weiteren Bilderfolgen.

Da der Global-Tracker im zweiten Teil der Studie nicht bei allen Testszenarien in absehbarer Zeit fertig wird, nutze ich die optimierte Variante (siehe: 2.3.3), bei der jeder Finger nur die Verbindung zu den nächsten zwei Kandidaten eingeht. Da sich die Anzahl der Finger bei allen Bilderfolgen nicht ändert, geht der ICP Algorithmus bei erfolgreicher Clusterbildung alle Verbindungen und bei falscher Clusterbildung nur die jeweils nächste Verbindung, vom Finger aus, ein.

Nachfolgend werden die Ergebnisse der ersten und zweiten Studie mit simulierten Frameraten beschrieben. Die Ergebnisse werden in jedem Teilabschnitt ausgewertet. Wobei der erste Frame für die Auswertung ausgelassen wird, da noch keine Zuordnungen stattfinden.

5.3 Teilstudie 1

Nachfolgend werden die Ergebnisse der ersten Studie mit simulierten Frameraten beschrieben. Dabei gehe ich zuerst auf die Beobachtungen der einzelnen Tracking Ansätze ein und komme anschließend zur Auswertung.

5.3.1 Beobachtung (Translations-Bewegung)

Diese Bilderfolge beschreibt eine Bewegung, bei der sich eine komplette Hand erst 20cm in die eine und anschließend bei konstanter Geschwindigkeit in die andere Richtung bewegt hat. Dies ist beispielsweise eine typische Translations-Geste.

(a) (b) (c)

Abbildung 27: Translations-Bewegung: Hand wandert von a nach b zurück zu c.

Insgesamt hat jede Hand und somit jeder Finger 40 cm innerhalb von 252 ms zurückgelegt. Dies entspricht umgerechnet einer Geschwindigkeit von ca. 1,59 Meter pro Sekunde, aufgenommen bei 120Hz.

Folgend sind die Frameraten, abhängig von der Anzahl an Bildern, und die daraus resultierende, zurückgelegte Distanz von Frame zu Frame dargestellt.

FPS	Anzahl an Bildern	Distanz pro Frame in cm
120	31	1.29
60	15	2.67
30	7	5.71
10	3	13.3

Nachfolgend sind die Ergebnisse der einzelnen Methoden tabellarisch aufgeführt.

5.3.1.1 KNN

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.029
60	14	14	28	2	0.026
30	12	12	24	4	0.027
10	7	2	9	4.5	0.027

K-nächste Nachbarn mit der unveränderten Position findet die richtige Zugehörigkeit der Finger bei einer Framerate von 120 Hz ohne Probleme. Pro Frame wird durchschnittlich eine Distanz von 1.29 cm zurückgelegt. Da meine Finger in dem Beispiel mindestens 3cm Abstand voneinander haben, befindet sich der nächste Kandidat auch immer in unmittelbarer Nähe von dem korrekten Finger.
Mit 60 FPS gibt es eine durchschnittliche Fehlerrate von zwei pro Frame. Das heißt zwei Finger, von den fünf aktiven Fingern, lassen sich nicht korrekt zuordnen. Da sich Zeige- und Mittelfinger auf einer Höhe in Bewegungsrichtung befinden, tendiert KNN dazu, diese falsch zuzuordnen. Einer der beiden Finger wird mit der ID des anderen aktualisiert und der andere bekommt eine neue zugewiesen.
Setzt man die Framerate noch weiter herunter auf 30 Hz werden vier Finger durchschnittlich pro Frame falsch zugeordnet. In diesem Fall findet eine Verschiebung der IDs von Zeige-, Mittel-, Ring- und kleinem Finger in die jeweilige Richtung statt und die beiden Finger am Ende der Kette bekommen eine neue ID zugewiesen. Bei 10 FPS wird zudem der entfernte Daumen einmal falsch zugewiesen. Die Zeit bleibt dabei bei allen Frameraten annähernd konstant.

5.3.1.2 KNN (vorhergesagte Position)

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	5	4	9	0	0.029
60	14	15	29	2.1	0.029
30	12	12	24	4	0.028
10	7	2	9	4.5	0.028

Bei KNN mit vorhergesagter Position schlägt die Zuordnung schon bei 120Hz fehl. Insgesamt konnte ich neun Fehler bei 31 Bildern zählen. Die vorhergesagten Positionen wandern beim Richtungswechsel in die falsche Richtung und vergrößern den Abstand zum eigentlichen Finger. Im Test ließ sich beobachten, dass dies eine drastische Auswirkung auf das Tracking zur Folge hat. Sobald sich die Hand in die entgegengesetzte Richtung bewegt, werden die IDs vertauscht und der Fehler der vermuteten Position setzt sich weiter fort, sodass sich erst nach neun falschen Zuordnungen das Tracking wieder stabilisiert.
Für 60, 30 und 10 FPS lässt sich ähnliches Verhalten wie bei KNN mit normaler Position feststellen.

5.3.1.3 KNN (Kombiniert)

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.066
60	14	14	28	2	0.064
30	12	12	24	4	0.061
10	7	2	9	4.5	0.056

Die Werte des kombinierten k-nächste-Nachbarn-Algorithmus gleichen den Ergebnissen des KNN-Trackers mit den tatsächlichen Positionen. Dies erfolgt bei doppelter Rechenzeit (Durchschnittlich 0,0639 ms).

5.3.1.4 Minimale-Distanzen-Tracker

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.005
60	3	6	9	0.64	0.05
30	3	9	12	2	0.005
10	0	4	4	2	0.005

Beim Einsatz des Minimalen-Distanzen-Trackers kann eine falsche Zuordnung bereits bei 60 FPS wahrgenommen werden. Es werden bei dieser Framerate durchschnittlich 0,64 Finger pro Bild falsch zugeordnet. Im Vergleich dazu lassen sich bei doppelter (30 FPS), sowie sechsfacher Bewegungsgeschwindigkeit (10 FPS) zwei Fehlern pro Frame ausmachen.
Der größere Updateradius bei 10 FPS sorgt dafür, dass keine Finger mehr verloren gehen, sondern alle Finger bei einer durchschnittlichen Rechenzeit von 0,005 ms ohne Rücksicht auf starke Positionssprünge getrackt werden.

5.3.1.5 Minimale-Distanzen-Tracker (vorhergesagte Position)

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	1	2	3	0.1	0.005
60	3	6	9	0.64	0.006
30	3	9	12	2	0.005
10	0	4	4	2	0.005

Der Minimale-Distanzen-Tracker mit vorhergesagten Position verliert beim Richtungswechsel die Zuordnung von drei Fingern. Alle weiteren Ergebnisse sind äquivalent zum Minimale-Distanzen-Tracker mit originalen Positionen.

5.3.1.6 Global-Tracker

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.006
60	0	0	0	0	0.013
30	0	12	12	2	0.090
10	0	6	6	3	0.113

Das Tracking des Global-Trackers liefert bis 60Hz stets die korrekte Zuordnung. Bei 30 Hz werden jeweils Mittel- und Ringfinger vertauscht. Eine Wiederholung des Vorganges bei 10Hz liefert eine durchschnittliche Fehlerquote von drei falsch zugewiesenen Fingern. Dies lässt sich darauf zurückführen, dass die sich kreuzenden Verbindungen eine geringere Gesamtdistanz als die annähernd parallelen Verbindungen der korrekten Zuordnung haben (siehe Abbildung 28).

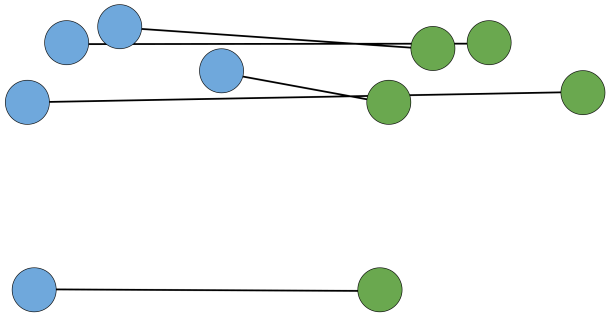


Abbildung 28: Falsche Zuordnung von Punkten über durch den Global-Tracker.

Zudem lässt sich eine große Steigerung der notwendigen Prozessierzeit beobachten. Von 120 auf 10 Hz benötigt der Global-Tracker 19-mal so lang, da mit dem Herabsetzten der Framerate automatisch der maximale Updateradius ansteigt und immer mehr Verbindungen zu anderen Kandidaten berücksichtigt werden müssen.

5.3.1.7 Global-Tracker (optimiert)

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.009
60	0	0	0	0	0.033
30	0	9	9	1.5	0.052
10	0	4	4	2	0.106

Im Vergleich zur vorherigen Variante kommt der optimierte Tracker auf verbesserte Ergebnisse. Diese minimalen Verbesserungen sind jedoch darauf zurückzuführen, dass bei der optimierten Variante die Reihenfolge eine Rolle spielt, in welcher die Finger abgearbeitet werden. Da immer nur die nächsten zwei Finger betrachtet werden, kommt es zu dem in Abbildung 28 gezeigtem Updateverhalten nicht.

Der optimierte Global-Tracker benötigt von 120 auf 10 Hz ca. 12-mal so lang für die Zuordnung und weist so eine geringere Prozessierzeit auf, als der normale Global-Tracker.

5.3.1.8 ICP mit einer Verbindung

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.058
60	0	0	0	0	0.061
30	0	0	0	0	0.062
10	0	0	0	0	0.066

Der Hand-Tracking Ansatz mit ICP Algorithmus mit einer Verbindung liefert bei jeder FPS die richtige Zuordnung. Durchschnittlich benötigt der reine ICP Algorithmus 0,07 ms.

5.3.1.9 ICP mit einer Verbindung (falsches Cluster)

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.067
60	0	0	0	0	0.065
30	0	0	0	0	0.068
10	0	0	0	0	0.072

Die Zuverlässigkeit des ICP Algorithmus lässt auch bei falscher Clusterbildung nicht nach. Nach maximal fünf Iterationsschritten pendelt sich die Zuordnung wieder ein und es werden die korrekten Ergebnisse erzielt. Eine negative Auswirkung auf die benötigte Rechenzeit bleibt ebenfalls aus.

5.3.1.10 ICP mit allen Verbindungen

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.058
60	0	0	0	0	0.061
30	0	0	0	0	0.063
10	0	0	0	0	0.067

Beim Einsatz des ICP Algorithmus mit allen Verbindung lassen sich keine Fehler finden und es kommt stets die korrekte Zuordnung als Ergebnis heraus.

5.3.1.11 ICP mit allen Verbindungen (falsches Cluster)

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate pro Frame	Zeit in ms
120	0	0	0	0	0.060
60	0	0	0	0	0.063
30	0	0	0	0	0.065
10	0	0	0	0	0.066

Der ICP Algorithmus mit allen Verbindung und falscher Clusterbildung kommt immer bereits nach einem Iterationsschritt auf die korrekte Zuordnung.

5.3.1.12 Community Core Vision 1.5

Nach Qian Liu et al. [Liu, 2010] schneidet die CCV Version im Vergleich zu Touchlib 2.4.3 und 2.4.2 beim Finger-Tracking am besten ab.
Um dies zu testen, habe ich die Translations-Bewegung in die CCV Version 1.5 als Videoformat eingeladen und die 120, 60, 30 und 10 Hz simuliert. Dabei habe ich die von der CCV Version gesendeten Daten ausgewertet. Bei 120 Hz findet die CCV Version immer die korrekte Zuordnung, aber bereits bei 60 Hz wurden die ersten Finger vertauscht. Insgesamt waren die Ergebnisse der CCV Version vergleichbar mit KNN 5.3.1.1.

5.3.2 Auswertung

Die sehr schnelle Translations-Bewegung wird, bis auf die Varianten mit vorhergesagten Position, von allen Tracking-Methoden bei 120 FPS richtig zugeordnet. Nur der ICP Algorithmus kommt in allen Varianten sogar bei 10

Hz auf die richtig Zuordnung. Umgerechnet entspricht diese Bewegung bei 10 FPS eine Geschwindigkeit von 19,08 Meter pro Sekunde bei 120 FPS. Diese theoretisch mögliche Geschwindigkeit überschreitet aber bei weitem die für gezielte Interaktionen notwendige maximale Geschwindigkeit. Meine definierte Maximalgeschwindigkeit liegt bei drei Metern pro Sekunde, was dem Szenario mit 60 FPS mit einer theoretischen Geschwindigkeit von 3,18 Metern pro Sekunde am nächsten kommt. Bei dieser Bewegungsgeschwindigkeit scheitern aber bereits die KNN Varianten mit zwei Fehlern pro Frame, sowie der Minimale-Distanzen-Tracker mit 0,64 Fehlern pro Frame, sodass diese für effektives Arbeiten mit der besagten Geschwindigkeit keine Option darstellen.

Der Global-Tracker kommt bei fünf aufliegenden Fingern stets auf eine vertretbare Rechenzeit. Dabei ist bereits eine Verminderung des benötigten Rechenanstiegs bei der optimierten Variante mit den nächsten zwei Kandidaten zu beobachten.
Zudem werden bis zu 60 FPS die richtigen Zuordnungen erkannt und somit bei dieser Bewegung und der vorliegenden Konstellation der Finger das fehlerfreie Arbeiten mit der von mir definierten Maximalgeschwindigkeit ermöglicht.

Die Effektivität vom ICP Algorithmus ist in diesem Szenario von keiner vorgestellten Methode reproduzierbar. Dabei benötigt der Algorithmus im Durchschnitt 0,07 ms Rechenzeit. Die benötigte Zeit von 0,06 ms für das vom Hand-Tracking notwendige Clustering ergibt zusammen eine benötigte Gesamtzeit für den Hand-Tracking Ansatz von 0,13 ms.

5.4 Teilstudie 2

Um die Arbeit übersichtlich zu halten, sind die Ergebnisse und die Frameraten in Abhängigkeit von der Anzahl an Bildern und der daraus resultierenden zurückgelegten Distanz pro Frame der jeweiligen Bilderfolge tabellarisch im Anhang aufgeführt (Seite 87 bis 92). Nachfolgend sind die Beobachtungen beschrieben. Darauf folgt die Auswertung.

5.4.1 Beobachtung

5.4.1.1 Rotations-Bewegung

Diese Bilderfolge beschreibt die Rotation einer kompletten Hand um 90° in eine Richtung, um sie anschließend im selben Winkel zurück in die Ausgangsposition zu bewegen. Die Ergebnisse befinden sich in Tabelle C.1.

(a) (b) (c)

Abbildung 29: Rotations-Bewegung: Hand rotiert von a nach b und wieder zurück zu c.

Die Bewegung von insgesamt 180° wurde in 738 ms ausgeführt. Bei 89 Bildern und 120 FPS entspricht das einer Rotation von 2.02° pro Frame. Dieser Wert liegt unter der von mir definierten Gradzahl (3,33°) für die Maximal-Rotation pro Frame bei 120 FPS.

Die Kombination der KNN Varianten findet bis einschließlich 30 Hz die korrekte Zuordnung. Der Minimale-Distanzen-Tracker benötigt mit Abstand die geringste Rechenzeit (0.005 ms) schließt aber beim Tracking der vorgegebenen Rotationsbewegung am schlechtesten ab.
Sowohl der Globale-, als auch der Hand-Tracking Ansatz findet stets die korrekte Zuordnung. Diese Bildfolge bestätigt, dass ICP auch bei Rotationsgesten stabile Ergebnisse liefert.

5.4.1.2 Zwei Hände zusammen und auseinander

In einer weiteren Bildfolge wird eine Bewegung festgehalten, bei der sich zwei Hände erst aufeinander zu und im Anschluss erneut voneinander weg bewegen. Die Ergebnisse befinden sich in Tabelle C.2.

(a) (b) (c)

Abbildung 30: Skalierungs-Bewegung: Zwei Hände bewegen sich aufeinander zu (b) und wieder zurück.

Beide Hände stoßen nach jeweils 10 cm zusammen und legen somit für Hin- und Rückbewegung jeweils 20 cm zurück. Die gesamte Geste dauerte 617 ms und

entspricht damit einer Geschwindigkeit von 32 cm pro Sekunde bei 120 Frames.

Bei dieser Bilderfolge sind Zeige-, Mittel- und Ringfinger beider Hände, sowie die nah aneinander vorbei bewegenden Daumen, problematisch. Die unterschiedlichen Tracking-Varianten erzielen hierbei sehr verschiedene Ergebnisse. KNN verliert sowohl bei 30 FPS, als auch bei 10 FPS rund sechs Finger pro Frame. Dabei werden die Daumen der Hände und die mittleren Finger vertauscht.
Der Minimale-Distanzen-Tracker schneidet im direkten Vergleich mit KNN bei 30 Hz mit 2.6 falsch zugeordneten Fingern pro Frame besser ab. Bei 10 Hz ist die Fehlerrate dagegen deutlich höher als die mit KNN. Zudem wird ein "gieriges" Verhalten deutlich: Bei 10 Hz werden fast alle Finger (9,5 pro Frame) falsch zugeordnet, aber kein einziger Finger neu erstellt. Hingegen kommen bei KNN fünf neue Finger pro Frame hinzu. Beide Tracking-Methoden haben folglich Probleme mit der Bewegung umzugehen und die richtige Zuordnung zu finden.
Sowohl Global-Tracker, als auch ICP stoßen bei 30 Hz an ihre Grenzen. Ersterer liefert mit nur 0,5 Fehlern pro Frame bessere Ergebnisse bei einer Abtastrate von 30 FPS und der Hand-Tracking Ansatz bei 10 FPS. Alle Fehlzuordnungen von ICP kommen erst während der falschen Clusterbildung zustande, wobei der Daumen der anderen Hand die Transformation fälschlich beeinflusst.

Zudem wird hier der Anstieg der Rechenzeit in Abhängigkeit der aufliegenden Finger des Global-Trackers deutlich. Bei zehn Fingern wird die optimierte Variante mit zwei Verbindungen pro Finger von 120 auf 10 Hz ca. 48-mal langsamer. Wird die Anzahl der maximalen Verbindungen bei 10 FPS weiter auf drei erhöht, steigt auch die benötigte Zeit von 0,48 ms auf 11,42 ms. Der Globale-Tracker mit allen Verbindungen braucht dagegen 2116,14 ms. Dabei bleibt das Ergebnis bei allen Varianten das gleiche.

5.4.1.3 Zwei Hände auseinander und zusammen

Diese Bilderfolge bildet das Gegenstück der vorherigen. Hier wird dieselbe Bewegung noch einmal aber in anderer Reihenfolge abgearbeitet. Zuerst befinden sich die Hände nah beieinander und bewegen sich nach außen, um schließlich wieder zurück in ihre Ausgangsposition zu wandern. Die Ergebnisse befinden sich in Tabelle C.3.

(a) (b) (c)

Abbildung 31: Skalierungs-Bewegung: Zwei Hände liegen dicht beisammen auf (a), wandern jeweils nach außen (b) und wieder zurück (c).

Da alle Daten äquivalent zu der vorherigen Bilderfolge sind und sich nur die Reihenfolge der Bilder geändert hat, kommen wir auch auf ähnliche Ergebnisse bei den reinen Positions-Tracking-Algorithmen KNN, Minimale-Distanzen-Tracker und Global-Tracker.
Das Ziel dieser Bilderfolge ist es, die Limitierung meines Hand-Tracking Ansatzes zu zeigen. Meine Implementierung des ICP Algorithmus benötigt die Handzugehörigkeit eines Fingers, um diesen tracken zu können. Diese ist bei falschen Clustern, wie sie in den ersten Bildern auftauchen, nicht gegeben. Die Hände sind in der Ausgangsposition sehr nah aneinander, sodass eine korrekte Clusterbildung erst später erfolgt.
Bei 120 FPS kommt es erst im siebten Frame zu einer korrekten Clusterisierung der Hände und es kann dementsprechend erst ab dem achten Bild korrekt getrackt werden. Bei 60 Hz werden die Finger im vierten Frame, bei 30 FPS im dritten und bei 10 FPS im zweiten, korrekt geclustert.
Rechnet man die Fehler auf die Anzahl der trackbaren Bilder hoch, wird sowohl bei 30, als auch bei 10 Hz durchschnittlich ein Finger falsch zugeordnet. Verglichen zur vorherigen Bilderfolge ist das auf den ersten Blick ein besseres Ergebnis. Es muss jedoch beachtet werden, dass der anfängliche schwer zu trackende Teil auf Grund des falschen Initial-Clusters ignoriert wurde.

5.4.1.4 Sechs Finger zusammen und auseinander

Folgende Bilderfolge ähnelt der Bilderfolge, bei der zwei vollständige Hände sich erst zueinander und anschließend voneinander weg bewegt haben. Eine ähnliche Bewegung wurde hier jedoch lediglich mit jeweils drei Fingern vollzogen. Die Ergebnisse befinden sich in Tabelle C.4.

(a) (b) (c)

Abbildung 32: Skalierungs-Bewegung: Zwei Hände mit jeweils 3 Fingern bewegen sich aufeinander zu (b) und wieder zurück.

Die Hände haben sich insgesamt 30 cm innerhalb von 265 ms, also mit einer Geschwindigkeit von 1,13 Metern pro Sekunde, bewegt. Umgerechnet entspricht das bei 120 Frames pro Sekunde und 63 Bildern eine Distanz von 0,48 cm pro Frame.

Der Hand-Tracking Ansatz über ICP schneidet bei dieser Testfolge etwas schlechter ab als der Global-Tracker. Beide garantieren bis einschließlich 30 FPS ein ordnungsgemäßes Tracking, sobald die Daumen jedoch sehr nah beieinander große Sprünge machen, wirkt sich das bei beiden Tracking-Methoden negativ aus.
Das Globale-Tracking vertauscht die IDs der Daumen bei 10 Hz jeweils für die Hin- und Rückbewegung. Hingegen ordnet der ICP Algorithmus, während der falschen Clusterbildung, alle Finger nicht korrekt zu.

5.4.1.5 Einen Finger zum Mittelpunkt einer anderen Hand hin- und wieder wegbewegen

Bei dieser Bilderfolge bewegt sich ein Finger über eine gedachte Linie von Daumen und Zeigefinger einer anderen ruhenden Hand zu dessen Mittelpunkt und wieder zurück. Dabei hat sich der eine Finger insgesamt 20 cm innerhalb von 299 ms bewegt. Die Ergebnisse befinden sich in Tabelle C.5.

(a) (b) (c)

Abbildung 33: Geste: Ein Finger kreuzt die gedachte Linie von Daumen und Zeigefinger einer anderen ruhenden Hand und wandert zu dessen Mittelpunkt (b) und wieder zurück zur Ausgangsposition (a) bzw. (c).

Die Bilderfolge wird von allen Methoden, bis auf KNN, bei jeder Abtastrate korrekt getrackt. Bei 30 und 10 Hz wird der sich bewegende Finger bei KNN fälschlicherweise auch dem Zeigefinger der ruhenden Hand zugeordnet und deswegen neu erstellt, anstatt aktualisiert zu werden.

5.4.2 Auswertung

Die Behauptung, je größer die Bewegungsgeschwindigkeit desto fehleranfälliger ist das Tracking, wurde bestätigt. Zudem kamen alle Methoden bei verringerter Abtastrate auf schlechtere Ergebnisse. Dabei wiesen die globalen Betrachtungen, wie der Global-Tracker und mein Hand-Tracking Ansatz stets bessere Ergebnisse auf als die übrigen Varianten. Im Vergleich zu KNN und dem Minimalen-Distanzen-Tracker waren diese zudem nicht so stark abhängig von größeren Abständen zwischen den Fingern und jeweiligen Kandidaten und konnten fehlerfrei bis zu dreifache Geschwindigkeiten tracken.

Sowohl KNN, als auch der Minimale-Distanzen-Tracker zeigen ähnliche Ergebnisse aber mit unterschiedlichem Verhalten. KNN tendiert dazu unwahrscheinliche Zuordnungen wegzulassen und anstatt zu updaten werden neue Finger erstellt. Im Gegenzug aktualisiert der Minimale-Distanzen-Tracker jeden Finger, sofern ein noch nicht verwendeter Kandidat in dessen Updateradius liegt. Durch ein solches hartnäckiges Verhalten kommt es in der Regel zwar zu weniger Fehlern, gleichzeitig werden große fehlerhafte Sprünge der Finger riskiert. Dies wird besonders bei kleinen Frameraten und dementsprechend hohen maximalen Updateradien zum Problem.
Meine definierte Maximalgeschwindigkeit von Fingern beträgt drei Meter pro Sekunde, dies entspricht bei einer Framerate von 10 FPS einem Updateradius von 30 cm. Da werden die IDs von Daumen und kleinem Finger vom Minimalen-Distanzen-Tracker zeitweise vertauscht. Dies bedeutet für die dar-

auf aufbauende Anwendung einen starken Geschwindigkeitsschub eines Fingers und wird je nach Interpretation der Anwendung zu unerwarteten Interaktionen führen.
Da das Ziel einer Touch-Software es sein sollte, möglichst genaue Daten zu liefern, müssen Sprünge in dieser Größenordnung abgefangen werden. Entweder man beruft sich dabei auf den zurückhaltenderen KNN Algorithmus und nimmt eine größere Anzahl an Fehlern in Kauf, erhält als Gegenleistung jedoch minimierte Ausreißer oder man muss das Verhalten vom Minimalen-Distanzen-Tracker weiter anpassen und beispielsweise die letzte Bewegungsgeschwindigkeit der Finger gewichten, um Sprünge abzufangen.

Beim Tracken über KNN und dem Minimalen-Distanzen-Tracker traten erste gemessene Probleme bei einer zurückgelegten Distanz von rund 2 cm pro Frame auf. Dies lässt sich leicht anhand der Fingerkonstellation nachvollziehen. Meine Finger lagen mindestens 3 cm voneinander entfernt auf. Dies entspricht bei Verwendung der originalen Positionen, einen Spielraum von 1,5 cm Radius bevor ein anderer Kandidat als potentieller Updatepartner in Frage kommt.

Insgesamt kommt der Hand-Tracking Ansatz mit ICP und der Global-Tracker ohne die Bilderfolge mit falschem Initialcluster auf eine ähnliche Anzahl an gesamten Fehlern (ICP: 39 und Global-Tracker: 38).
Der Globale-Tracker weist diese jedoch auch bei korrekten Clustern auf, sodass sich aus dem Durchschnittswert ergibt, dass die Gesamtzahl der Fehler bei falscher Clusterbildung vergleichsweise gering ausfällt. Dies lässt sich daraus schließen, dass jeder Finger unabhängig betrachtet und geupdatet wird. So werden beispielsweise zwar die Daumen der beiden Hände vertauscht, aber dies hat im Gegensatz zum ICP keinen Einfluss auf die restlichen Finger.

Die Laufzeiten von KNN, Minimale-Distanzen-Tracker und ICP sind relativ konstant. Beim Globalen-Tracker steigt der Rechenaufwand mit der Zahl der aufliegenden Finger im maximalen Updateradius stark an. Je näher die Finger zusammen kommen, desto größer der Rechenaufwand. In folgendem Graph (siehe Abb. 34) ist dies mit einem Updateradius von 10 cm verdeutlicht.

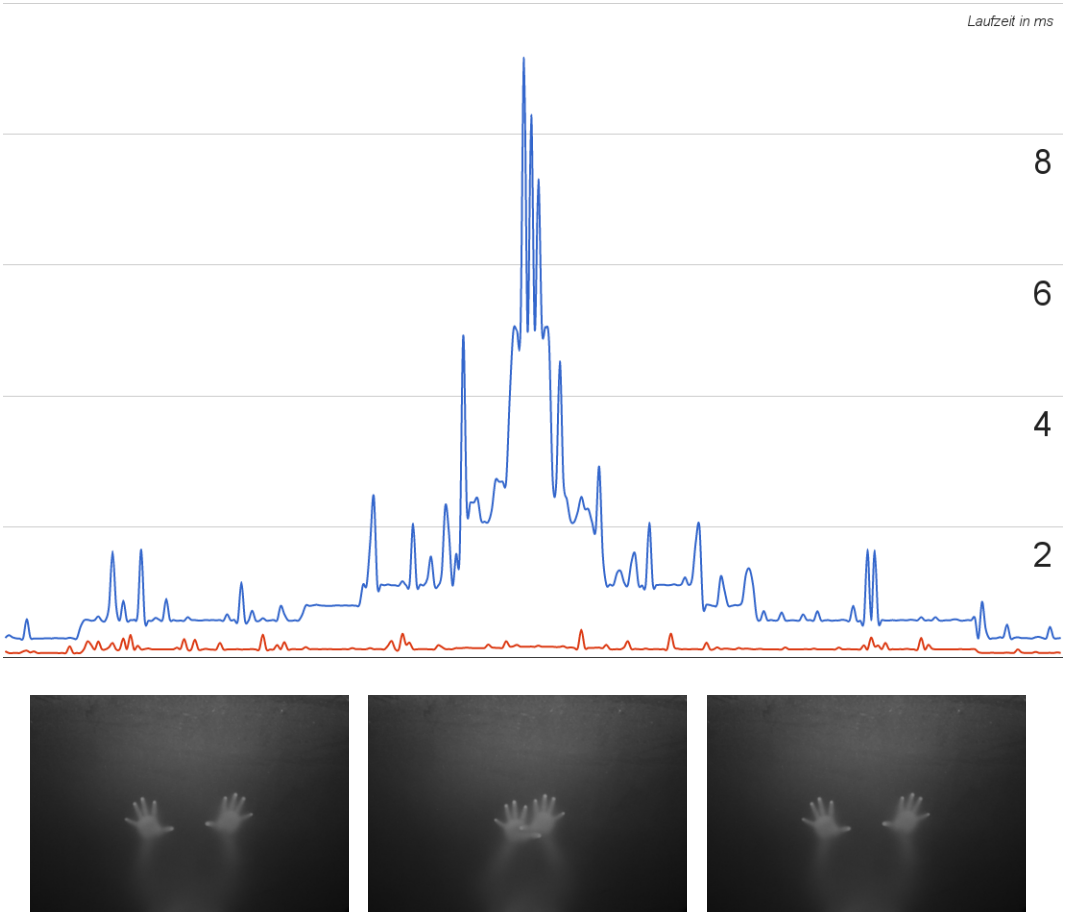


Abbildung 34: Vergleich der Laufzeiten in ms (Y-Achse) des einfachen Globale-Trackers (Blau) mit dem optimierten Global-Tracker (Orange) in Abhängigkeit der sich in der Nähe befindenden Finger (X-Achse). Die Anzahl der Finger beziehen sich dabei auf die visualiserte Bilderfolge.

Der Anstieg des einfachen Global-Trackers (Blau) ist deutlich Wahrnehmbar. Von anfänglichen 0,3 ms steigt die Laufzeit bei nahen Händen in der Mitte auf bis zu 9,17 ms an. Dies entspricht einem Anstieg auf das 31. Fache. Die optimierte Variante (Orange) ist dabei deutlich konstanter und liegt bei maximal 0,43 ms.

Die Auswertung der Bilderfolgen bestätigte, dass der ICP Algorithmus bei korrekter Clusterbildung auch stets die korrekte Zuordnung finden kann. Problematisch waren nur die Frames, in denen die Hände zu nah beieinander lagen und es zur inkorrekten Clusterbildung kam. Während dieses Falls kam es erstmalig bei ermittelten Geschwindigkeiten von 66,6 cm pro Sekunde zu falschen Zuordnungen. Die Problematik dabei ist stark abweichende Initialzuordnungen

der Finger zu Kandidaten anderer Hände, wie beispielsweise das Vertauschen der beiden Daumen. Dies beeinflusst die Transformation der Punktwolke so stark, dass die Zuordnung der Finger darunter leidet.
Um den Einfluss von falschen Initalvermutungen der Verbindungen zu vermindern, ist es ratsam den ICP Algorithmus in verschiedensten Konstellationen, also mit maximal zwei möglichen Verbindungen der Finger zu den Kandidaten oder mit der vermuteten Position, zu verwenden. Darüber hinaus sollte am Ende die Zuordnungen ausgewählt werden, bei der man für alle Hände in unmittelbarer Nähe auf die höchste Anzahl an Updates kommt. Dies stabilisiert das Tracken mit ICP bei falscher Clusterbildung.
Die Problematik falscher Initialcluster kann über den Hand-Tracking Ansatz nicht gelöst werden, da die Handzuordnung nicht bekannt ist. Um dennoch die Punkte zu tracken, kommen nur die reinen Punkt-Tracking Verfahren wie KNN, Minimale-Distanzen-Tracker und der Global-Tracker in Frage. Dabei greife ich in Anbetracht der deutlich besseren Ergebnisse zu dem optimierten Global-Tracker. Hier ist es ratsam, ein Abbruchkriterium fest zu setzen, um bei zu vielen nahen Fingern nicht auf langen Rechenzeiten hängen zu bleiben und anschließend mit dem kombinierten KNN Algorithmus zu tracken.

Die Studie zeigt, dass alle Methoden ihre Schwachstellen haben und keine von mir beschriebene Methode alle Bilderfolgen bei allen Abtastraten korrekt zuordnen konnte. So bleibt das Tracken von Fingern und Händen bei falscher Clusterbildung und hohen Geschwindigkeiten ein bestehendes Problem.

Kapitel 6

Fazit

Die Evaluierung zeigt, dass die entwickelte Touch-Software und damit der neue Hand-Tracking Ansatz bei konstanter Rechenzeit bessere Ergebnisse erzielt, als bei ähnlichen Methoden, die von quelloffenen Referenzsystemen genutzt werden. Weder die Community Core Vision Version 1.5, noch reacTIVision oder die Touch-Software im Tabletop PixelSense von Microsoft kann mit der Erkennung und dem Tracking meiner Software mithalten. Leider basiert diese Behauptung in Bezug auf reacTIVision und PixelSense nur auf meiner Erfahrung und konnte aufgrund fehlender Vergleichsmittel noch nicht mit Zahlen belegt werden.
Somit bildet vorrangig nun nicht mehr das Tracking den limitierenden Faktor, sondern die Erkennung, welche durch das anfällige Infrarotlicht immer noch störende Fehler verursachen kann.

Um das Tracking zu ermöglichen wird aus dem Bild die zusätzliche Informationen der Handzugehörigkeit der einzelnen Finger genutzt und diese, aufgrund der anatomischen Eigenschaften von Fingern, im Verbund betrachtet. Diese sogenannten Cluster werden über den in der Arbeit angepassten ICP Algorithmus zu Händen getrackt. Um die Handzugehörigkeit zu extrahieren wird das Kamerabild über den MSER Algorithmus in hierarchische Regionen aufgeteilt, welche über den Intensitätsverlauf im Bild bestimmt werden und erkannten Fingerspitzen der darüberliegenden Hand, sowie zu Armregionen, zugeordnet werden. Um Finger-, Hand- und Armregionen zu bestimmen und anschließend die Korrektheit eines gefundenen Clusters zu prüfen, ist ein eigenes Handmodell definiert, welches an anatomisch korrekte Werte angelehnt ist.

Die implementierte Touch-Software wurde bisher nur bei dem selbstgebauten Tabletop der Bauhaus-Universität in Weimar und dem Microsoft Surface PixelSense getestet, unterstützt aber theoretisch jegliche Systeme auf Basis Diffuser

Illumination. Dabei werden die Parameter für das Tracking und das Erkennen mithilfe des Handmodells über die von der genutzten Kameraauflösung verwendete Framerate und der definierten Displaygröße automatisch angepasst. Um die Erkennung weiter zu stabilisieren müssen lediglich vier Parameter über die Benutzeroberfläche manuell eingestellt werden.
Meine Software unterstützt bereits eine große Anzahl an möglicher Eingabegeräte und ermöglicht über eine generische Softwarestruktur die einfache Einbindung weiterer Geräte.

Die hier vorgestellte Arbeit beinhaltet eine Software-Entwicklung und den Test für das Tracken von Finger-Gesten für professionelle großformatige Anwendungen durch mehrere Nutzern gleichzeitig. Diese Untersuchungen haben bestätigt, dass es ein weitaus besserer Ansatz ist, als es bislang frei verfügbare Touch-Auswertungssysteme bieten. Das Potential für Erweiterungen und Verbesserungen 7 ist ausführlich dargestellt, um so die Stabilität und sehr genaue Interaktionen zu garantieren.

Kapitel 7

Zukünftige Arbeit

Trotz der zufriedenstellender Ergebnisse sind während der Bearbeitung des Themas weitere Vorschläge zur Verbesserung des Ansatzes von mir ausgearbeitet wurden, welche aus kapazitiven Gründen nicht umgesetzt werden konnten und in diesem Kapitel vorgestellt werden.

7.1 Methodische Verbesserungen

Der fehleranfälligste und unberechenbarste Teil der Software ist die Erkennung von Fingern und Händen. Trotz Hintergrundsubtraktion und Helligkeits-Normalisierung ist die Erkennung von Parametern und System abhängig und muss immer an die jeweilige Ausleuchtung angepasst werden.
Damit sich die Software dynamisch an die Umgebungsbeleuchtung anpasst, wird jede Sekunde ein neues Hintergrundbild aufgenommen, sofern keine Objekte erkannt wurden. Plötzliche Veränderungen der Beleuchtungssituation (Fenster öffnen, direkte Sonnenstrahlen) werden oftmals als Objekte erkannt und müssen manuell über die Aufnahme eines neuen Hintergrundbildes wieder eliminiert werden. Hier würde eine dynamische Hintergrundssubtraktion helfen.

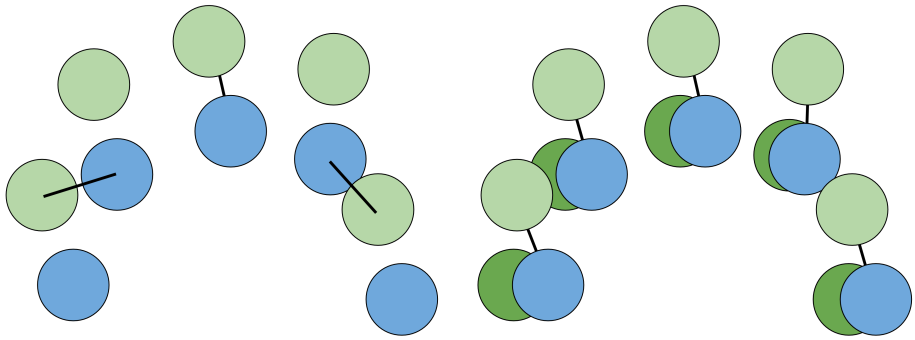
Um die Erkennung für Hände- und Finger zu stabilisieren, sollten die notwendigen Parameter nicht mehr für die gesamte Tischfläche fest definiert werden, sondern für jede Region und somit an die unterschiedliche Ausleuchtung des Systems dynamisch angepasst werden. Dafür bietet sich beispielsweise eine gesteuerte Hand-Kalibrierung an. Der Nutzer kann vor Inbetriebnahme dazu aufgefordert werden, die Hand an ausgewählte Positionen abzulegen und die Software extrahiert automatisch die notwendigen Parameter, damit Finger und Handfläche an dieser Stelle robust erkannt werden. Anstatt anschließend bei der Erkennung nur auf einen Wert für den gesamten Bereich zurückzugreifen,

wird auf die positionsgebundenen Werte zurückgegriffen und dazwischen interpoliert.

Um das Clustering zu verbessern und diese bei nahen Händen bereits früher zu erkennen, biete sich die Kombination mehrerer Cluster-Methoden an (siehe 3.1). Dies würde die Handzugehörigkeit der Finger schneller erkennbar machen und realisieren, dass Handgesten bei nahen Händen bereits früher erkannt werden.
Ein noch nicht getesteter Vorschlag, um die Fingerrichtung zu bestimmen und somit die Handzugehörigkeit zu extrahieren, ist es für jeden Finger den Threshold für die Binarisierung solange zu erhöhen, bis die Kontur des Fingers mit einer anderen Nicht-Finger-Kontur, potentiell die der Hand, verschmilzt. Nun hat man zwangsweise für jeden erkannten Finger eine ellipsenförmige Kontur und kann die Fingerrichtung nach Dang et al. bestimmen oder direkt die Schnittstelle der ermittelten Nicht-Finger-Kontur als Richtung verwenden [Dang and André, 2011].
Zudem kann eine ermittelte Fingerrichtung auch das Clustering über den Komponentenbaum stabilisieren. Sobald ein Finger nicht über den Komponentenbaum zugeordnet werden kann, wird dieser bisher dem Cluster mit der nächsten Handfläche zugeordnet. Die zusätzliche Betrachtung der Richtung lässt ein stabileres Clustering vermuten.

Auch die anschließende Überprüfung der Cluster über das definierte Handmodell bietet Verbesserungspotential. Dafür muss das verwendete Handmodell nur noch weiter ausgefeilt werden und Werte, wie beispielsweise die Relation der Finger untereinander, zusätzlich beschrieben und überprüft werden.

Um den KNN Algorithmus mit vorhergesagten Positionen bei abrupten Bewegungen zu stabilisieren, bietet es sich an, sowohl die vermutete als auch die Original-Position zu verwenden und mit $K = 3$ zu tracken.
In folgender Abbildung ist schematisch eine linke Hand abgebildet, welche bei einem abrupten Stoppen der Bewegung einerseits mit $K = 1$ mit der vermuteten Position (Hellgrün) und andererseits für $K = 3$ zusätzlich mit der aktuellen Position (Dunkelgrün) getrackt wird. Dabei werden die neuen Fingerpositionen der stehengebliebenen Hand (Hellblau) unmittelbar in der Nähe der aktuellen Position (Dunkelgrün) erkannt.



(a) KNN mit K = 1 (b) KNN mit K = 3

Abbildung 35: Vergleich Zuordnung für K = 1 (vorhergesagte Position) und K = 3 (vorhergesagte und originale Position).

Wie im obigen Bild dargestellt, würde das Tracken mit der vermuteten Position für K = 1 auf eine falsche Zuordnung hinauslaufen. Die vermuteten Punkte wandern in die Bewegungsrichtung weiter nach oben, sodass Daumen und kleiner Finger fälschlicherweise dem Zeige- und Ringfinger zugeordnet werden.
Betrachtet man aber sowohl die aktuelle, als auch die vermutete Position und trackt mit K = 3, geht das Ganze wie im unteren Teil abgebildet wunderbar auf. Im dargestellten Fall ist sowohl die aktuelle, als auch die vorhergesagte Position des korrekten Fingers vorhanden und somit ordnungsgemäß zugewiesen.
Meine Vermutung ist, dass dieser Ansatz auch bessere Ergebnisse erzielt, als die von mir vorgestellte schlichte Kombination der KNN Varianten. Dies gilt es aber zu testen.

Um die Laufzeit des Global-Trackers weiter zu verringern, können Suchverfahren, wie zum Beispiel A* verwendet werden, um die Suche nach der optimalen Lösung frühzeitig abbrechen zu können. Dabei werden die Verbindungen anhand der geringsten Länge abgearbeitet und es wird versucht die optimale Lösung zu finden, ohne alle Verbindungen miteinander zu vergleichen.

Bisher wird beim Tracking von falschen Clustern auf die Kombination unterschiedlicher ICP Ansätze gesetzt. Anhand meiner Evaluierung lässt es sich aber nicht abschätzen, ob das auch der richtige Ansatz ist. Alternativ kommt der optimierte Global-Tracker in Frage. Dies müsste in einer weiteren Studie nochmal ausführlich getestet werden.

Beim Iterative Closest Point Algorithmus ist zu beachten, dass dieser zwar immer gegen ein lokales Minimum der Abstände konvergiert, jedoch muss dies

nicht zwingend das gewünschte globale Minimum sein. Dies lässt sich auch bei den fehlerhaften Zuordnungen reproduzieren. Ein effizienter Algorithmus, welcher stets das globale Minimum zweier Punktwolken findet, wird vermutlich auf stabilere Ergebnisse kommen. Leider kam ich bei meine Recherche nach einem effizienten Algorithmus bisher auf keine Alternative zum ICP.

Wie die durchgeführte Evaluierung verdeutlicht, ist die Verwendung der geringsten Gesamtdistanz, neben der Anzahl an Updates, nicht optimal, da der Globale-Tracker nicht immer auf korrekte Ergebnisse kommt. Während der Evaluierung konnte festgestellt werden, dass sich die die Verbindungen zwischen korrekten korrespondierenden Punktpaaren in der Regel nie oder nur mit einem sehr kleinem Winkel kreuzen. Das Betrachten dieser Eigenschaft, um zwischen Trackingergebnissen zu vergleichen und die optimale Lösung auszuwählen, wird erwartbare Vorteile mit sich bringen. Dadurch wird vor allem der Globale-Tracker, aber auch die Kombination der unterschiedlichen Tracking-Methoden, auf stabilere Ergebnisse kommen.

7.2 Anwendungen

Die entwickelte Software extrahiert mehr Informationen als Referenzsysteme. Anstatt nur die Position weiterzugeben, wird die Handzugehörigkeit der einzelnen Finger bestimmt und zusätzlich, sofern möglich, die Art der Finger (Daumen, Zeigefinger, Mittelfinger, Ringfinger und kleinem Finger) und Hände (Links, Rechts) klassifiziert. Darüber hinaus wird die Richtung der Hände berechnet und weitergegeben.
Die aufbauende Anwendung kann somit das Potential von komplexen Gesten ausschöpfen und zwischen linker und rechter Handgeste unterscheiden.
Da die Klassifizierung bisher nur visuell dargestellt und geprüft wurde, ist die fehlerfreie Verwendung für Gesten derzeit nicht absehbar. An dieser Stelle müsste die Klassifizierung noch ausführlicher getestet werden.
Zudem garantiert die Software, bis zu einer bestimmen Entfernung, die Erkennung von schwebenden Händen. Diese Information kann zusätzlich von Anwendungen verwendet werden, um beispielsweise das Aufnehmen und Ablegen von Objekten zu ermöglichen. Durch eine genauere Parametrisierung der Erkennung sollte auch zwischen aufliegenden und schwebenden Fingern unterschieden werden können.

Da sich durch das hier vorgestellte System die Handrichtung bestimmen lässt, können Gesten auch Nutzern bzw. Tischseiten zugeordnet werden. Dies erleichtert das kollaborative Arbeiten an ein und demselben Display. Jeder Nutzer

steht an seiner Tischseite und seine Gesten werden nur kombiniert, wenn sie von seinen eigenen Händen ausgeführt werden. Mithilfe eines entsprechenden Systems, bei dem jeder Nutzer sein eigenes Bild dargestellt bekommt, könnte jeder Nutzer auch unabhängig auf dem Tisch arbeiten ohne von den Interaktionen eines Anderen gestört zu werden.

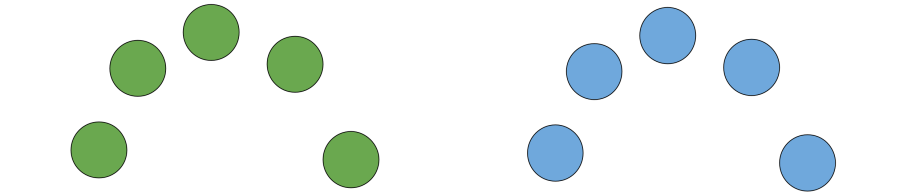
7.3 Implementierung

Bisher wird nur die Hintergrundsubtraktion parallelisiert. Weitere Vorteile sind zudem bei der parallelen Betrachtung der verbundenen Komponenten oder dem Tracken der vollkommen unabhängigen Cluster zu erwarten. Leider ließen sich bei ersten Versuchen mit OpenMP [OpenMP,] bisher keine Verringerung der Laufzeit feststellen. An dieser Stelle muss nochmal genauer nachgeforscht werden.

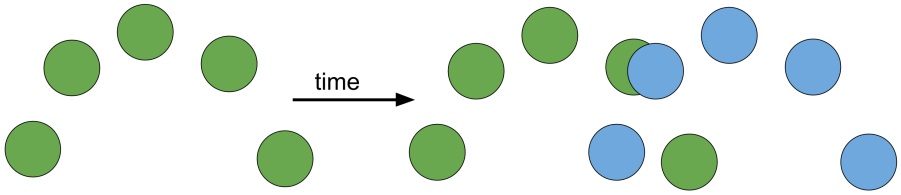
7.4 Erweiterungen

Alexander Holzammer et al. haben eine Variante vorgestellt, um Diffuse Illumination mit FTIR zu kombinieren und dadurch beide Vorteile zu verbinden [Holzammer et al., 2009]. Einerseits bleibt das komplette Bild erhalten, aber um speziell die Erkennung der Fingerspitzen zu vereinfachen, findet zusätzlich das FTIR System Verwendung. Leider konnte ich dieses System aufgrund von erhöhten Hardware-Anforderungen bisher noch nicht umsetzten.

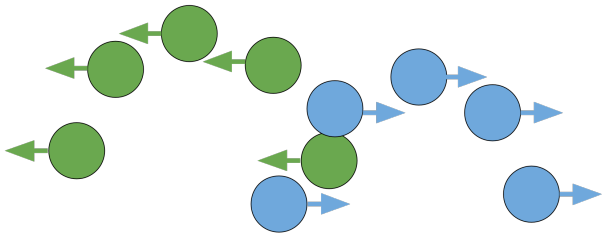
Des Weiteren ist der Ansatz des Hand-Trackings nicht auf die Verwendung eines Systems mit Diffuser Illumination beschränkt. Die Handzugehörigkeit der Finger lässt sich auch ohne Bildinformationen über Kohärenzen bestimmen. In Abbildung 36 ist die Idee visualisiert dargestellt.



(a) Örtliche-Kohärenz



(b) Zeitliche-Kohärenz



(c) Richtungs-Kohärenz

Abbildung 36: Möglichkeit Finger über örtliche (a), zeitliche (b) und Richtungs-Köhärenzen (c) zu clustern

Finger einer Hand befinden sich immer in einem definierten maximalen Radius (a), werden zeitnah (b) aufgelegt und bewegen sich anschließend im Verbund in ähnliche Richtung (c). Diese Informationen lassen sich über einfaches Blob-Tracking extrahieren und dadurch Hände vermuten. Um den Suchraum und dadurch potentiell das Ergebnis vom Blob-Tracking zu verbessern, können gefundene Hände erst über ICP und anschließend die unqualifizierten Finger über Blob-Tracking erkannt werden. Dies hätte den Vorteil, dass mit dem stabileren Hand-Tracking einerseits bessere Ergebnisse erzielt werden, andererseits aber auch der Suchraum für das anschließende Blob-Tracking verkleinert wird. Zusätzlich wird dabei die Zuordnung stabilisiert.

Anhang A

Anhang 1

(a) Erfasstes Kamerabild.

(b) Kalibriertes Kamerabild.

(c) Gefundene Verbundene Komponenten.

(d) Gefundene Verbundene Komponenten.

(e) Extrahiertet Komponentenbaum.

(f) Gefundene Cluster.

Abbildung 37: Software Pipeline: Bildverarbeitungsschritte

Anhang B

Anhang 2

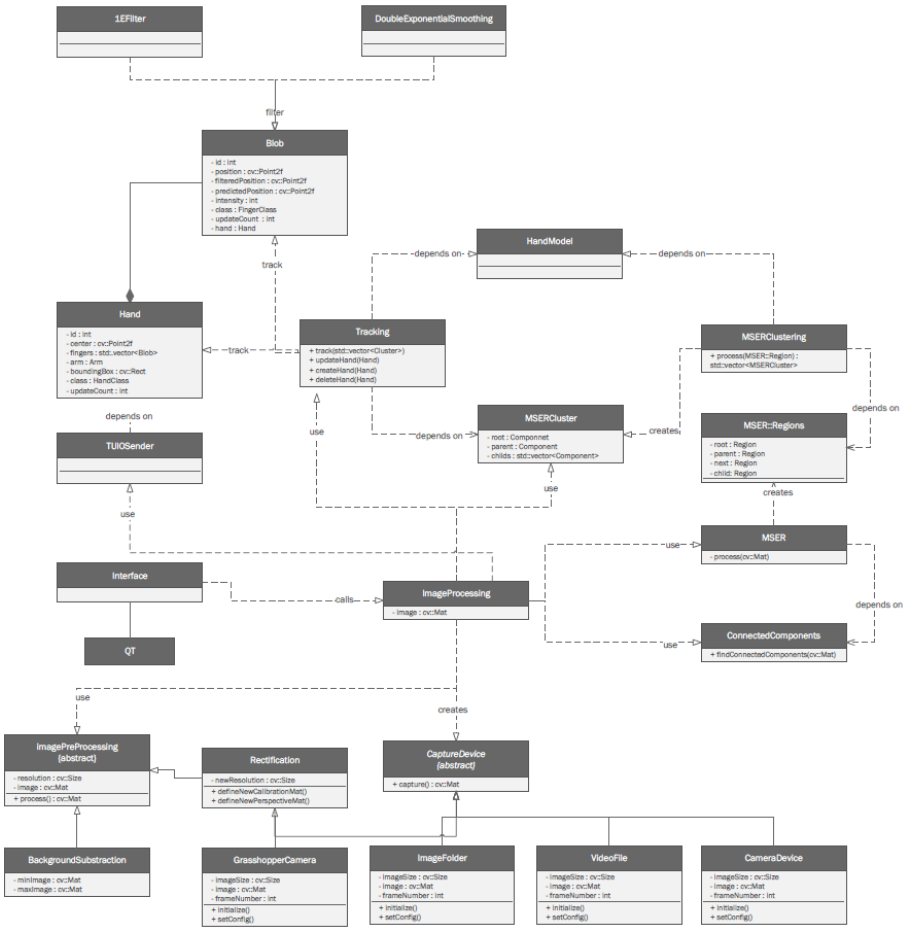


Abbildung 38: Klassendiagramm zur Software

Anhang C

Anhang 3

C.1 Rotation

FPS	Anzahl an Bildern	Rotation pro Frame in °	Distanz pro Frame in cm
120	89	2,02	0,3
60	45	4,0	0,6
30	23	7,83	1,2
20	15	12	1,8
10	8	22,5	3,6

Tabelle C.1: Ergebnisse: Rotations-Bewegung

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate Frame	Zeit in ns
KNN					
120	0	0	0	0	29576
60	0	0	0	0	27062
30	0	0	0	0	30364
10	6	12	18	2,6	32040
KNN (vorhergesagte Position)					
120	0	0	0	0	28156
60	0	0	0	0	28559
30	1	1	2	0,1	32890
10	6	13	19	3,1	28115
KNN (Kombiniert)					
120	0	0	0	0	41316
60	0	0	0	0	41330
30	0	0	0	0	43673
10	6	12	18	3	45338
Minimale-Distanzen-Tracker (Kombiniert)					
120	0	0	0	0	11182
60	0	0	0	0	10888
30	0	2	2	0,1	10100
10	0	14	14	2,3	9936
Global-Tracker (Optimiert)					
120	0	0	0	0	11182
60	0	0	0	0	10888
30	0	2	2	0,1	10100
10	0	14	14	2,3	9936
ICP (Hand-Tracking)					
120	0	0	0	0	63895
60	0	0	0	0	65305
30	0	0	0	0	68967
10	0	0	0	0	60921

C.2 TwoHandsTo

FPS	Anzahl an Bildern	Distanz pro Frame in cm
120	37	0,54
60	19	1,05
30	9	2,22
10	3	6,67

Tabelle C.2: Ergebnisse: Zwei Hände zusammen und auseinander

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate Frame	Zeit in ns
KNN (Kombiniert)					
120	0	0	0	0	58362
60	0	0	0	0	65196
30	18	30	48	6	55935
10	10	3	13	6,5	61212
Minimale-Distanzen-Tracker (Kombiniert)					
120	0	0	0	0	19904
60	0	0	0	0	20704
30	0	21	21	2,6	20708
10	6	19	19	9,5	17186
Global-Tracker (Optimiert)					
120	0	0	0	0	9458
60	0	0	0	0	29371
30	0	4	4	0,5	96138
10	6	10	10	5	476787
ICP (Hand-Tracking)					
120	0	0	0	0	66608
60	0	0	0	0	72461
30	1	8	9	1,125	70670
10	1	2	3	1,5	83842

C.3 TwoHandsFrom

FPS	Anzahl an Bildern	Distanz pro Frame in cm
120	37	0,54
60	19	1,05
30	9	2,22
10	3	6,67

Tabelle C.3: Ergebnisse: Zwei Hände auseinander und zusammen

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate Frame	Zeit in ns
KNN (Kombiniert)					
120	0	0	0	0	58362
60	0	0	0	0	65196
30	18	30	48	6	55935
10	10	3	13	6,5	61212
Minimale-Distanzen-Tracker (Kombiniert)					
120	0	0	0	0	19904
60	0	0	0	0	20704
30	1	21	22	2,75	20708
10	6	19	19	9,5	17186
Global-Tracker (Optimiert)					
120	0	0	0	0	9458
60	0	0	0	0	29371
30	0	4	4	0,5	96138
10	6	10	10	5	476787
ICP (Hand-Tracking)					
120	0	0	0	0	72472
60	0	0	0	0	64920
30	2	3	5	1 (ab 3 Frame Cluster)	78428
10	4	0	4	1 (ab 2 Frame Cluster)	99556

C.4 Sechs Finger zusammen und auseinander

FPS	Anzahl an Bildern	Distanz pro Frame in cm
120	63	0,48
60	33	0,91
30	15	2
10	5	6

Tabelle C.4: Ergebnisse: Sechs Finger zusammen und auseinander

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate Frame	Zeit in ns
KNN (Kombiniert)					
120	0	0	0	0	44055
60	0	0	0	0	41677
30	2	2	4	0,29	43030
10	8	12	20	5	40033
Minimale-Distanzen-Tracker (Kombiniert)					
120	0	0	0	0	11544
60	0	0	0	0	12436
30	0	9	9	0,64	11476
10	6	11	11	2,75	11336
Global-Tracker (Optimiert)					
120	0	0	0	0	6901
60	0	0	0	0	7190
30	0	0	0	0	15919
10	6	4	4	1	39419
ICP (Hand-Tracking)					
120	0	0	0	0	60629
60	0	0	0	0	65339
30	0	0	0	0	59857
10	2	8	10	2,5	65934

C.5 Einen Finger zum Mittelpunkt einer anderen Hand hin- und wieder wegbewegen

FPS	Anzahl an Bildern	Distanz pro Frame in cm
120	35	0,57
60	17	1,18
30	9	2,50
10	3	6,67

Tabelle C.5: Ergebnisse: Einen Finger zum Mittelpunkt einer anderen Hand hin- und wieder wegbewegen

FPS	Verlorene Finger	Falsche Zuordnungen	Gesamte Fehlerrate	Fehlerrate Frame	Zeit in ns
KNN (Kombiniert)					
120	0	0	0	0	39952
60	0	0	0	0	42981
30	1	0	1	0,125	50353
10	1	0	1	0,5	41108
Minimale-Distanzen-Tracker (Kombiniert)					
120	0	0	0	0	10890
60	0	0	0	0	11506
30	0	9	9	0,64	9752
10	6	11	11	2,75	10216
Global-Tracker (Optimiert)					
120	0	0	0	0	9618
60	0	0	0	0	32474
30	0	0	0	0	135002
10	6	4	4	1	153749
ICP (Hand-Tracking)					
120	0	0	0	0	108065
60	0	0	0	0	107642
30	0	0	0	0	108473
10	2	8	10	2,5	117759

Literaturverzeichnis

[ct:, 2008] (2008). Finger-fertig? *c't*, 14:150–155.

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). Method for registration of 3-d shapes. In *Robotics-DL tentative*, pages 586–606. International Society for Optics and Photonics.

[Casiez et al., 2012] Casiez, G., Roussel, N., and Vogel, D. (2012). 1€ filter: a simple speed-based low-pass filter for noisy input in interactive systems. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2527–2530. ACM.

[Dang and André, 2011] Dang, C. T. and André, E. (2011). Usage and recognition of finger orientation for multi-touch tabletop interaction. In *Human-Computer Interaction–INTERACT 2011*, pages 409–426. Springer.

[Ewerling et al., 2012] Ewerling, P., Kulik, A., and Froehlich, B. (2012). Finger and hand detection for multi-touch interfaces based on maximally stable extremal regions. In *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*, ITS '12, pages 173–182, New York, NY, USA. ACM.

[Ho,] Ho, N. Finding optimal rotation and translation between corresponding 3d points.

[Holzammer et al., 2009] Holzammer, A., Hahne, U., and Alexa, M. (2009). *Combining diffuse illumination and frustrated total internal reflection for touch detection*. PhD thesis, Beilin: Technology University of Berlin.

[Kaltenbrunner, 2009] Kaltenbrunner, M. (2009). reactivision and tuio: a tangible tabletop toolkit. In *Proceedings of the ACM international Conference on interactive Tabletops and Surfaces*, pages 9–16. ACM.

[Kaltenbrunner et al., 2005] Kaltenbrunner, M., Bovermann, T., Bencina, R., and Costanza, E. (2005). Tuio: A protocol for table-top tangible user interfa-

ces. In *Proc. of the The 6th Int'l Workshop on Gesture in Human-Computer Interaction and Simulation*, pages 1–5.

[KNN,] KNN. Example of k-nearest neighbour classification.

[Kulik et al., 2011] Kulik, A., Kunert, A., Beck, S., Reichel, R., Blach, R., Zink, A., and Froehlich, B. (2011). C1x6: A stereoscopic six-user display for co-located collaboration in shared virtual environments. In *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, pages 188:1–188:12, New York, NY, USA. ACM.

[LaViola, 2003] LaViola, J. J. (2003). Double exponential smoothing: an alternative to kalman filter-based predictive tracking. In *Proceedings of the workshop on Virtual environments 2003*, pages 199–206. ACM.

[Liu, 2010] Liu, Q. (2010). Tuio, touchlib, reactivision and community core vision. *University of California, Santa Barbara*.

[Matas et al., 2004] Matas, J., Chum, O., Urban, M., and Pajdla, T. (2004). Robust wide-baseline stereo from maximally stable extremal regions. *Image and vision computing*, 22(10):761–767.

[Nistér and Stewénius, 2008] Nistér, D. and Stewénius, H. (2008). *Computer Vision – ECCV 2008: 10th European Conference on Computer Vision, Marseille, France, October 12-18, 2008, Proceedings, Part II*, chapter Linear Time Maximally Stable Extremal Regions, pages 183–196. Springer Berlin Heidelberg, Berlin, Heidelberg.

[OpenMP,] OpenMP. Openmp - an api for multi-platform shared-memory parallel programming in c/c++ and fortran.

[PointGrey, 2015] PointGrey (2015). *Grasshopper: Datenblatt*.

[Schöning et al., 2010] Schöning, J., Hook, J., Bartindale, T., Schmidt, D., Oliver, P., Echtler, F., Motamedi, N., Brandl, P., and von Zadow, U. (2010). Building interactive multi-touch surfaces. In *Tabletops-Horizontal Interactive Displays*, pages 27–49. Springer.

[Touchlib,] Touchlib. Touchlib - a multi-touch development kit.

[TUIO,] TUIO. Tuio - common protocol and api for tangible multitouch surfaces.

[Wang et al., 2008] Wang, F., Ren, X., and Liu, Z. (2008). A robust blob recognition and tracking method in vision-based multi-touch technique. In *Parallel and Distributed Processing with Applications, 2008. ISPA'08. International Symposium on*, pages 971–974. IEEE.