Fachhochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

Diplomarbeit

„Entwicklung eines Motion Tracking-Systems zur Positionssteuerung per Handbewegung“

Diplomarbeit zur Erlangung des akademischen Grades Diplominformatiker an der Fachhochschule für Technik und Wirtschaft Berlin

Eingereicht von Alexander Miehlke

1. Betreuer: Prof. Dr.-Ing. Thomas Jung
2. Betreuer: Prof. Dr. Horst Hansen

Berlin, 4. September 2007

Inhaltsverzeichnis

1 Einleitung

Hätten sich die Eingabegeräte in der Geschichte des Personalcomputers nicht immer mehr auf den Menschen zubewegt, würden heute nicht 70 Prozent aller deutschen Haushalte einen Computer besitzen. In den Anfängen des Computers war es Spezialisten vorbehalten, ihn allein mit Lochkarten zu steuern. Es folgte die Computertastatur und im Jahre 1968 die erste Computermaus.

Während die Tastatur eine Eingabe in schriftlicher Form ermöglicht, werden mit der Computermaus einfache Gesten wie Zeigen und Auswählen erfasst. Beide Geräte können wie nahezu alle anderen Eingabegeräte natürlich nur einen Bruchteil der menschlichen Kommunikation abbilden, die neben Gestik auch Sprache und Körpersprache beinhaltet. Ein Eingabegerät sollte so nah wie möglich der menschlichen Kommunikation entsprechen.

Mit wachsender Verarbeitungsgeschwindigkeit der Computer entstanden neue Systeme, die neben Handschrift auch gesprochene Sprache verstehen konnten. Ein relativ neues Fachgebiet namens *Computer Vision* ermöglicht es einem Computer zu sehen. Dies bedeutet, dass der Computer durch Technologien der Computer Vision unter anderem in der Lage ist, Objekte in Bildern zu erkennen und zu klassifizieren. So werden z.B. auf Flughäfen Personen und auf Straßen Fahrzeuge identifiziert, in Fabrikationsstraßen fehlerhafte Produkte erkannt oder auch Mimik und Gestik eines Menschen erfasst. Sogenannte *Eye-Tracking-Systeme* verfolgen die Bewegung des Auges und steuern den Cursor auf dem Bildschirm. Auch existieren Systeme, die menschliche Stimmungen am Gesichtsausdruck erkennen können. Dadurch erschließen sich völlig neue Anwendungsbereiche.

In dem Science-Fiction-Film „Minority Report“ aus dem Jahre 2002 bedient Tom Cruise in der Figur des John Anderton einen Computer durch verschiedene Handgesten mit seinen bloßen Händen. Er blättert in digitalen Akten, indem er einzelne Seiten mit seiner Hand ins Blickfeld zieht und wieder beiseite legt. Im Washington der Zukunft enthalten Werbeplakate Kameras, die in Sekundenbruchteilen Passanten anhand ihrer Iris identifizieren und zeitgleich angepasste Werbung anzeigen. Dies alles erscheint dem Kinobesucher als ferne Zukunftsvision, obwohl alle Technologien schon heute zur Verfügung stehen.

1.1 Aufgabenstellung

Im Rahmen dieser Arbeit soll ein Motion Tracking-System entstehen, welches einem Computerbenutzer ermöglicht, den Mauszeiger in nahezu Echtzeit mit seiner Hand zu steuern. Die Bewegung der Hand soll dabei über eine einzelne handelsübliche Videokamera (Webcam), die an dem Computer angeschlossen ist, erfasst werden. Der Anwender soll zur Steuerung lediglich die bloße Hand benutzen. Ein Handschuh oder spezielle Sensoren dürfen nicht zum Einsatz kommen. Das Videobild soll von dem System aufbereitet und

analysiert werden, um letztendlich die Position des Zeigefingers im Videobild zu ermitteln. Die Position des Zeigefingers soll die des Mauszeigers bestimmen. Neben der Positionssteuerung soll es auch möglich sein, einen Mausklick auszuführen. Dies kann z.B. über eine bestimmte Handgeste erfolgen.

1.2 Kapitelübersicht

Dieser Abschnitt gibt einen Überblick über die Kapitel dieser Arbeit:

Grundlagen Die Grundlagen sollen zum besseren Verständnis der nachfolgenden Kapitel beitragen. Der Abschnitt Bildverarbeitung stellt verschiedene digitale Filter vor und befasst sich mit der Segmentierung von Objekten in Bildern als Vorbereitung für die Konturverfolgung und Klassifikation. Zwei weitere Abschnitte stellen Software-Frameworks[1] vor, die für die Implementierung des Systems in Frage kommen.

Analyse In der Analyse werden bereits vorhandene Verfahren und Technologien vorgestellt, die eine visuelle Erkennung von Körperteilen wie der Hand oder des Gesichts ermöglichen. Neben Verfahren, die durch mehrere Kameras die Position der Hand dreidimensional bestimmen können, werden auch Techniken vorgestellt, die Körperteile anhand von Hautfarbe, Bewegung und Infrarotstrahlung erkennen. Nachfolgend werden die Verfahren bewertet und miteinander verglichen. Als Grundlage für den Entwurf folgen die Anforderungen an das System. Unter anderem werden die Anforderungen an die Funktion, die Geschwindigkeit und die Benutzeroberfläche des Systems formuliert.

Entwurf Nachdem beschrieben wurde, wie sich das System in das Betriebssystem integriert, werden die einzelnen Schritte von der Aufnahme des Videobildes bis zur Steuerung des Mauszeigers entworfen. Dabei wird detailliert auf die Kalibrierung des Systems auf die Hautfarbe des Anwenders, die anschließende Bildverarbeitung, die Erkennung der Fingerspitzen und die Errechnung der Bildschirmposition des Mauszeigers eingegangen.

Implementierung Die in den Grundlagen vorgestellten Frameworks werden nun miteinander verglichen und das geeignetste ausgewählt. Als nächstes wird auf die Voreinstellungen der Sprache, der Benutzeroberfläche und des Systems eingegangen. Es werden die wichtigsten Klassen vorgestellt und ihre Funktion anhand von Code-Auszügen erläutert.

Test In diesem Kapitel werden die Ergebnisse von verschiedenen Tests des implementierten Systems zusammengetragen. Dazu werden mehrere Testszenarien erstellt und anschließend bewertet.

[1] Rahmenstruktur, Design-Grundstruktur, die einen beim Erstellen einer Anwendung unterstützt

Zusammenfassung und Ausblick Die Ergebnisse dieser Arbeit werden an dieser Stelle zusammengefasst, bewertet und diskutiert. Anschließend werden Vorschläge für eine Weiterentwicklung des Systems gemacht und ein Ausblick auf mögliche Eingabeformen in der Zukunft gegeben.

2 Grundlagen

Den größten Teil der Grundlagen nimmt die Bildverarbeitung ein. Es werden die Techniken und Verfahren vorgestellt, die für das spätere Verständnis und für die Implementierung des Motion Tracking-Systems notwendig sind. Des Weiteren werden zwei Frameworks beschrieben, mit der sich das System realisieren lässt.

2.1 Bildverarbeitung

Mit der digitalen Bildverarbeitung stehen verschiedene Verfahren zur Verfügung, um die Bildsignale des aufgenommenen Videos aufzubereiten und auszuwerten.

2.1.1 Filter

Digitale Filter gehören zu den lokalen Bildoperationen und überführen ein Eingangsbild durch mathematische Verfahren in ein Ausgangsbild. Ein gewünschter Effekt ist die Hervorhebung oder Unterdrückung benachbarter Pixel. Dabei wird um ein sogenanntes aktuelles Pixel ein Operatorfenster (auch Filtermaske oder Filterkern) H geöffnet. Die Farbwerte in diesem Fenster werden miteinander verknüpft und erzeugen dabei einen Ergebnisfarbwert, welcher im Ausgangsbild eingetragen wird. Anwendungen von Filtern sind beispielsweise das Entfernen von Rauschen (Glättung) oder die Hervorhebung von Kanten. Nachfolgend werden zwei in [BB06] und [Her05] beschriebene Filter behandelt:

Mittelwertfilter und Medianfilter

Der Mittelwertfilter errechnet den mittleren Intensitätswert innerhalb eines Operatorfensters. Dadurch wird eine Glättung der Unebenheiten erreicht und ein vorhandenes Bildrauschen entfernt (siehe Abbildung 2.1). Jedoch werden auch Grauwertübergänge wie Kanten, Punkte und Linien verwischt, wodurch die gesamte Bildqualität reduziert wird.

Auch der Medianfilter entfernt störendes Rauschen aus einem Bild. Im Gegensatz zum Mittelwertfilter bleiben aber die Kanten besser erhalten. Der Medianfilter ist ein sogenannter *Rangordnungsfilter*, der die Bildpunkte im Bereich des Filterkerns nach der Größe des Intensitätswertes sortiert. Der mittlere Wert wird dann zum neuen Intensitätswert im Ausgangsbild.

2.1.2 Morphologische Filter

Mithilfe von morphologischen Filtern kann die Struktur von (Binär-) Bildern gezielt beeinflusst werden. Dabei wird in [BB06] zwischen den beiden Grundoperationen Wachsen

(a) vorher (b) nachher

Abbildung 2.1: Beispiel Mittelwertfilter

und Schrumpfen unterschieden. Durch die Kombination dieser beiden Grundoperationen lassen sich z.B. unerwünschte kleine Strukturen in einem Bild entfernen oder Löcher in Strukturen „stopfen“. Wie bei einem linearen Filter wird der morphologische Filter durch eine Matrix namens *Strukturelement* bestimmt. Dieses Strukturelement besitzt ein eigenes Koordinatensystem mit einem *hot spot* als Ursprung und wird punktweise mit dem Bild verknüpft.

Dilatation

Die Dilatation lässt Strukturen in einem Bild wachsen. Ist im Zentrum des Strukturelements im Ausgangsbild der Bildpunkt gesetzt, so werden im Ausgangsbild alle Bildpunkte des Strukturelements gesetzt.

Die Abbildung 2.2 zeigt, wie die Dilatation die Strukturen des Originalbildes vergrößert. Zur Verdeutlichung wurde die Dilatation dreimal nacheinander durchgeführt.

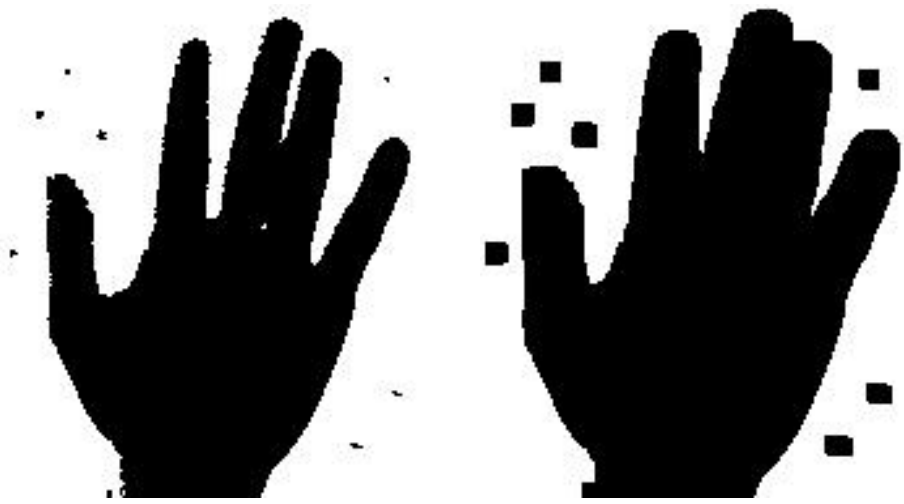
Erosion

Im Gegensatz zur Dilatation lässt die Erosion Stukturen in einem Bild schrumpfen. Dabei wird das Strukturelement auf das zu bearbeitende Bild gelegt. Nur dort, wo sich das Strukturelement vollständig mit dem Teilbereich des Bildes deckt, wird ein entsprechender Bildpunkt im Ausgangsbild gesetzt.

Die Abbildung 2.3 zeigt, wie die Erosion die Strukturen des Originalbildes verkleinert bzw. kleine Strukturen verschwinden lässt. Zur Verdeutlichung wurde die Erosion dreimal nacheinander durchgeführt.

Opening und Closing

Durch die Kombination von Dilatation und Erosion lassen sich weitere morphologische Operationen durchführen. Beim Opening wird zunächst eine Erosion und anschließend eine Dilatation durchgeführt. Dabei wird jeweils dasselbe Strukturelement verwendet. Als



(a) vorher (b) nachher

Abbildung 2.2: Beispiel Dilatation

(a) vorher (b) nachher

Abbildung 2.3: Beispiel Erosion

Ergebnis werden kleine Strukturen eliminiert, große Strukturen bleiben dagegen unverändert. Demnach eignet sich das Opening gut für das Entfernen von Rauschen. Darüber hinaus werden durch das Opening lose, durch kleine Strukturen verbundene Objekte, voneinander getrennt.

Beim Closing ist die Reihenfolge der Operationen genau umgekehrt. Nach einer Dilatation folgt eine Erosion. Dadurch werden Löcher gestopft und nahe liegende Objekte miteinander verbunden.

Die Abbildung 2.4 zeigt die Auswirkungen der beiden Operationen.

(a) Original (b) Opening (c) Closing

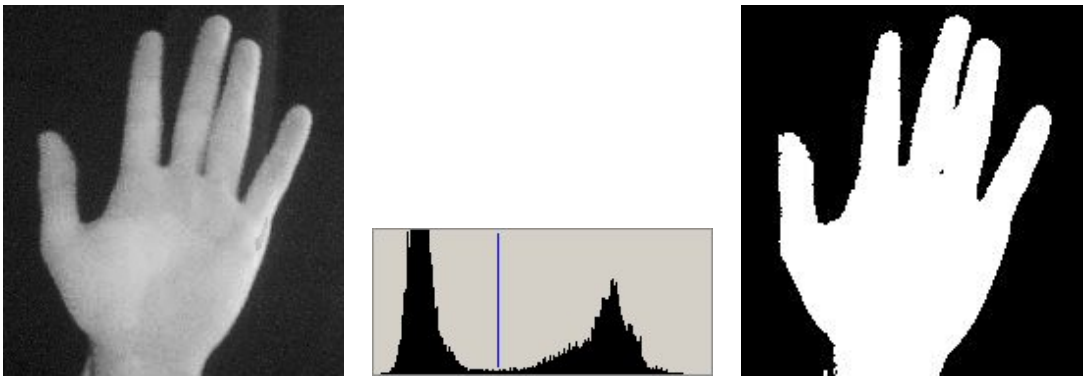
Abbildung 2.4: Beispiel Opening und Closing

2.1.3 Segmentierung

Bei der Segmentierung handelt es sich um die Trennung von Objekten in Bildern von übrigen Bildstrukturen, also vom Hintergrund oder anderen Objekten. Auch die weitere Aufteilung von Unterobjekten gehört zum Begriff der Segmentierung. Im Eingangsbild befinden sich einzelne Pixel, deren isolierte Betrachtung wenig über die im Bild enthaltenen Objekte aussagt. Erst die Findung und Zusammenfassung der Pixel zu einem Objekt ermöglicht eine weitergehende Analyse des Bildes. Nachfolgend sind zwei Verfahren zur Segmentierung aufgeführt:

Schwellenwertverfahren

Falls sich Objekte durch gut voneinander trennbare Grauwerte unterscheiden, können sie mit Hilfe des Schwellenwertverfahrens segmentiert werden. Im einfachsten Fall handelt es sich um ein Bild mit einem hellen Objekt vor einem dunklen Hintergrund. Im Histogramm des Bildes treten dann zwei ausgeprägte Maxima auf. Am Minimum zwischen diesen beiden Maxima legt man nun den Schwellenwert und erhält ein Binärbild, in dem alle schwarzen Pixel zum 1. Objekt und alle weißen zum 2. Objekt gehören. Die Abbildung 2.5 verdeutlicht dies. Abbildung 2.5(b) zeigt dabei das sich ergebene Histogramm, in dem der Schwellenwert blau markiert ist.



(a) Eingangsbild (b) Histogramm (c) Ausgangsbild

Abbildung 2.5: Schwellenwertverfahren

Probleme bei der Segmentierung können durch Bildrauschen entstehen, daher sollte im Zuge der Bildvorverarbeitung beispielsweise ein Mittelwertfilter angewandt werden (siehe 2.1.1). Falls das Bild mehr als zwei verschiedene Grauwerte enthält, kann das Verfahren auf mehrere Schwellenwerte erweitert werden. Auch hier müssen die Minima jeweils zwischen zwei ausgeprägten Maxima liegen, um die Objekte möglichst eindeutig voneinander trennen zu können.

Farbbildsegmentierung

Bei der Farbbildsegmentierung wird das Bild in Bereiche mit gleichen Farbwerten unterteilt. Alternativ kann in einem Bild nach einem bestimmten Farbton gesucht werden, so dass man Vorder- von Hintergrund trennt und ein Binärbild erhält.
Eine Segmentierung eines klassischen Strandfotos ergibt im Idealfall mindestens drei Bereiche: einen gelben (Strand), einen dunkelblauen (Meer) und einen hellblauen (Himmel). Dabei stößt der Computer bei der Unterscheidung zwischen „hellblau“ und „dunkelblau“ schon an technische Grenzen. Als Mensch lassen sich die beiden Blautöne leicht voneinander unterscheiden. Für den Computer sind es dagegen nicht lediglich zwei Blautöne, sondern eine Ansammlung von verschiedenen Farbwerten, die neben Blautönen auch viele andere Farben, wie z.B. Weiß (Gischt, Wolken) und Gelb (Sonne, Farbreflexe) enthalten. Demnach kann eine automatische Segmentierung nur annähernd erfolgen. Eine Möglichkeit ist, eine „Lookup-Tabelle“ zu verwenden, die eine Vielzahl von in Frage kommenden Farbwerten enthält und bestimmt, ob ein Pixel des zu segmentierenden Bildes zum Objekt gehört oder nicht. Es bieten sich dazu zwei Farbräume an:

RGB-Farbraum Der RGB-Farbraum ist der wohl bekannteste Farbraum und basiert auf der Kombination der drei Primärfarben Rot R, Grün G und Blau B. Er findet Verwendung bei Computermonitoren, bei Film- und Fotokameras und bei der Speicherung von Bilddateien. Die RGB-Werte bewegen sich bei Digitalbildern im Bereich von 0 bis 255 und bestimmen jeweils die Intensität. Demnach wird Schwarz durch $(r, g, b) = (0, 0, 0)$ und

Weiß durch $(255, 255, 255)$ dargestellt. Weitere Beispiele: Rot: $(255, 0, 0)$, Grün: $(0, 255, 0)$, Blau: $(0, 0, 255)$, 50% Grau: $(128, 128, 128)$.

Da die unterschiedliche Intensität der Farbkomponenten sowohl den Ton, die Farbsättigung als auch die Helligkeit der sich ergebenden Farbe bestimmt, kann nicht klar beantwortet werden, was die drei Primärfarben physisch bedeuten. Insofern hat die Metrik des RGB-Farbraums nicht viel mit der menschlichen Wahrnehmung gemein. [BB06]

HSB-Farbraum In ihm wird die Farbinformation durch den Farbton (**H**ue), die Farbsättigung (**S**aturation) und Helligkeit (**B**rightness) dargestellt. Damit ähnelt HSB der menschlichen Wahrnehmung, und es fällt im Gegensatz zu RGB leichter, eine gewünschte Farbe aus den drei HSB-Komponenten auszuwählen. Er ist eher an den intuitiven menschlichen Methoden angelehnt. Zwischen RGB und HSB kann problemlos umgerechnet werden. Die Abbildung 2.6 zeigt den HSB-Farbraum in der traditionellen Darstellung als hexagone Pyramide. Dabei entspricht der Helligkeitswert V $(=B)$ der vertikalen Richtung, die Farbsättigung S dem Radius und der Farbton H dem Drehwinkel. W bzw. S entsprechen dem Weiß- bzw. Schwarzwert. Auf den sechs Ecken der Pyramide liegen die Grundfarben Rot (R), Grün (G) und Blau (B) sowie die Mischfarben Cyan (C), Magenta (M) und Gelb (Y). [BB06]

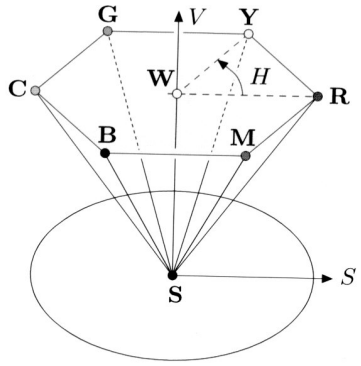
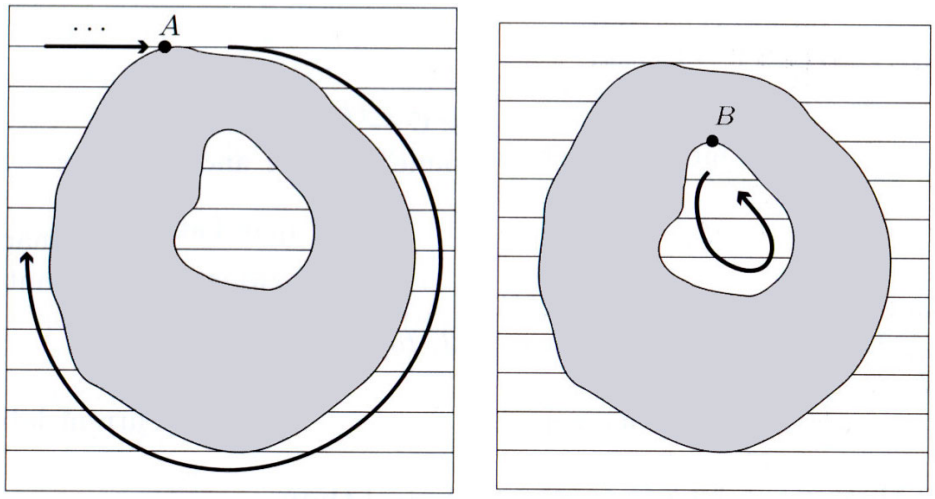


Abbildung 2.6: HSB-Farbraum [BB06]

2.1.4 Konturfindung

Nachdem das Bild segmentiert worden ist und durch die oben vorgestellten Verfahren Regionen vorliegen, gilt es, diese zu lokalisieren. Ein geeignetes Verfahren dazu ist die Konturfindung, die die Pixel entlang von Regionsgrenzen markiert. Man erhält pro Region eine geordnete Folge der Konturpunkte. In [BB06] wird eine kombinierte Regionenmarkierung und Konturfindung beschrieben. Dabei werden in einem Durchlauf sowohl Regionen identifiziert und markiert als auch ihre äußere oder innere Kontur ermittelt. Das als Binärbild vorliegende Eingangsbild wird zeilenweise von links nach rechts und von oben nach unten durchlaufen, bis auf ein Vordergrundpixel *(A)*, d.h. ein Pixel einer Region gestoßen wird (siehe Abbildung 2.7(a)). Die Region wird kreisförmig umlaufen, indem die Randpixel der Region besucht und markiert werden, bis der Ausgangspunkt wieder erreicht ist. Danach

wird die zeilenweise Suche fortgesetzt, bis alle Regionen umlaufen und somit identifiziert und markiert worden sind. Das vorgestellte Verfahren unterscheidet dabei zwischen äußerer und innerer Kontur. Eine äußere Kontur wird, wie beschrieben, vom Wechsel von Hintergrund- auf ein (noch nicht markiertes) Vordergrundpixel erkannt. Eine innere Kontur (also ein „Loch“ in einer Region) wird umgekehrt beim Wechsel von Vordergrund- auf Hintergrundpixel *(B)* erkannt (siehe Abbildung 2.7(b)).



(a) Äußere Kontur (b) Innere Kontur

Abbildung 2.7: Konturfindung [BB06]

2.1.5 Kettencode

Durch die in 2.1.4 beschriebene Konturverfolgung erhält man die Position aller zu einer Region gehörenden Konturpunkte. Diese liegen dann in Form eines Kettencodes (*Chain Codes*) vor. Der Kettencode enthält einen Startpunkt x_s und Richtungscodes, die zusammen die gesamte Kontur der Region beschreiben. Der Richtungscode gibt an, in welcher Richtung der nächste Konturpunkt zu finden ist.

Kettencode bei 4er-Nachbarschaft

Bei der 4er-Nachbarschaft sind zwei Konturpunkte durch ihre Kante miteinander verbunden. Im Kettencode bedeutet eine *0*, dass der nächste Konturpunkt rechts vom aktuellen Konturpunkt zu finden ist. Eine *1* steht für oben, eine *2* für links und eine *3* für unten. Abbildung 2.8 zeigt ein Beispiel eines 4er-Kettencodes.

Kettencode bei 8er-Nachbarschaft

Bei der 8er-Nachbarschaft sind zwei Konturpunkte durch ihre Kanten *oder* ihre Eckpunkte miteinander verbunden. Eine *0* hat dieselbe Bedeutung wie bei der 4er-Nachbarschaft.

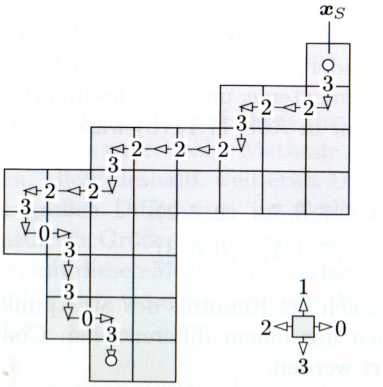


Abbildung 2.8: Kettencode bei 4er-Nachbarschaft: 3223222322303303...111 [BB06]

Eine *1* dagegen bedeutet, dass der nächste Konturpunkt an der oberen rechten Ecke des aktuellen Konturpunktes zu finden ist. Abbildung 2.9 verdeutlicht die restlichen Codes und zeigt ein Beispiel.

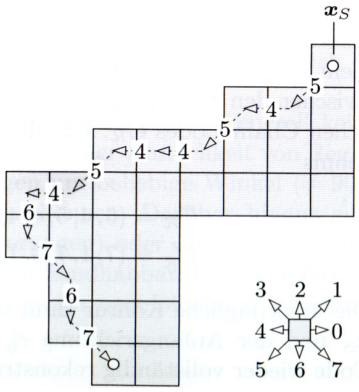


Abbildung 2.9: Kettencode bei 8er-Nachbarschaft: 54544546767...222 [BB06]

Relativer Kettencode

Der Kettencode ist zwar positions-, aber nicht orientierungsabhängig. Er könnte anstatt der Richtungsangabe auch die Richtungsänderung repräsentieren. So gibt der relative Kettencode nicht die Differenz zwischen den Positionen der Konturpixeln an, sondern die Änderungen der Richtung (Krümmung) entlang der Kontur. Für den ersten Konturpunkt muss dann neben der Position auch die Richtung angegeben werden. Der relative Kettencode C'_R lässt sich leicht aus dem normalen Kettencode C_R berechnen, indem die Differenz zwischen dem aktuellen Code c_i und dem folgenden Code c_{i+1} gebildet wird. Ist das Ergebnis negativ, wird je nach Nachbarschaft 4 oder 8 addiert. Der Vorteil des relativen Kettencodes liegt in der besseren Vergleichbarkeit zweier Konturen, da deren Ausrichtung und Position ausgeblendet werden können.

$$c'_i = \begin{cases} (c_{i+1} - c_i) \ mod \ 8 & \text{für} \quad 0 \leq i < M-1 \\ (c_0 - c_i) \ mod \ 8 & \text{für} \quad i = M-1 \end{cases} \tag{2.1}$$

$$\begin{aligned} C_R &= (5,4,5,4,4,5,4,6,7,6,7,...2,2,2) \\ C'_R &= (7,7,7,0,1,7,2,1,7,1,1,...,0,0,3) \end{aligned} \tag{2.2}$$

Mathematische Eigenschaften

Der Vorteil des Kettencodes liegt in der einfachen Berechnung der mathematischen Eigenschaften der beschrieben Kontur. Dadurch lassen sich digitale Bilder oder Bildteile anhand deren Eigenschaften klassifizieren und vergleichen. Dies wird auch als *Mustererkennung* bezeichnet.

Umfang Der Umfang *(Perimeter)* einer zusammenhängenden Region R wird durch die Länge der äußeren Kontur bestimmt. Bei der 4er-Nachbarschaft ist der Umfang identisch mit der Länge des Kettencodes. Bei der 8er-Nachbarschaft berechnet sich der Umfang wie folgt:

$$Perimeter(R) = \sum_{i=0}^{M-1} \left\{ \begin{array}{lll} 1 & \text{für} & c = 0,2,4,6 \\ \sqrt{2} & \text{für} & c = 1,3,5,7 \end{array} \right\} \tag{2.3}$$

Fläche Die Fläche *(Area)* ist durch die Anzahl der in der Kontur enthaltenen Bildpunkte bestimmt und kann durch die Gauß'sche Flächenformel für Polygone berechnet werden. Die Kontur ist durch M Koordinatenpunkte $(x_0, x_1, ..., x_{M-1})$, wobei $x_i = (u_i, v_i)$, definiert:

$$Area(R) = \frac{1}{2} \cdot \left| \sum_{i=0}^{M-1} (u_i \cdot v_{[(i+1) \ mod \ M]} - u_{[(i+1) \ mod \ M]} \cdot v_i) \right| \tag{2.4}$$

Bounding Box Als Bounding Box wird das minimale, achsenparallele Rechteck bezeichnet, welches die Region vollständig umschließt. Sie berechnet sich durch die minimalen und maximalen Koordinatenwerte aller Punkte (u_i, v_i) in x- bzw. y-Richtung:

$$BoundingBox(R) = (u_{min}, u_{max}, v_{min}, v_{max}) \tag{2.5}$$

2.2 Java Media Framework

Bei der Java Media Framework API (JMF) handelt es sich um eine Programmierschnittstelle zur Handhabung von zeitbasierten Medien. Mit ihr können Java-Anwendungen und -Applets sehr einfach Audio- und Videodaten verarbeiten und auf Multimedia-Hardware wie Mikrofon, Lautsprecher und Videokamera zugreifen.

Die Version 1.0 wurde ursprünglich von Sun Microsystems, Intel und Silicon Graphics entwickelt, später erfolgte die Weiterentwicklung durch Sun und IBM. Wobei die Version 1 lediglich die Wiedergabe von Audio- und Videodaten zuließ, war es mit der Version 2 möglich, Audio und Video aufzunehmen, umzuwandeln und als Datenströme über Netze zu senden und zu empfangen. JMF 2.0 definiert zudem eine PlugIn-Schnittstelle, über die Software-Anbieter die Funktionalität von JMF erweitern and anpassen können. Die aktuelle Version 2.1.1e stammt aus dem Jahre 2003. Derzeit ist nicht klar, ob es eine Weiterentwicklung geben wird. [15]

Die nachfolgenden Abschnitte beschreiben weitere technische Details vom JMF: [9]

Unterstützte Plattformen

Die aktuelle Version JMF 2.1.1e unterstützt folgende Plattformen:

- Windows 95, 98, NT, 2000, XP, Vista
- Linux
- Solaris/Sparc

Die *Cross Platform Version* unterstützt sämtliche Plattformen, auf denen eine Java Virtual Machine läuft. Unter Apple OS und der dort laufenden Mac OS Runtime For Java 2.1.4 (MRJ) gibt es zurzeit noch Audio-Video-Synchronisationsprobleme, seitdem Apple dort einen Audio-Puffer von 6 Sekunden eingeführt hat.

Unterstützte Formate

JMF unterstützt von Hause aus diverse Audio- und Videoformate:

Audioformate

- Wave (.wav)
- AIFF (.aiff)
- GSM (.gsm)
- IBM HotMedia (.mvr)
- midi (.mid)
- MPEG 1 Layer 1, 2, 3 (.mp2, .mp3)
- Quicktime Audio (.mov)
- Sun Audio (.au)
- Wave (.wav)

Videoformate

- AVI (.avi)
- MPEG-1 Video (.mpg)
- Quicktime Video (.mov)

Architektur

Der Aufbau des Java Media Frameworks ähnelt der realen Medien-Architektur. Bekannte Geräte wie z.B. Videokamera, Videoband, Videorekorder, Fernseher und Lautsprecher haben ihre Entsprechungen im JMF. Abbildung 2.10 zeigt einen Aufbau der genannten Geräte. In den Klammern stehen die im JMF verwendeten Begriffe. Die *Data Source* beinhaltet den Medienstrom analog dem Videoband. Der *Player* steuert die Verarbeitung des Medienstroms analog dem Videorekorder. Zum Abspielen und zur Aufnahme mit dem Capture Device greift JMF auf die Ein- und Ausgabegeräte Videokamera und Lautsprecher zurück.

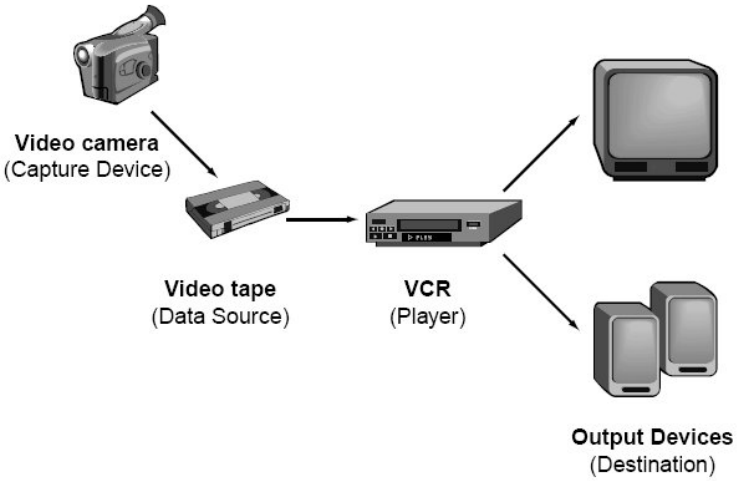


Abbildung 2.10: JMF Model [9]

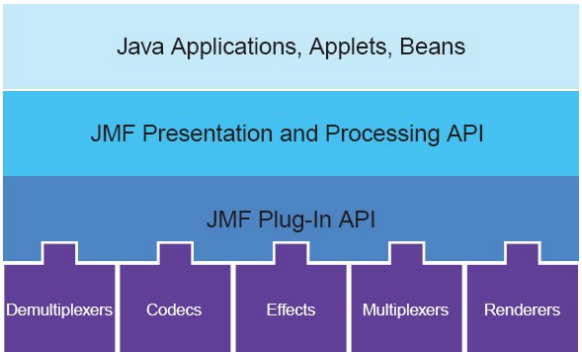


Abbildung 2.11: High Level JMF Architektur [9]

JMF-Manager

Die JMF-API besteht hauptsächlich aus Schnittstellen, die das Verhalten und die Interaktion von Objekten beschreiben. Implementierungen dieser Schnittstellen agieren innerhalb der Struktur des Frameworks. Durch die Verwendung von dazwischenliegenden Objekten, genannt *Manager*, sind weitere Implementierungen sehr einfach möglich. JMF definiert vier Manager:

- **Manager** - Behandelt den Aufbau von „Players“, „Processors“, „DataSources“ und „DataSinks“.
- **PackageManager** - Verwaltet die Registrierung von Paketen, welche JMF-Klassen enthalten, wie „Players“, „Processors“, „DataSources“ und „DataSinks“.
- **CaptureDeviceManager** - Verwaltet die Registrierung von vorhandenen Aufzeichnungsgeräten.
- **PlugInManager** - Verwaltet die Registrierung von vorhandenen PlugIn-Komponenten, wie Multiplexer, Demultiplexer, Codecs, Effekte und Renderer.

Um eine JMF-Anwendung zu schreiben, ist es lediglich notwendig, die **Manager create**-Methode aufzurufen, um ein „Player“-, „Processor“-, „DataSource“- und „DataSinks“-Konstrukt zu erstellen. Für die Aufzeichnung von einem Eingabegerät erhält man durch den **CaptureDeviceManager** eine Liste der verfügbaren Geräte.

2.3 OpenCV

OpenCV (Open Source Computer Vision) ist eine freie Bibliothek für Computer Vision-Anwendungen. OpenCV wurde erstmals im Jahre 2000 veröffentlicht, die Version 1.0 erschien aber erst im Jahre 2006. Die Bibliothek stellt unter anderem folgende Funktionen und Algorithmen zur Verfügung:

- Mensch-Maschine-Kommunikation und -Interaktion
- Objekterkennung
- Gesichtserkennung
- Gestenerkennung
- Motion Tracking
- Mobile Robotik
- Bildverarbeitung

Bei der Entwicklung von OpenCV wurde sehr großen Wert auf Effizienz und Geschwindigkeit gelegt. Gerade die Algorithmen im Bereich der Bildverarbeitung weisen einen geringen Overhead und eine hohe Performance auf. OpenCV ist kompatibel mit der Intel Image Processing Library (IPL) und benutzt intern das IPL-Format zur Bilddarstellung. Neben den Programmiersprachen C und C++ gibt es Wrapper für das .NET Framework, Python und Ch. [2],[12]

3 Analyse

In diesem Kapitel werden bereits vorhandene Verfahren und Technologien vorgestellt, die eine visuelle Erkennung von Körperteilen wie der Hand oder des Gesichts ermöglichen.

3.1 Einteilung der Verfahren

Die Abbildung 3.1 zeigt eine Einteilung der Verfahren.

Bei zweidimensionalen Verfahren werden Farbbilder oder Infrarotbilder über eine einzelne Foto- oder Videokamera aufgezeichnet und mit Hilfe der Bildverarbeitung (siehe 2.1) gefiltert, segmentiert und klassifiziert. Als Ergebnis erhält man die zweidimensionale Position und Größe des Körperteils. Bei der Hand kann nun eine weitere Verarbeitung anschließen, die beispielsweise die Finger zählt oder Gesten erkennt. Beim Gesicht können weitere Bestandteile wie Augen, Mund und Mimik erkannt und ausgewertet werden. Bei der Detektion der Hautfarbe unterscheidet man zwischen automatischen Verfahren und Verfahren, die eine einmalige Kalibrierung voraussetzen. In diesem Kapitel werden zwei automatische Verfahren vorgestellt, während die Kalibrierung im Entwurf behandelt wird.

Dreidimensionale Verfahren bestimmen zusätzlich die Tiefe, erfassen also die räumliche Position, Größe und Ausrichtung der Körperteile. Dadurch wird natürlich ein viel größerer Anwendungskreis erschlossen. So lassen sich die räumlichen Informationen auch zur Bewegung in einer dreidimensionalen virtuellen Umgebung verwenden.

3.2 Stereo Vision Systeme

Mittels eines Stereo-Systems, bestehend aus einer Stereokamera oder zwei einzelnen herkömmlichen Kameras, lassen sich dreidimensionale Informationen aus den zwei gleichzeitig aufgenommenen Bildern errechnen. Diese Informationen ermöglichen es, die dreidimensionale Position eines aufgenommenen Objektes, z.B. einer Hand, festzustellen.

In [5] wird ein solches System beschrieben. Der Finger eines Benutzers wird durch eine Stereokamera aufgezeichnet und ermöglicht ihm, z.B. eine Präsentation zu steuern. Der bloße Finger ersetzt den Laserpointer, Zeigestab bzw. die Computermaus. Das Block-Diagramm in Abbildung 3.2 beschreibt den groben Ablauf des Systems.

Mittels einer globalen Suche wird versucht, den Finger in den Stereo-Bildern zu detektieren. Es wird dabei davon ausgegangen, dass der Hintergrund weitaus dunkler als die Hand ist. Die Bilder werden per Schwellenwert (siehe 2.1.3) in Binärbilder umgewandelt. Zum Entfernen des Rauschens wird eine morphologische Schließung (siehe 2.1.2) durchgeführt. Der Finger wird durch die folgende Operation vom Handgelenk separiert:

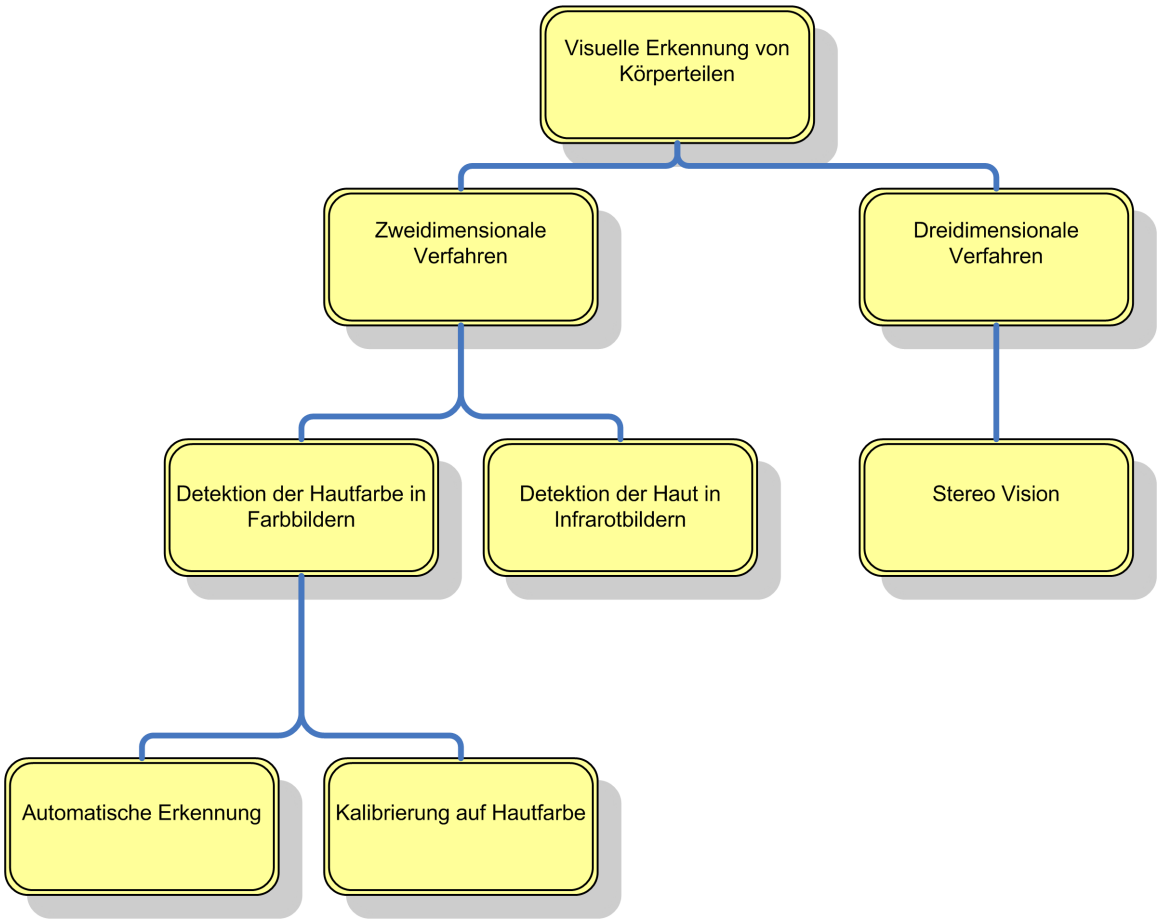


Abbildung 3.1: Einteilung der Verfahren zur visuellen Erkennung von Körperteilen

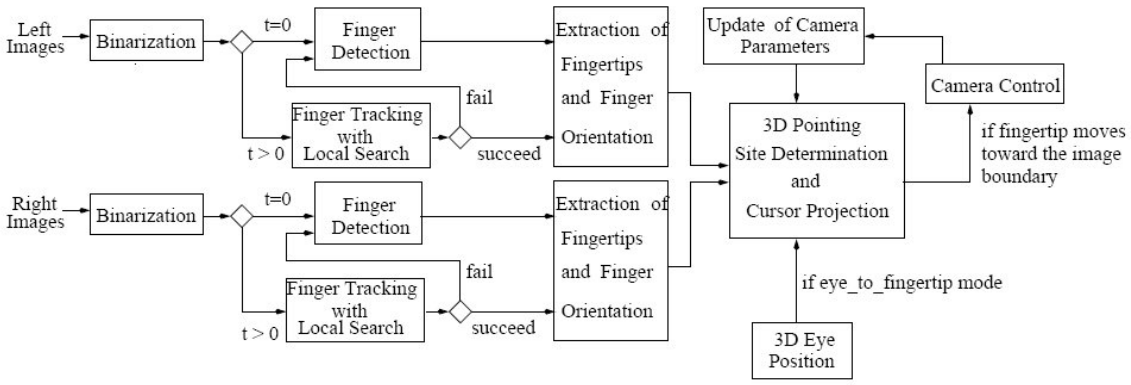


Abbildung 3.2: Block-Diagramm „free hand pointer“ [5]

$$I' = I - I \circ K \tag{3.1}$$

I' stellt dabei das Ergebnisbild, I das Originalbild und K das strukturierende Element für die morphologische Öffnung dar. Durch Berechnung der maximalen Entfernung aller Konturpunkte zum Handgelenk wird die Fingerspitze ermittelt. Die Zeigerichtung entspricht dem Vektor, der durch den Schwerpunkt des Fingers und der Fingerspitze gebildet wird. Sobald der Finger detektiert ist, wird die nächste Suche des Fingers auf ein Fenster beschränkt, welches über den Mittelpunkt des Fingers zentriert ist. Durch diese Verkleinerung des Suchraums wird die Berechnung dramatisch beschleunigt. Wird der Finger dennoch aus dem Fenster herausbewegt, wird wieder eine globale Suche im ganzen Bild durchgeführt.

Nachdem die zweidimensionalen Positionen der Fingerspitzen in beiden Bildern ermittelt worden sind, kann eine Umrechnung der Position und Zeigerichtung in den dreidimensionalen Raum mittels Triangulation erfolgen. In der Arbeit werden zwei Möglichkeiten aufgeführt, die Projektion der Zeigerichtung auf die Bildschirm- oder Projektionsfläche aus der Zeigerichtung zu bestimmen:

Finger-Orientation Mode Beim Finger-Orientation Mode werden mehrere Punkte auf der Achse des Fingers im linken Bild ausgewählt. Anschließend werden die korrespondierenden Punkte anhand Epipolarlinie im rechten Bild bestimmt. Die dreidimensionalen Positionen der Punkte werden errechnet und zu einer Linie im Raum verbunden.

Eye-to-Finger Mode Beim Eye-to-Finger Mode bewegt der Anwender die Fingerspitze zu einem Punkt zwischen seinem Auge und der Projektionsfläche. Die dreidimensionale Position der Fingerposition wird errechnet, und man erhält einen Richtungsvektor zwischen dem Auge und der Projektionsfläche. Mit dem Wissen über die durchschnittliche Entfernung zwischen Auge und Hand lässt sich die Position des Auges im Raum bestimmen. Während der Anwender seine Fingerspitze bewegt, errechnet das System laufend seine Position im Raum und daraus den Schnittpunkt aus dem Auge-zu-Finger-Vektor und der Projektionsfläche.

Leider ist die Auflösung beim Finger-Orientation Mode eher gering. Bei Rauschen im Videosignal wird die Berechnung der Fingerposition schnell ungenau. Dagegen arbeitet der Eye-to-Fingertip Mode zuverlässiger, da der Vektor zwischen Auge und Fingerspitze durch die größere Entfernung genauer ist als der Vektor durch die kurze Mittelachse des Fingers.

In der Arbeit wurden aktive Kameras verwendet, die der Bewegung des Anwenders folgen. Dies hat den Vorteil, dass der Bewegungsspielraum des Anwenders nicht durch den starren Fokus wie bei einer herkömmlichen Kamera eingeschränkt wird. Als Alternative zu einer aktiven Kamera könnte man auch eine Kamera mit einem großen Weitwinkel einsetzen. Dabei ist aber zu berücksichtigen, dass dadurch die erfassten Objekte kleiner aufgezeichnet werden und dadurch die Auflösung und Genauigkeit sinkt.

Die Kameras müssen vor ihrem Einsatz kalibriert werden, damit die Berechnungen möglichst genaue Ergebnisse liefern. In einem Testaufbau wurde dieses Ziel erreicht. Der „prediction error“ lag nur bei ca. einem Pixel, der „epipolar error“ nur bei 0,2 Pixel. Die

Abbildung 3.3 zeigt das System mit dem Eye-to-Fingertip Mode in Aktion. Im linken Teil des Bildes ist die Stereokamera zu erkennen. Der Anwender bewegt den Zeiger gerade unter das Wort „3D“.

Abbildung 3.3: Testaufbau „free hand pointer“ [5]

Die Abbildungen 3.4 bzw. 3.5 zeigen die Bewegungsabläufe im linken und rechten Bild (a) sowie die Ergebnisse (b) beim Schreiben des Buchstaben *W* mit dem Eye-to-Fingertip Mode bzw. des Buchstabens *Y* mit dem Finger-Orientation Mode. Gut zu erkennen ist die höhere Genauigkeit beim Buchstaben *W* (b). Dem entgegen steht jedoch der größere Platzbedarf beim Eye-to-Fingertip Mode, der jedoch durch die aktiven Kameras ausgeglichen werden kann.

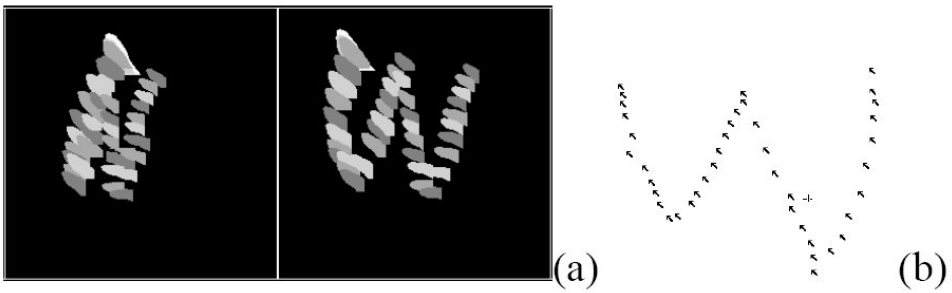


Abbildung 3.4: Ergebnis mit Eye-to-Fingertip Mode [5]

Das Fraunhofer Institut bietet mit dem *Stereo Hand Tracker* [6] ein bereits funktionierendes und lizensierbares System an. Die Handposition wird dabei ebenfalls durch eine Stereokamera dreidimensional erfasst. Je nach Kamera erfolgt die Positionsbestimmung bis zu 50 Mal in der Sekunde, womit ein flüssiges Arbeiten ermöglicht wird. Die Positionsabweichungen betragen dabei in x- und y-Richtung bis zu 5 mm, in der Tiefe bis zu 15 mm, wobei ein Bereich von 40 mal 50 cm bei einer Entfernung von 50 cm erfasst wird. Um den Bewegungsspielraum zu erweitern, können weitere Kameras in das System integriert werden. In Kombination mit einem 3D-Display (siehe Abbildung 3.6) ist es möglich, virtuelle Objekte im Raum zu berühren und zu verändern.

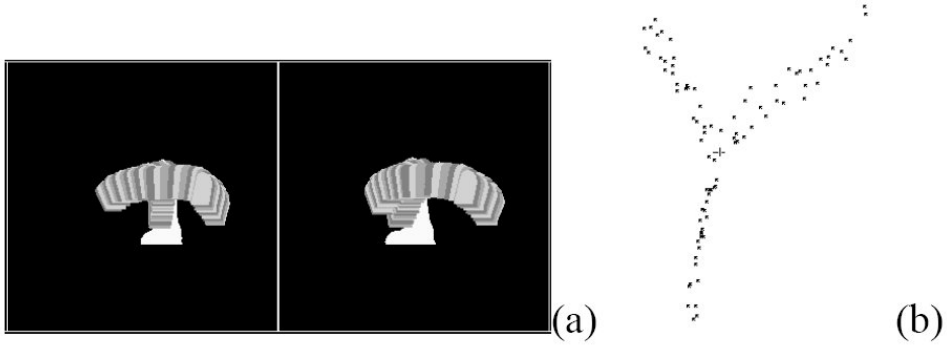


Abbildung 3.5: Ergebnis mit Finger-Orientation Mode [5]

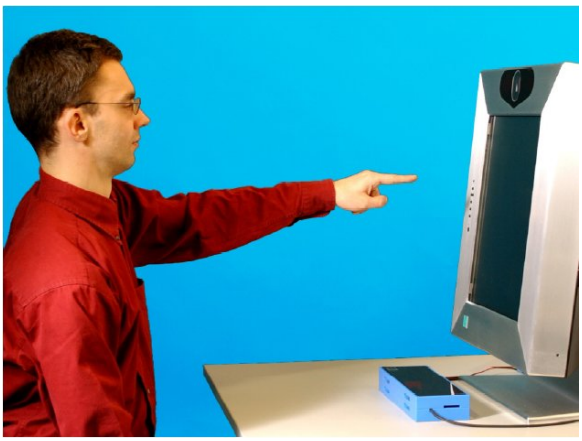


Abbildung 3.6: Stereo Hand Tracker des Fraunhofer Instituts [6]

3.3 Motion Segmentation

Motion Segmentation ist ein Teilgebiet der *Motion Detection*. Während Motion Detektion das Erkennen und Erfassen von Bewegung im Verhältnis zu einem als statisch betrachteten Hintergrund beschreibt, werden bei der Motion Segmentation bewegte Bildausschnitte erkannt, vom Hintergrund getrennt und schließlich in Klassen von Objekten eingeteilt.

Eine typische Anwendung der Motion Segmentation ist das Extrahieren von Objekten aus einem Videostream, beispielsweise bei der Videoüberwachung auf Flughäfen oder öffentlichen Plätzen. Sich bewegende Menschen und Fahrzeuge werden vom Hintergrund getrennt, um dann per *Motion Tracking* verfolgt zu werden. Schließlich hilft das *Scene Understanding* festzustellen, ob z.B. von den erkannten Personen und Fahrzeugen eine Gefahr ausgeht oder nicht.

In [8] werden zwei gängige Methoden der Motion Segmentation beschrieben:

3.3.1 Background Subtraction

Bei dieser Methode wird der Hintergrund ohne bewegte Vordergrundobjekte einmalig in einer Matrix gespeichert. Diese Matrix bildet das Referenzmodell, welches als Ausgangspunkt für die nachfolgende Videoverarbeitung dient. Jeder Frame[1] des fortlaufenden Videosignals wird Pixel für Pixel mit der Hintergrundmatrix verglichen. Der Unterschied zwischen den Farbwerten führt zu einem Differenzbild, aus dem sich dann auf die Objekte im Vordergrund schließen lässt. Durch kleinste Änderung im Umgebungslicht und das unvermeidbare Rauschen einer Videokamera werden zwangsweise aber immer Unterschiede auftreten, die fälschlich als Vordergrundobjekte interpretiert werden. Daher muss hier mit einem Schwellenwert (Threshold) gearbeitet werden. Erst wenn die absolute Differenz zwischen jedem Pixel des Hintergrundbildes und dem korrespondierenden Pixel des aktuellen Frames über diesem Schwellenwert liegt, wird ein Eintrag im Differenzbild vermerkt. Der Schwellenwert sollte so gewählt werden, dass das Rauschen herausgefiltert wird, die Objekte im Vordergrund jedoch erhalten bleiben.

Das Verfahren ist einfach zu implementieren und liefert bei hoher Verarbeitungsgeschwindigkeit gute Ergebnisse. Voraussetzung jedoch ist, dass der Hintergrund statisch bleibt und sich nicht bewegt.

3.3.2 Temporal Differencing

Diese Methode arbeitet ähnlich wie Background Subtraction, jedoch wird der aktuelle Frame nicht mit einem zu Beginn aufgenommenen Hintergrundbild verglichen, sondern jeweils mit dem vorangegangenen Frame. Hat sich ein Pixelwert von einem zum nächsten Frame verändert, lässt sich daraus schließen, dass dort eine Bewegung stattgefunden hat. Die Methode liefert also in erster Linie sich bewegende Objekte. Wenn der Hintergrund statisch bleibt, lässt sich aber durch dieses Verfahren auch der Vorder- vom Hintergund trennen.

[1] Einzelbild im kontinuierlichen Datenfluss einer Videokamera, Videofilms oder auch Computerspiels

Bei Objekten mit wenig Struktur, entstehen im Differenzbild jedoch Löcher, meist ist nur eine Kontur zu erkennen. Durch das Rauschen der Kamera muss ein Schwellenwert berücksichtigt werden, da ansonsten das Rauschen als Bewegung interpretiert werden würde. Der Schwellenwert muss mit Bedacht gewählt werden. Ist er zu klein, wird das Rauschen nicht herausgefiltert, ist er zu groß, werden geringe Bewegungen nicht erfasst oder entstehen größere Löcher im Differenzbild.

Der Vorteil vom Temporal Differencing gegenüber der Background Subtraction ist jedoch, dass kein Referenzmodell nötig ist. Die Verarbeitungsgeschwindigkeit der beiden Verfahren ist ähnlich hoch.

3.4 Bildbasierte Segmentierung

Bei der bildbasierten Segmentierung stehen als Eingangsmaterial im Gegensatz zur Motion Segmentation nicht eine Bildfolge, sondern nur einzelne isolierte Bilder zur Verfügung. Insofern gibt es keine Informationen über vorangegangene Bilder. Die folgenden Abschnitte zeigen Verfahren auf, die dennoch Körperteile wie Hände und Gesicht im Bild erkennen und vom Hintergrund trennen:

3.4.1 Detektion von Hautfarben in Farbbildern

Die menschliche Hautfarbe besteht hauptsächlich aus Rot- und Gelbtönen, besitzt wenig Struktur und bewegt sich in einem relativ gut abgegrenzten Bereich im HSB-Farbraum (vgl. 2.1.3), sofern man die Helligkeit (**B**rightness) außer Acht lässt. Der RGB-Farbraum kommt dabei nicht in Frage, weil er im Gegensatz zum HSB-Farbraum keine Farbtöne kennt und sich die Helligkeit nicht von den Farbkomponenten Rot, Grün und Blau trennen lässt. Die Abbildung 3.7 zeigt eine Punktwolke, die die Verteilung der Hautfarbe im HSB-Farbraum verdeutlicht. Es liegt also nahe, sich die menschliche Hautfarbe für die Segmentierung der Eingangsbilder zunutze zu machen. Man erhält nur die Bildbereiche, in denen sich Körperteile wie Hände und Gesicht befinden.

In [7] wird ein Verfahren zur Gesichtserkennung in Farbbildern in Form eines **Skin Filters** (Hautfilter) beschrieben. Ausgangspunkt ist ein RGB-Bild, dessen Farbwerte in den IRgBy-Farbraum umgewandelt werden:

$$I = \frac{L(R) + L(B) + L(G)}{3} \tag{3.2}$$

$$R_g = L(R) - L(G) \tag{3.3}$$

$$B_y = L(B) - \frac{L(G) + L(R)}{2} \tag{3.4}$$

$$wobei\ L(x) = 105 * log_{10}(x + 1) \tag{3.5}$$

Die entstehenden Matrizen R_g und B_y werden anschließend mit einem Medianfilter (siehe 2.1.1) versehen und in Farbton (*hue*) und Farbsättigung (*saturation*) umgerechnet:

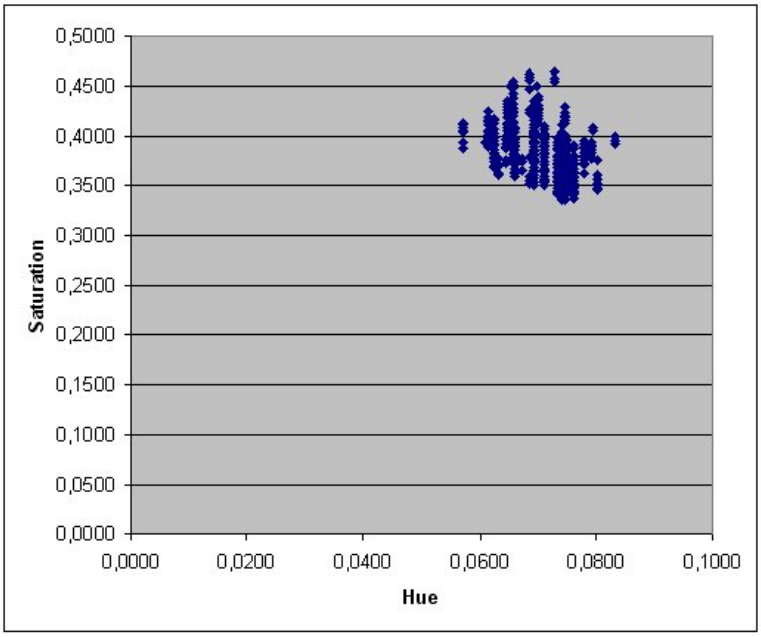


Abbildung 3.7: Hautfarbe im HSB-Farbraum

$$hue = atan^2(R_g, B_y) \tag{3.6}$$

$$saturation = \sqrt{R_g^2 + B_y^2} \tag{3.7}$$

Da die menschliche Haut eine relativ geringe Textur aufweist, bietet es sich an, dies ebenfalls beim Hautfilter zu berücksichtigen. Um Bereiche mit einer geringen Textur zu finden, wird eine *Texture Amplitute Map* erstellt. Dazu wird die Matrix I ebenfalls mit einem Medianfilter weichgezeichnet. Die Ergebnismatrix I' wird von I subtrahiert und nochmals weichgezeichnet. Mit Grenzwerten zu den drei Matrizen Farbton, Farbsättigung und Textur lassen sich nun hautfarbene Pixel herausfiltern und ein Binärbild erstellen. In [7] werden folgende Grenzwerte als praxistauglich angesehen:

$$120 < hue < 160, \quad 10 < saturation < 60, \quad texture < 4,5 \tag{3.8}$$

$$oder\ 150 < hue < 180, \quad 20 < saturation < 80, \quad texture < 4,5 \tag{3.9}$$

Die Abbildung 3.8 zeigt das Ergebnis des Hautfilters.

Der Autor des Artikels weist aber darauf hin, dass dem Hautfilter Grenzen gesetzt sind und Objekte ggf. fälschlicherweise als Haut identifiziert werden, obwohl sie keine Haut darstellen. Der Fehler lässt sich nicht durch eine andere Wahl der Grenzwerte beheben, er tritt auf, wenn größere Bereiche im Bild eine intensiv rote oder gelbe Farbe haben.

(a) Original (b) Binärbild nach Hautfilter

Abbildung 3.8: Anwendung des Hautfilters auf ein Farbbild [7]

In [14] wurde das Verfahren erweitert. Durch die Berücksichtigung des mittleren Farbtonwertes α und Sättigungswertes β können bessere und auf das jeweilige RGB-Bild abgestimmte Grenzwerte bestimmt werden. Die mittleren Werte berechnen sich wie folgt:

$$\alpha = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} H(i,j)}{n \times m} \tag{3.10}$$

$$\beta = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m} S(i,j)}{n \times m} \tag{3.11}$$

$H(i,j)$ bzw. $S(i,j)$ stellen dabei die in 2.1.3 ermittelten Farbton- bzw. Sättigungs-Matrizen dar. n und m sind die Dimensionen, also Breite und Höhe des RGB-Bildes. Durch die Verwendung der beiden Werte kann auch auf sich verändernde Beleuchtungssituationen reagiert werden. Eine weitere Verbesserung der Detektion der Hautfarben kann erreicht werden, wenn die mittleren Farbton- und Sättigungswerte für die bereits erkannten hautfarbenen Bereiche berechnet werden und das RGB-Bild mit entsprechend veränderten Grenzwerten ein zweites Mal gefiltert wird.

In [1] wird ein weiteres Verfahren zur automatischen Erkennung von Hautfarbe beschrieben. Ein wesentlicher Bestandteil stellt dabei das **Skin Color Model** (Hautfarbenmodell) dar. Dieses Modell nutzt im Gegensatz zum oben beschriebenen Verfahren nicht den HSB-, sondern den chromatischen Farbraum. Auch der chromatische Farbraum entfernt den Helligkeitsanteil aus der Farbe. Die Umrechnung vom RGB- in den chromatischen Farbraum erfolgt dabei für jeden Bildpunkt wie folgt:

$$r = \frac{R}{R + G + B} \tag{3.12}$$

$$b = \frac{B}{R+G+B} \tag{3.13}$$

Der Grünanteil berechnet sich durch $r+b+g=1$, ist demzufolge also redundant. Obwohl die Hautfarbe von Mensch zu Mensch teilweise stark differiert, liegt sie in einem kleinen, relativ gut abgegrenzten Bereich des chromatischen Farbraums. Die Ursache dafür ist, dass die Hautfarbe sich eher in der Helligkeit als im Farbton unterscheidet. Durch die Verwendung von 32.500 „Hautproben“ in 17 Farbbildern von Personen verschiedener Hautfarbe wurde das Hautfarbenmodell erstellt. Die Proben wurden zuvor mit einem Mittelwertfilter (siehe auch 2.1.1) geglättet, um das Rauschen zu minimieren.

Die Abbildung 3.9 zeigt die Verteilung der Hautfarbe im chromatischen Farbraum in einem dreidimensionalen Histogramm. Auf der waagerechten Fläche unten sind die chromatischen Farben Rot und Blau auf je einer Achse abgebildet. Je höher der senkrechte Wert desto öfter kam die Kombination aus Rot und Blau in den verwendeten Hautproben vor. In der Abbildung kann man klar den abgegrenzten Bereich erkennen, auf den sich die Hautfarbe im chromatischen Farbraum beschränkt.

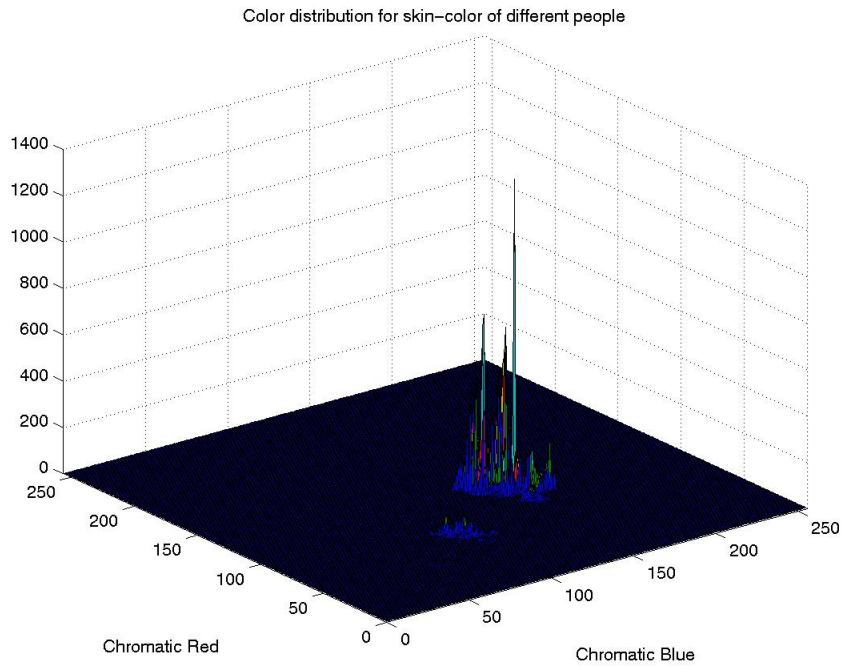


Abbildung 3.9: Verteilung der Hautfarbe im chromatischen Farbraum [1]

Das Histogramm kann nun dazu benutzt werden, um hautfarbene Regionen in weiteren Farbbildern zu erkennen. Sei $H(r,g)$ das Histogramm und n die Anzahl aller Histogrammeinträge, so berechnet sich die Wahrscheinlichkeit $P((r,g)|skin)$, dass ein bestimmter Farbwert (r,g) hautfarben ist, wie folgt:

$$P((r,g)|skin) = \frac{H(r,g)}{n} \tag{3.14}$$

In [1] wurde nun der Schritt vom speziellen Fall aufgrund von Beispielbildern zum allgemein formulierten Hautfarbenmodell gemacht, indem das Histogramm durch eine Gauß-Verteilung repräsentiert wird, die auf einer mathematischen Formel basiert. Die Wahrscheinlichkeit lässt sich nun unabhängig vom obigen Histogramm errechnen. Die Abbildung 3.10 zeigt diese Gauß-Verteilung.

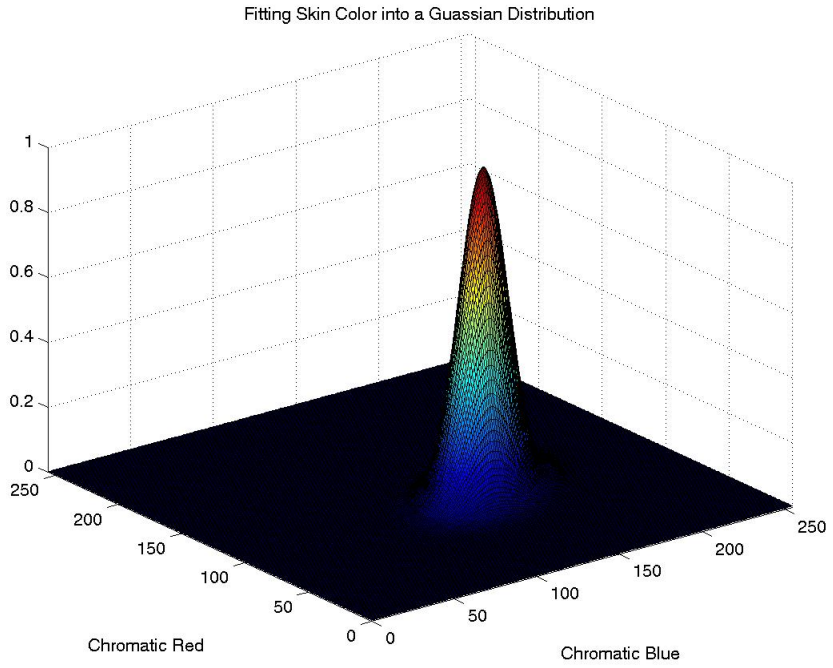


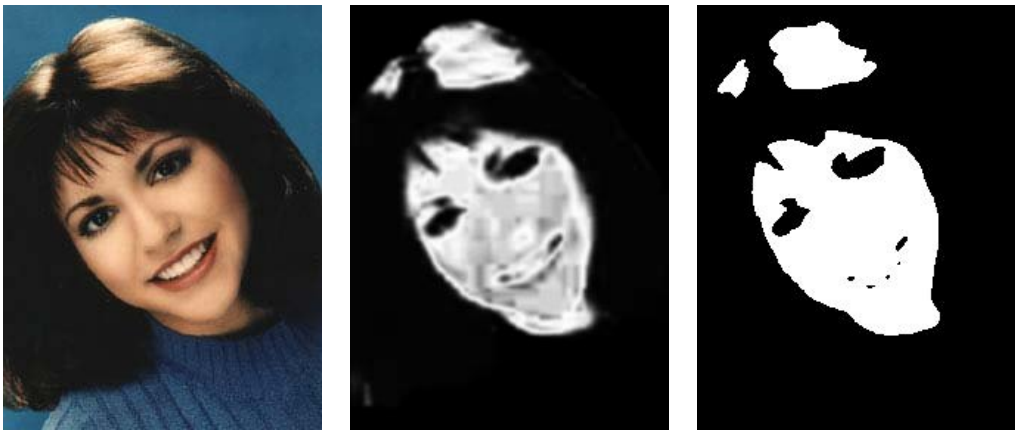
Abbildung 3.10: Repräsentation der Hautfarbe in Gauß-Verteilung [1]

Ein Farbbild kann durch die mathematische Repräsenatation in ein Graustufenbild umgewandelt werden, wobei die Helligkeit jedes Bildpunktes die Wahrscheinlichkeit angibt, dass es sich um einen hautfarbenen Bildpunkt handelt. Mit einem geeigneten Schwellenwertverfahren (siehe auch 2.1.3) kann das Graubild in ein Binärbild umgewandelt werden, wobei beispielsweise weiße Bereiche Hautfarbe anzeigen. Die Abbildung 3.11 zeigt ein Beispiel.

3.4.2 Detektion von Haut in Infrarotbildern

Neben der farbbasierten Detektion besteht auch die Möglichkeit, die infrarote Strahlung der menschlichen Haut für die Erkennung von Körperteilen zu verwenden. In [4] werden neben den Vorteilen von infrarotem Licht gegenüber normalem sichtbaren Licht auch ein Verfahren beschrieben, Gesichter mittels einer Infrarotkamera zu verfolgen.

Das meiste Licht im mittel- bis langwelligen infrarotem Bereich wird emittiert und nicht reflektiert. Das heißt, dass Infrarotlicht unabhängig vom Umgebungslicht ist. So macht es keinen Unterschied, ob die Aufnahmen tagsüber oder nachts gemacht werden oder die Beleuchtungsverhältnisse sich ständig ändern. Eine ausreichende Beleuchtung, ohne die eine farbbildbasierte Detektion nicht funktionieren würde, ist nicht notwendig. Auch spielen



(a) Original (b) Hautwahrscheinlichkeit (c) Hautsegmentierung

Abbildung 3.11: Verarbeitung eines Farbbildes mit dem Hautfarbenmodell [1]

verschiedene Hautfarben keine Rolle. Das Verfahren funktioniert unabhängig davon, ob die untersuchte Person nun hell-, rot-, gelb- oder dunkelhäutig ist.

Neben nackter Haut erkennt das Verfahren auch von Haaren oder Kleidung bedeckte Hautpartien, da infrarote Strahlung Haut und Kleidung durchdringen kann. Abbildung 3.12 zeigt die Segmentierung eines Bildes in Haut, bedeckte Haut und Hintergrund.

Ein weiterer Vorteil dieses Verfahrens ist, dass die Erkennungsgenauigkeit von Hautpartien bei Infrarotbildern sehr hoch ist. Da lediglich pixelorientierte Operationen durchgeführt werden, um festzustellen, ob ein Pixel im Eingangsbild zu einer Hautpartie gehört oder nicht, ist auch die Verarbeitungsgeschwindigkeit sehr hoch.

Wichtig für die einwandfreie Funktion des Systems ist die Kalibrierung der Infrarotkamera auf die Umgebungs- bzw. Raumtemperatur. Wird die Kamera z.B. von drinnen nach draußen bewegt, ist eine erneute Kalibrierung notwendig. Als Ergebnis stellten die Autoren des Verfahrens fest, dass zwischen einer manuellen und berechneten Gesichtsmitte lediglich eine Abweichung von zwei Pixeln auftrat.

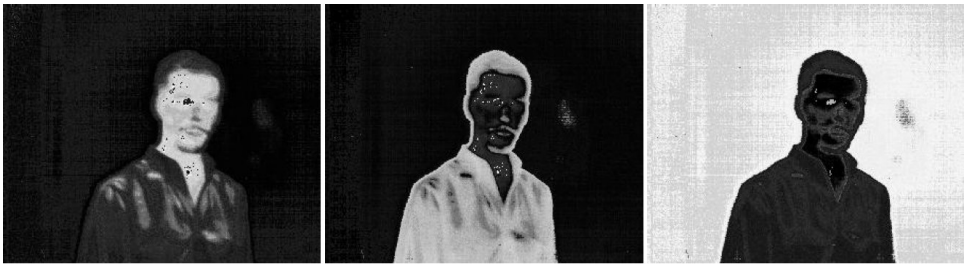


Abbildung 3.12: Segmentierung eines Infrarotbildes in Haut, bedeckte Haut und Hintergrund [4]

3.5 Vergleich der Verfahren

In diesem Kapitel wurden verschiedene Verfahren zur visuellen Erkennung von Körperteilen vorgestellt. Neben der zweidimensionalen Verarbeitung von Farb- und Infrarotbildern mit jeweils nur einer Kamera wurden auch Verfahren für die dreidimensionale Verarbeitung beschrieben. Insbesondere die Erkennung von Hautpartien in Farbbildern wurde ausführlich behandelt. Die Tabelle 3.1 stellt die Verfahren gegenüber. Ausgewählte Spalten der Tabelle werden im Folgenden näher erläutert.

3.5.1 Abhängigkeit vom Umgebungslicht

Optische Systeme, die mit sichtbarem Licht arbeiten, sind natürlich immer vom Umgebungslicht abhängig und benötigen eine ausreichende Beleuchtung. Dieses Kriterium beschreibt aber auch zusätzlich die Empfindlichkeit bei Änderung der Beleuchtung.

Das *Stereo Vision* Verfahren hat eine hohe Abhängigkeit vom Umgebungslicht, da davon ausgegangen wird, dass der Hintergrund weitaus dunkler als die Hand ist. Eine solche Beleuchtungssituation ist schwer herzustellen. Über den ähnlich arbeitenden *Stereo Hand Tracker* ist diesbezüglich nichts bekannt, die Abhängigkeit wird auf ein mittleres Maß geschätzt.

Beim *Background Subtraction* wird das aktuelle Videobild ständig vom anfänglich aufgenommenen Hintergrundbild subtrahiert. So stellt man fest, welche Bereiche sich im Videobild verändert haben und trennt Vordergrund von Hintergrund. Sollte sich die Beleuchtung im Hintergrund aber zu stark ändern, unterscheidet sich das aktuelle Videobild fast überall vom Hintergrundbild. Vorder- und Hintergrund lassen sich nicht mehr voneinander trennen. Das Verfahren versagt.

Im Gegensatz dazu kann sich beim *Temporal Differencing* die Beleuchtung ändern, ohne dass es zu Problemen beim Trennen von Vorder- und Hintergrund kommt. Da fortlaufend zwei aufeinanderfolgende Frames miteinander verglichen werden, reagiert das Verfahren sehr schnell auf Veränderungen.

Der *Hautfilter* und das *Hautfarbenmodell*, die beide eine Hauterkennung durchführen, haben eine mittlere Abhängigkeit. Dadurch, dass der Helligkeitsanteil in den erfassten Farbwerten keine Rolle spielt und nur Farbton und -sättigung verwendet werden, reagieren die Verfahren gut auf wechselnde Beleuchtung. Jedoch können zu starke Sonneneinstrahlung mit vielen Glanzpunkten oder harte Schlagschatten auf der Haut eine fehlerhafte Hauterkennung verursachen.

Das *Infrarotverfahren* erfasst die Hautpartien nicht über das sichtbare Licht, demnach spielt die Beleuchtung keine Rolle. Im Gegensatz dazu besteht eine große Abhängigkeit von der Raumtemperatur, die sich aber in Räumen weniger stark ändern sollte als die Lichtverhältnisse.

3.5.2 Bewegungsspielraum

Der Bewegungsspielraum gibt an, in welchem Bereich sich der Anwender aufhalten kann und von der Kamera erfasst wird.

Verfahren	Anwendung	Abhängigkeit von Umgebungslicht	Dimensionen	Hardware	Bewegungsspielraum	Echtzeitfähigkeit	Genauigkeit
Stereo Vision	Eingabegerät für Präsentationen	Hoch	3D	Stereokamera oder min. zwei herkömmliche Kameras	Groß (bei aktiven Kameras)	Ja	Sehr hoch bei ca. 1 Pixel Abweichung
Stereo Hand Tracker	3D-Eingabegerät in virtuellen Umgebungen	Mittel	3D	Stereokamera, 3D-Display	40 x 50 cm (erweiterbar durch zusätzliche Kameras)	Ja	hoch bei einer Abweichung von 3 mm in der Höhe und Breite bzw. 10 mm in der Tiefe
Background Subtraction	Bewegungserkennung	Hoch	2D	Eine herkömmliche (Video-) Kamera	Nur Kamerafokus	Ja	Mittel
Temporal Differencing	Bewegungserkennung	Niedrig	2D	Eine herkömmliche (Video-) Kamera	Nur Kamerafokus	Ja	Mittel
Hautfilter	Gesichtserkennung	Mittel	2D	Eine herkömmliche Kamera	Nur Kamerafokus	Bedingt	Mittel
Hautfarbenmodell	Gesichtserkennung	Mittel	2D	Eine herkömmliche Kamera	Nur Kamerafokus	Ja	Mittel
Infrarot	Gesichtserkennung	Keine	2D	Infrarotkamera	Nur Kamerafokus	Ja	Sehr gut bei ca. 2 Pixel Abweichung

Tabelle 3.1: Vergleich vorhandener Verfahren für die visuelle Erkennung von Körperteilen

Das *Stereo Vision System* setzt aktive Kameras ein, die der Bewegung des Anwenders folgen und so einen großen Bewegungsspielraum ermöglichen. Genaue Angaben liegen für den *Stereo Hand Tracker* vor. Sein Bewegungsspielraum hat eine Größe von 40 mal 50 cm und ist durch zusätzliche Kameras erweiterbar. Alle anderen Verfahren beschränken den Anwender auf den Sichtbereich der Kamera.

3.5.3 Echtzeitfähigkeit

Fast alle Verfahren können in einem Echtzeitsystem eingesetzt werden, d.h. sie sind schnell genug, auf die Bewegung des Anwenders in einer definierten Zeit und ohne spürbare Verzögerung zu reagieren. Die Ausnahme bildet der *Hautfilter*. Eine Testimplementierung ergab, dass der Hautfilter zwar für die Verarbeitung von Fotos geeignet ist, nicht jedoch für die Live-Verarbeitung eines Videostreams. Gegebenenfalls kann die aufwendige Verarbeitung auf einer Workstation[2] in Echtzeit geschehen. Auf einem handelsüblichen Computer ist das Verfahren zu langsam.

3.5.4 Genauigkeit

Die beiden *Stereo-Systeme* und das *Infrarot-Verfahren* haben laut Angaben der Autoren nur wenige Pixel bzw. Millimeter Abweichung, so dass mit einer hohen Erkennungsgenauigkeit gerechnet werden kann. Leider fehlen bei den restlichen Verfahren konkrete Angaben, die angegebenen Genauigkeiten sind daher geschätzt.

3.6 Anforderungen

Nachdem im vorherigen Abschnitt eine Ist-Analyse unter verschiedenen Gesichtspunkten vorgenommen wurde, folgt nun eine Soll-Analyse, die die verschiedenen Anforderungen an das Motion Tracking-System beschreibt. Dabei wird zwischen Muss-Kriterien und Kann-Kriterien unterschieden. Während die Umsetzung der Muss-Kriterien zwingend ist, ist ein Kann-Kriterium optional.

3.6.1 Funktionsweise

Muss-Kriterium

Die Funktionsweise des Motion Tracking-System muss letztendlich der eines Maustreibers entsprechen. Dieser ermöglicht es einer Anwendung, auf die Maus und ihren Status (Position und Maustasten) zuzugreifen, ohne sich mit der speziellen Hardware auseinandersetzen zu müssen. So kann die Maus ausgetauscht werden, ohne die Anwendung entsprechend anpassen zu müssen. Der Maustreiber stellt eine Schnittstelle zur Verfügung, über die die Anwendung mit der Maus kommuniziert.

[2] leistungsfähiger und zuverlässiger Rechner für die Ausführung von anspruchsvollen Anwendungen

Die Maus als Eingabegerät wird in diesem Fall durch die Webcam ersetzt. Das Motion Tracking-System muss durch Auswertung der Videosignale die Position des Mauszeigers und den Status der Maustasten ermitteln und über eine Schnittstelle der Anwendung zur Verfügung stellen. Alternativ zur Schnittstelle kann das System die ermittelten Daten auch an einen vorhandenen Maustreiber senden, der sie dann an die Anwendung weiterreicht.

3.6.2 Hardware

Muss-Kriterium

Seitens des Betreuers wurden in Bezug auf die zu verwendende Video-Kamera klare Vorgaben gemacht. Es darf in diesem System nur eine Video-Kamera verwendet werden. Das Verwenden mehrerer Kameras oder einer Stereokamera ist nicht möglich. Neben der Kamera dürfen keine Hilfsmittel, wie z.B. ein Handschuh oder spezielle an der Hand befestigte Sensoren, benutzt werden. Trotz dieser Einschränkungen muss das System die bloße Hand in dem zweidimensionalen Kamerabild einwandfrei erkennen können.

3.6.3 Benutzeroberfläche

Muss-Kriterium

Da das System für die Steuerung einer grafischen Benutzeroberfläche konzipiert werden soll, muss es selbst auch eine solche bereitstellen. Die Benutzeroberfläche soll zumindest folgende Bedienelemente enthalten:

- Einen Monitor, der das aktuelle Videobild der Webcam zeigt und mit dem der Anwender die korrekte Position, Ausrichtung und Einstellung der Webcam kontrollieren kann.
- Einen Schalter, mit dem die Positionssteuerung per Handbewegung gestartet und gestoppt werden kann.
- Einen Schieberegler zum Einstellen der Geschwindigkeit und Beschleunigung des Mauszeigers

Für diese drei Bedienelemente, wie auch für die gesamte Benutzeroberfläche, bestimmt die ISO-Norm *DIN EN ISO 9241-110* unter anderem folgende Grundsätze für die Dialoggestaltung:

Aufgabenangemessenheit

Muss-Kriterium

„Aufgabenangemessenheit bedeutet, dass die Benutzeroberfläche den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen“ [16].

Dies bedeutet für den oben genannten Schalter, dass dieser z.B. mit einem Tastaturkürzel verbunden sein sollte, der systemweit erreichbar ist. So lässt sich die Positionssteuerung alternativ per Mausklick auf eine Schaltfläche oder über ein Tastaturkürzel starten und

stoppen. Es nützt dem Anwender nichts, wenn sich bei einer Fehlfunktion die Positionssteuerung nicht wieder zurück auf die Computermaus übertragen lässt, weil der Schalter zum Stoppen nicht erreichbar ist.

Erwartungskonformität

Muss-Kriterium

In [16] ist Erwartungskonformität wie folgt definiert: „Ein Dialog ist erwartungskonform, wenn er konsistent ist und den Merkmalen des Benutzers entspricht...“

Auf die konkrete Anwendung übertragen heißt dies, dass bei Verwendung von Symbolen, Begriffen und Piktogrammen möglichst auf allgemein bekannte zurückgegriffen werden sollte. Am Beispiel der Schaltfläche zum Starten und Stoppen könnten die Symbole Dreieck und Viereck verwendet werden, so wie sie am CD-Spieler oder jeder Abspielsoftware zu finden sind.

Selbstbeschreibungsfähigkeit

Muss-Kriterium

„Ein Dialog ist selbstbeschreibungsfähig, wenn jeder einzelne Dialogschritt durch Rückmeldung des Dialogsystems unmittelbar verständlich ist oder dem Benutzer auf Anfrage erklärt wird.“ [16]

Nachdem der Anwender die Schaltfläche zum Starten betätigt hat, sollte ihm das System die durchgeführte Aktion optisch bestätigen. Dies kann z.B. in Form eines langsam blinkenden Dreiecks erfolgen, natürlich ist auch jede andere Form der Bestätigung denkbar.

Individualisierbarkeit

Muss-Kriterium

„Ein Dialog ist individualisierbar, wenn das Dialogsystem Anpassungen an die Erfordernisse der Arbeitsaufgabe sowie an die individuellen Fähigkeiten und Vorlieben des Benutzers zulässt.“ [16]

Als Beispiel sind hier die oben erwähnten Schieberegler zu nennen, mit denen man das System auf seine Bedürfnisse einstellen kann.

Fehlertoleranz

Muss-Kriterium

„Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“ [16]

Eine Benutzeroberfläche sollte Fehleingaben abfangen bzw. gar nicht erst zulassen. Durch die Verwendung von Schiebereglern statt eines freien Eingabefeldes ist eine Fehleingabe

ausgeschlossen. Würde der Anwender die Beschleunigung der Maus mit einem freien Eingabefeld bestimmen können, wären z.B. auch negative Eingaben möglich, ein Schieberegler hat dagegen einen definierten Bereich.

3.6.4 Geschwindigkeit

Muss-Kriterium

Eine große Rolle für die Akzeptanz des Systems spielt die Geschwindigkeit der Verarbeitung. Die Berechnung der Position aufgrund der Handbewegung und die Übertragung auf den Mauszeiger muss nahezu in Echtzeit geschehen. Eine merkliche Verzögerung, d.h. ein „Nachziehen“ des Mauszeigers, macht das System langsam bis unbedienbar. Die Anwender sind die direkte Umsetzung ihrer Bewegung durch die Computermaus gewöhnt und würden eine Verzögerung nicht akzeptieren. Neben dem Tracken der Handposition ist auch das zeitnahe Erkennen einer Geste für den Mausklick wichtig. Wird die Geste zu spät erkannt, hat der Anwender die Position ggf. schon verändert, und der Mausklick erfolgt nicht bzw. an einer verkehrten Stelle. Bei einer Framerate der Webcam von 30 Bildern pro Sekunde darf die Verarbeitung somit maximal $\frac{1000}{30} = 33$ Millisekunden nicht überschreiten.

3.6.5 Genauigkeit

Muss-Kriterium

Heutige Computermonitore mit ihren hohen Bildschirmauflösungen haben sehr kleine Kontrollelemente, die es zu treffen gilt. Um z.B. die Größe eines Fensters zu verändern, muss man die Maus zu dem nur wenige Pixel großen Fensterrahmen bewegen. Die Positionierung des Mauszeigers muss daher pixelgenau stattfinden können.

3.6.6 Robustheit

Muss-Kriterium

Ein optisches System wie das Tracking System ist ständig Störfaktoren ausgesetzt. Während eine optische Computermaus einen relativ gleichmäßigen Hintergrund (Mauspad) unter konstanten Lichtbedingungen (die einzige Lichtquelle befindet sich in der Maus) vorfindet, kann sich die Umgebung für die Webcam ändern. Gering wechselnde Beleuchtungsverhältnisse müssen vom System herausgefiltert werden können.

Kann-Kriterium

Weitere Störfaktoren sind z.B. im Hintergrund durchs Videobild laufende Personen, komplexe und bewegte Hintergründe oder andere Körperteile außer der Hand im Videobild. Auch mit diesen Störfaktoren muss die Erkennung der Handbewegung einwandfrei funktionieren.

Kann-Kriterium

Dadurch, dass die Computermaus durch einen Systemtreiber auf systemnaher Ebene angesprochen wird, ist sie unempfindlich gegen Leistungsschwankungen des Computers. Leistungschwankungen enstehen beispielsweise durch das Starten von speicher- oder rechenintensiven Anwendungen. Auch wenn der Computer vollkommen ausgelastet ist, kann die Computermaus noch zuverlässig bewegt werden. Mausklicks werden gepuffert und - wenn auch verzögert - ausgeführt. Dieselben Eigenschaften sollte auch das Tracking System aufweisen.

3.6.7 Plattformunabhängigkeit

Muss-Kriterium

Das System soll unter den derzeit gebräuchlichsten Betriebssystemen Windows, Linux und Apple OS gleichermaßen zum Einsatz kommen können. Dabei soll jede mit einer Maus bedienbare Anwendung unterstützt werden.

4 Entwurf

Nachdem beschrieben wurde, wie sich das System in das Betriebssystem integriert und die Rahmenbedingungen feststehen, werden in diesem Kapitel die einzelnen Schritte von der Aufnahme des Videobildes bis zur Steuerung des Mauszeigers entworfen. Dabei wird detailliert auf die Kalibrierung des Systems auf die Hautfarbe des Anwenders, die anschließende Bildverarbeitung, die Erkennung der Fingerspitzen und die Errechnung der Bildschirmposition des Mauszeigers eingegangen.

4.1 Systemintegration

Wie in der Abbildung 4.1 ersichtlich, fügt sich das Motion Tracking-System in einer weiteren Ebene in das Betriebssystem ein. Es sitzt zwischen Anwendungsebene und Gerätetreiber und steuert eine Anwendung indirekt. Bei der Bedienung einer Anwendung mit einer Computermaus führt die Anwendung in regelmäßigen Abständen eine Statusabfrage über den Maustreiber aus, der wiederum mit der Hardware, also der Maus, verbunden ist. Ähnlich verläuft die Kommunikation des Motion Tracking-Systems mit der Webcam. Über den Treiber der Webcam erhält das System das laufende Videobild, errechnet daraus die Position des Fingers, erkennt einen Mausklick und gibt diese Informationen an den Maustreiber weiter.

Alternativ wäre eine direkte Koppelung von Anwendung und Motion Tracking-System denkbar, indem das System eine Schnittstelle ähnlich dem Maustreiber zur Verfügung stellt. Dies hätte aber den Nachteil, dass jede Anwendung diese Schnittstelle unterstützen müsste. Dies entfällt beim Umweg über den Maustreiber, den standardmäßig jede Anwendung ansprechen kann.

4.2 Rahmenbedingungen

Für eine einwandfreie Funktion des Systems müssen einige Rahmenbedingungen eingehalten werden. Die verwendete Videokamera muss bestimmte Voraussetzungen erfüllen und an der richtigen Stelle positioniert werden. Neben einer ausreichenden Beleuchtung sollte der Anwender auch auf den Hintergrund und geeignete Kleidung achten. Auch die korrekte Ausrichtung der Hand spielt eine wichtige Rolle für die Fingerspitzenerkennung.

4.2.1 Auswahl der Kamera

Bei der verwendeten Videokamera kann es sich um eine einfache Webcam handeln, wie sie relativ preiswert in jedem Kaufhaus zu finden ist. Webcams im untersten Preissegment

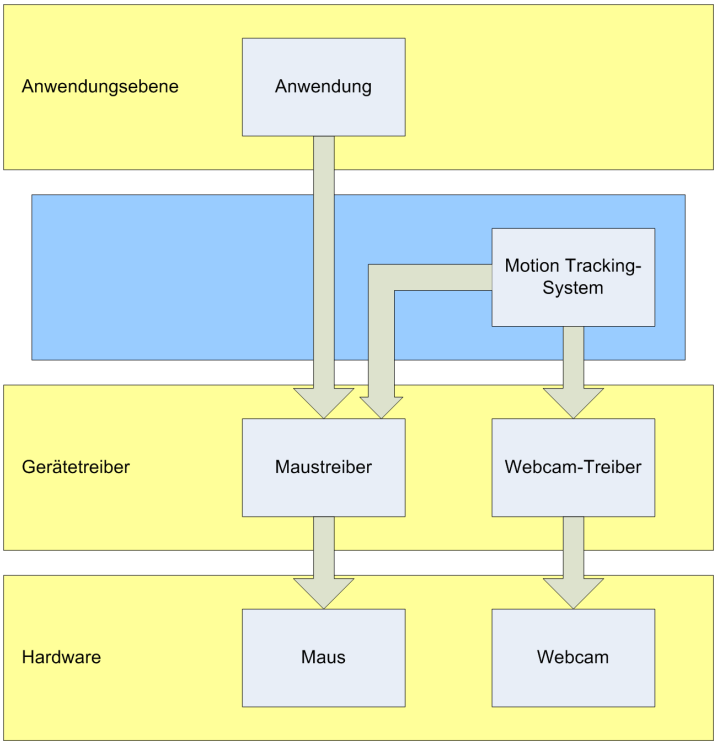


Abbildung 4.1: Integration des Motion Tracking-Systems in das Betriebssystem

haben leider aber oft nur eine mäßige Bildqualität und weisen bei ungünstigen Lichtverhältnissen ein starkes Rauschen auf. Von ihrem Einsatz wird daher abgeraten.

Die Standardauflösung heutiger Webcams liegt bei 320 mal 240 Bildpunkten bei 30 Bildern pro Sekunde. Diese Auflösung und Framerate ist für das System ausreichend. Sofern die Webcam mit einer höheren Auflösung z.B. von 640 mal 480 Bildpunkten betrieben wird, muss beachtet werden, dass dadurch unter Umständen die Verarbeitungsgeschwindigkeit des Systems durch die erhöhte Datenmenge herabgesetzt wird.

Die meisten Kameras passen Helligkeit, Kontrast und Weißabgleich automatisch an. Nicht immer finden sie jedoch die optimalen Einstellungen. Daher muss es möglich sein, dass der Anwender diese Einstellungen auch manuell vornehmen kann.

4.2.2 Positionierung der Kamera

Im Handel werden hauptsächlich Webcams angeboten, die fest im Gerät eingebaut sind oder extern per USB (Universal Serial Bus) angeschlossen werden. Fest eingebaute Kameras in Laptops oder Monitoren sind meist oberhalb des Bildschirms positioniert. Sie richten ihren Fokus auf Oberkörper und Kopf des Anwenders, da sie für Videochats konzipiert sind. Externe Webcams werden mit einer Klemme ebenfalls oben am Bildschirm befestigt und/oder besitzen einen Standfuß zur freien Positionierung.

Da die Hand vom System durch ihre Hautfarbe erkannt wird, sollte der Fokus der Kamera auf die Hand gerichtet sein und nicht auch andere Körperteile wie Gesicht oder Arme

erfassen. Daher sind fest eingebaute Webcams eher ungeeignet, es sei denn, sie lassen sich ausreichend weit schwenken. Externe Webcams sollten links oder rechts neben dem Monitor platziert und auf die seitlich vom Körper gehaltene Hand gerichtet sein. Eine erhöhte Position der Webcam verhindert, dass Schreibtischutensilien mit aufgezeichnet werden.

Natürlich wäre auch denkbar, die Kamera über oder unter der Hand zu positionieren. Eine Aufnahme von oben hätte den Vorteil, dass sich unter der Hand nur die Schreibtischunterlage befinden würde. So könnte leicht für einen homogenen und kontrastreichen Hintergrund gesorgt werden, was die Arbeit des Hautfilters vereinfacht (auf den Hautfilter wird später in diesem Kapitel unter 4.4.3 detailliert eingegangen). Dies ist bei einer Positionierung der Kamera vor der Hand schwieriger, da im Hintergrund oft Möbel und andere Einrichtungsgegenstände stehen, die sich nicht so leicht entfernen lassen. Würde die Kamera von unten auf die Hand gerichtet sein, wird als Hintergrund die Decke aufgenommen. Eine Deckenbeleuchtung könnte hier zu unerwünschten Blendungen der Kamera führen. Im Gegensatz führt z.B. eine weiße Deckenfarbe zu sehr guten Ergebnissen beim Hautfilter. Wie oben bereits erwähnt, lassen sich aber gängige Webcams mit den mitgelieferten Befestigungen weder über noch unter der Hand anbringen.

Die Abbildung 4.2 zeigt eine geeignete Positionierung der Kamera.

Abbildung 4.2: Positionierung der Kamera

4.2.3 Beleuchtung

Ohne eine ausreichende Beleuchtung wird kein optisches Aufnahmegerät gute Ergebnisse erzielen können. Daher spielt sie auch bei diesem System eine sehr wichtige Rolle. Di-

rekte Sonneneinstrahlung ist aber zu vermeiden, da es hierbei zu Überstrahlungen und zu vielen Glanzpunkten kommen kann. Diese lassen z.B. Teile der Hand im Kamerabild als weiß erscheinen, so dass die Hautfarbe nicht mehr als Erkennungsmerkmal ausreicht. Die Abbildung 4.3 zeigt eine solche Überstrahlung. Bei einem Pixel wurde mit Hilfe einer Bildverarbeitungssoftware die Farbe festgestellt. Es handelte sich nicht etwa um einen sehr hellen Hautton, sondern um reines Weiß. Dasselbe gilt bei direktem Licht aus künstlichen Lichtquellen. Gute Ergebnisse lassen sich daher mit indirektem künstlichen Licht erzielen, während man zu starkes Tageslicht durch z.B. Jalousien abschwächt. Der Abschnitt 6.2 im Kapitel Literaturtest zeigt die Auswirkungen von verschiedenen Beleuchtungssituationen auf die Erkennung der Handkontur.

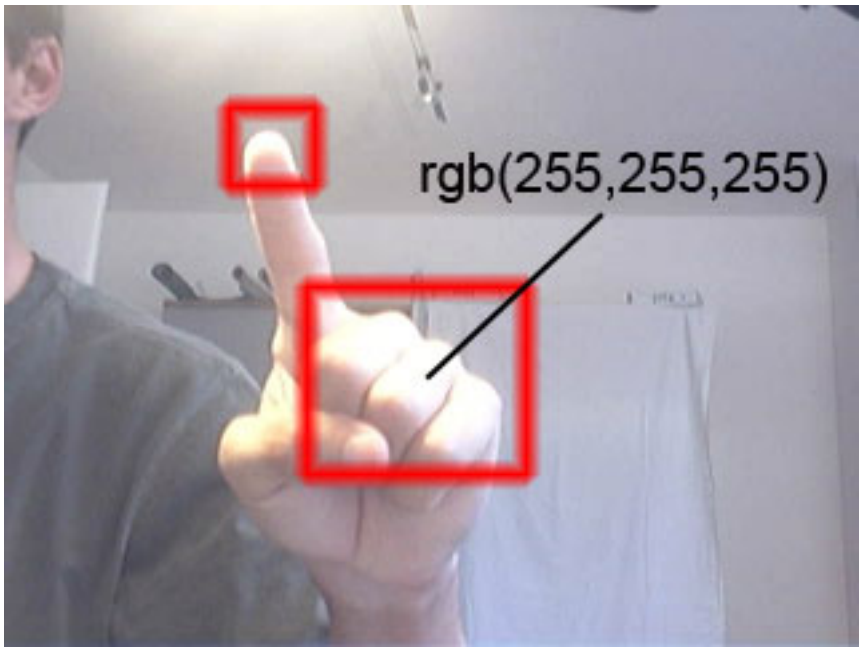


Abbildung 4.3: Überstrahlung durch direktes Licht

4.2.4 Kleidung und Hintergrund

Für eine einwandfreie Erkennung der Hautfarbe muss auf die richtige Kleidung geachtet werden. Zunächst sollte der Anwender ein langärmeliges Kleidungsstück tragen, damit nur die Hautpartien der Hand und nicht des Armes vom Hautfilter erfasst werden. Auch die Farbe der Kleidung spielt eine wichtige Rolle. Verständlicherweise sind hautfarbene oder transparente Kleidungsstücke denkbar ungeeignet. Auch bei Kleidung in Rot- und Gelbtönen kann es passieren, dass sie als Haut identifiziert wird und sozusagen mit der Hand verschmilzt.

Gleiches gilt für den Hintergrund, d.h. der Teil des Raumes, der neben dem Anwender mit von der Kamera aufgenommen wird. Gerade Holzmöbel ähneln sehr stark der menschlichen Hautfarbe. Auch scheinbar weiße Wände können einen gelblichen Ton haben, der sich ebenso in der Hautfarbe wiederfindet.

4.2.5 Ausrichtung der Hand

Eine optimale Ausrichtung der Hand zur Kamera ist dann gegeben, wenn die benötigten Merkmale der Hand klar und eindeutig durch das System erfasst werden können. So müssen z.B. für eine Gestenerkennung alle Finger erkennbar sein. Dies ist dann der Fall, wenn die Hand aufrecht und in einer Ebene parallel zur Kamera bewegt wird. Wird die Hand schräg gehalten, verzerrt sich das Kamerabild und Finger erscheinen kürzer, als sie es in Wirklichkeit sind. Die Abbildung 4.4(b) zeigt einen solchen Fall, bei dem die einzelnen Finger aber durchaus erkannt werden können. In Abbildung 4.4(b) jedoch erfasst die Kamera vom Zeigefinger lediglich die Fingerspitze, die sozusagen den Rest des Fingers verdeckt. Während das menschliche Gehirn ohne Zweifel einen Zeigefinger von vorne sieht, hat der Computer in dieser zweidimensionalen Darstellung Probleme, einen gestreckten Finger zu erkennen. Wird wie in Abbildung 4.4(c) die Hand aufrecht vor die Kamera gehalten, so kann über die Kontur der Zeigefinger erfasst werden. Der Abschnitt 4.5 beschäftigt sich ausführlich mit diesem Thema.

(a) Handebene parallel zur Kamera

(b) Schräge Haltung

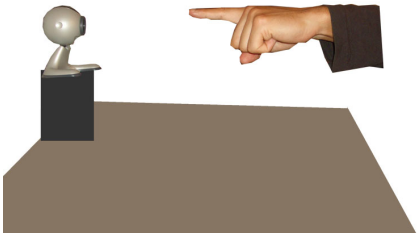
(c) Handebene parallel zur Kamera

(d) Schräge Haltung

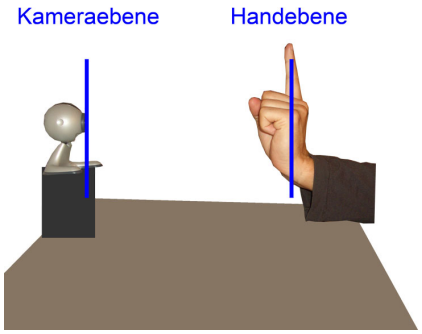
Abbildung 4.4: Ausrichtung der Hand

Die natürlichen Zeigegesten des Armes und der Hand zeichnen sich dadurch aus, dass der gestreckte Zeigefinger eine Linie mit dem Unterarm bildet. Damit die Hand aber aufrecht gehalten werden kann, muss je nach Positionierung der Kamera das Handgelenk nach oben gestreckt werden. Je niedriger die Kamera positioniert ist, desto stärker fällt die

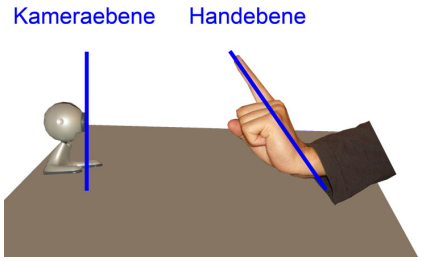
Streckung des Handgelenkes aus und desto geringer ist der Bewegungsspielraum. Abbildung 4.5(a) zeigt die natürliche Zeigegeste, bei der jedoch im Kamerabild der Zeigefinger vom Computer schwer zu erkennen ist. In der Abbildung 4.5(b) muss der Anwender zwar sein Handgelenk nach oben strecken, dafür erscheint die Hand so im Kamerabild, dass eine Fingererkennung gute Ergebnisse liefert. Die erhöhte Position der Kamera gewährt dem Anwender einen ausreichenden Bewegungsspielraum. Dagegen muss in Abbildung 4.5(c) der Anwender wegen der tiefen Kameraposition sein Handgelenk nahezu auf dem Tisch ablegen. Aber auch in dieser Haltung ist es nicht möglich, das Handgelenk weit genug nach oben zu strecken, damit die Hand parallel zur Kamera ist. Zudem schränkt diese Haltung den Bewegungsspielraum zu sehr ein.



(a) Natürliche Zeigegeste



(b) Kamera- und Handebene sind parallel



(c) Zu tiefe Kameraposition

Abbildung 4.5: Beugung des Handgelenks

4.3 Verarbeitungskette

Die Abbildung 4.6 zeigt ein Ablaufdiagramm, welches die gesamte Verarbeitungskette des Systems vom Einlesen eines Videoframes bis zur Bewegung des Mauszeigers bzw. der Erkennung des Mausklicks zeigt. Dieser Abschnitt soll einen Überblick über die einzelnen Schritte geben. Einfache, überschaubare Verfahren werden komplett beschrieben. Umfangreichere Verfahren werden kurz zusammengefasst, Details aber in einem separaten Abschnitt behandelt.

Abbildung 4.6: Ablaufdiagramm

Video-Frame Als erstes wird aus dem fortlaufenden Videosignal der Videokamera ein einzelner Frame entnommen. Dieser Vorgang wird auch als Grabben[1] bezeichnet und geschieht z.B. 30 Mal in der Sekunde bei einer Auflösung von 320 mal 240 Bildpunkten. Die erhaltenen Rohdaten werden meist in Form einer Liste von Farbwerten geliefert, die gegebenenfalls noch in die gewünschte Form, in der Regel ein RGB-Bild, konvertiert werden muss.

Unterabtastung Für die folgenden Verarbeitungsschritte reicht eine geringere Bildauflösung aus. Die Auflösung wird z.B. auf die Hälfte, also 160 mal 120 Bildpunkte reduziert. Durch die geringere Datenmenge wird das System beschleunigt.

Hautfilter Der Hautfilter wandelt das Farbbild in ein Binärbild um. Weiße Bereiche kennzeichnen dabei den Hintergrund, schwarze Bereiche den Vordergrund, d.h. die erkannten Hautpartien. Der Hautfilter wird durch die einmalig beim Start durchzuführende Kalibrierung erstellt. Eine genaue Beschreibung der Kalibrierung ist unter 4.4 zu finden.

Konturfindung Mit dem Hautfilter werden zwar die hautfarbenen Bereiche markiert, jedoch noch nicht in Regionen unterteilt. Dazu muss herausgefunden werden, welche Pixel zu welcher Region gehören, wieviele Regionen existieren und wo diese sich im Bild befindet. Dazu bietet sich das Verfahren der kombinierten Regionenmarkierung und Konturfindung an, welches bereits unter 2.1.4 behandelt wurde. Dieses Verfahren liefert eine Liste von allen gefundenen Konturen in Form von Kettencodes oder Konturpunkten.

Wahl der größten Kontur Es wird davon ausgegangen, dass es sich bei dem größten hautfarbenen Objekt um die Hand handelt. Anhand des Kettencodes einer Kontur können eine Reihe von mathematischen Eigenschaften berechnet werden. Eine davon liefert die Fläche der Kontur in Pixeln (siehe 2.1.5). Die Kontur mit der größten Fläche wird ausgewählt.

Fingerspitzenerkennung Um die Positionen der Fingerspitzen im Bild zu ermitteln, wird zunächst der Mittelpunkt der Handfläche festgelegt. Von diesem Punkt aus wird die Entfernung zu jedem Konturpunkt berechnet. Bei den Punkten mit den größten Entfernungen handelt es sich mit hoher Wahrscheinlichkeit um die Fingerspitzen. Im Abschnitt 4.5 wird das Verfahren detailliert vorgestellt.

Wahl des längsten Fingers Die Wahl des längsten Fingers gestaltet sich nach der Fingerspitzenerkennung sehr einfach. Die Positionen der Fingerspitzen im Bild stehen fest, ebenso der Mittelpunkt der Handfläche. Die Fingerspitze mit der größten Entfernung zum Mittelpunkt verweist auf den längsten Finger.

[1] engl.: *to grab* aufgreifen, packen

Detail-Fingerspitzenerkennung Da die Auflösung der Bildes zu Anfang reduziert wurde, ist die Position des längsten Fingers ungenau. Dadurch wird die Fingerspitzenerkennung in einem kleinen Bereich rund um die erkannte Fingerspitze noch einmal in der ursprünglichen hohen Auflösung durchgeführt. Im Abschnitt 4.6 wird das Verfahren detailliert vorgestellt.

Bestimmung der Cursor-Position Die Position der Fingerspitze im Bild wird auf die Auflösung des Bildschirmes skaliert. Der Mauszeiger wird dann entsprechend positioniert. Details werden im Abschnitt 4.7 beschrieben.

Zählen der Finger Die Anzahl der potentiellen Finger liefert die Fingerspitzenerkennung. Damit nicht auch halb ausgestreckte Finger oder Bildfehler als Finger erkannt werden, wird ein Schwellenwert benutzt. Erst wenn die Fingerlänge über diesem Schwellenwert liegt, wird der Finger gezählt.

Erkennung Mausklick Ein Mausklick mit der linken Maustaste wird durch das kurze Abspreizen eines zweiten Fingers signalisiert. Dabei wechselt die Anzahl der Finger kurz von einem auf zwei und dann wieder auf einen Finger. Wird eine solche Sequenz erkannt, wird der Mausklick an der aktuellen Cursor-Position ausgeführt. Details dieses Verfahrens werden im Abschnitt 4.8 beschrieben.

Erkennung komplexer Handgesten Neben der Steuerung des Cursors und dem Erkennen eines Mausklicks lassen sich mit dem System auch weitere Handgesten erkennen. Dazu werden zwei Finger gestreckt und die Hand ruckartig nach links, rechts, oben oder unten bewegt. Die Kombination mehrerer Bewegungen ermöglicht komplexe Handgesten. Details werden im Abschnitt 4.9 beschrieben.

4.4 Kalibrierung

Ein relativ einfaches aber sehr effektives Verfahren zur Segmentierung von Hautfarbe ist die Kalibrierung des Systems auf die Hautfarbe des Anwenders. Das Verfahren muss einmalig beim Start des Systems durchgeführt werden und verläuft zweistufig:

4.4.1 Aufnahme der Hautfarbe

Im ersten Schritt wird die Hautfarbe (Vordergrund) aufgenommen. Dazu wird der Anwender aufgefordert, seine Hand kurz in die Kamera zu halten. Dem Anwender wird ein Live-Bild der Kamera gezeigt, in das zusätzlich eine Markierung eingeblendet wird (siehe Abbildung 4.7). Für die Kalibrierung auf die Hautfarbe der Hand bietet sich eine kreisförmige Markierung an; sie entspricht in etwa der Form einer Faust. Der Anwender hat darauf zu achten, dass der eingeblendete Kreis vollständig von seiner Faust bedeckt wird, so dass nur seine Hautfarbe und keine Farben aus der Umgebung mit aufgezeichnet werden. Dabei sollte die Markierung möglichst viel von der Faust umschließen, damit alle Hauttöne und -Schattierungen erfasst werden.

4.4.2 Aufnahme des Hintergrundes

Es reicht nicht aus, nur die Hautfarbe aufzuzeichnen. Kommen Hautfarben im Hintergrund vor, so würde der Hautfilter sie als Körperteil klassifizieren. Rötliche und gelbe Objekte wie z.B. Holzmöbel und Holzböden bewegen sich im selben Farbbereich wie die Haut. Daher wird im zweiten Schritt der Hintergrund aufgenommen. Dazu muss der Anwender das Kamerabild kurz verlassen, so dass keine nackten Körperteile mehr aufgezeichnet werden. Zur Kontrolle wird ihm wie bei der Aufnahme des Vordergrundes ein Live-Bild der Kamera angezeigt. Der Anwender betätigt dann eine Schaltfläche oder eine Taste, und das aktuelle Videobild wird gespeichert.

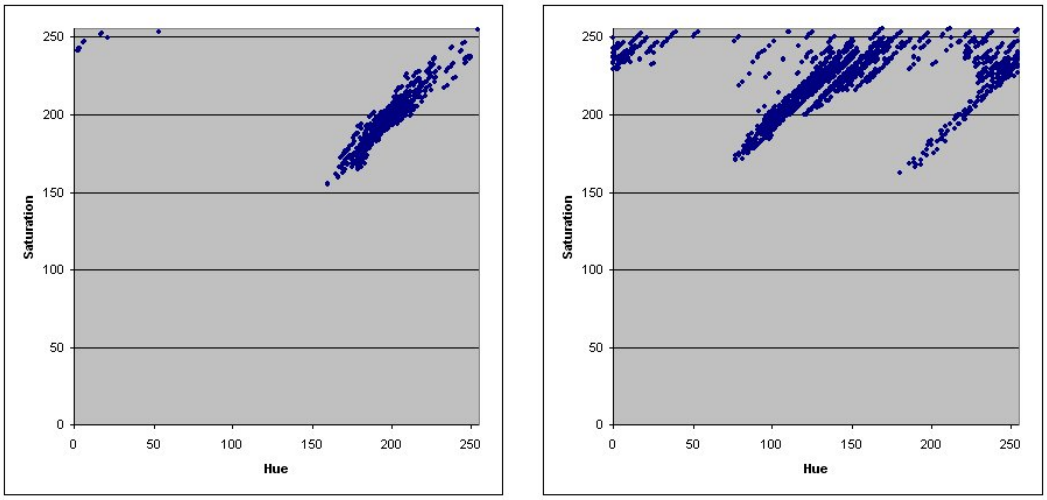
Abbildung 4.7: Kalibrierung des Systems auf die Hautfarbe des Anwenders

4.4.3 Erzeugung des Hautfilters

In beiden Schritten wird das aktuelle Videobild in Form einer Liste von RGB-Werten temporär gespeichert. Die RGB-Werte werden in den HSB-Farbraum umgerechnet, skaliert und in eine zweidimensionale Matrix der Größe 256 mal 256 eingetragen. Dabei werden nur der Farbton (**H**ue) und die Sättigung (**S**aturation) verwendet. Die Helligkeit (**B**rightness) bleibt unberücksichtigt, damit der Filter unempfindlich gegenüber sich änderndes Umgebungslicht ist. Die entstandenen Matrizen können auch als Binärbilder interpretiert werden. Für die Vordergrund-Matrix, die aus der Aufnahme der Hautfarbe entstanden ist, gilt also, dass genau die Stellen gesetzt sind, an denen die Hautfarbe des Anwenders im HSB-Farbraum zu finden sind. Es bilden sich Punktwolken, die die Farbtöne und Sättigungen wiedergeben. Die Abbildungen in 4.8 zeigen die Punktwolken für Vorder- und Hintergrund.

Die beiden Momentaufnahmen geben natürlich nicht das ganze Farbspektrum des Hintergrundes bzw. der Hand wieder. Daher werden die Matrizen mit der morphologischen Schließung (vgl. 2.1.2) verdichtet, so dass zusammenhängende Bereiche entstehen. Durch die Mengenoperation Differenz auf Vordergrund (F) und Hintergrund (B) werden sich deckende Bereiche in der Vordergrundmatrix gelöscht:

$$R := F \setminus B \tag{4.1}$$



(a) Vordergrund (Hautfarbe) (b) Hintergrund

Abbildung 4.8: Vorder- und Hintergrund im HSB-Farbraum

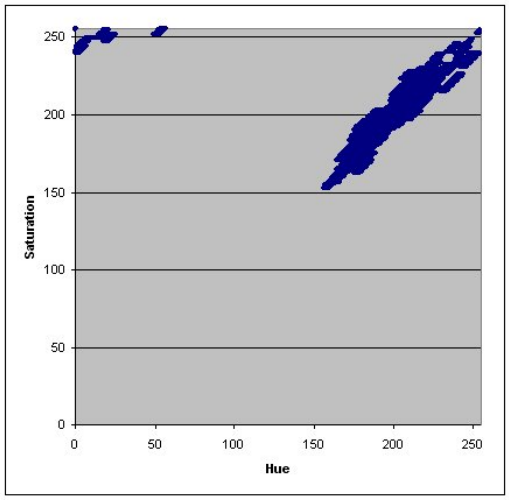
Übrig bleiben die Hauttöne (R), die nur an der Hand und nicht im Hintergrund vorkommen. Die Abbildung 4.9 zeigt das Resultat nach der Schließung und der Mengenoperation. Die rot markierten Bereiche sind die Farbtöne, die in Vorder- und Hintergrund gleichermaßen vorkamen.

Als Ergebnis erhält man einen Hautfilter. Um nun ein vorhandenes Bild zu filtern, wird jeder Bildpunkt vom RGB in den HSB-Farbraum umgerechnet. Dann wird geprüft, ob die Kombination aus Farbton und Sättigung jedes Bildpunkts in der Ergebnismenge R vorkommt. Falls ja, handelt es sich bei diesem Bildpunkt um Haut, andernfalls nicht. Aus dem Farbbild entsteht ein Binärbild, in dem Weiß den Hintergrund und Schwarz die Hautfarbe markiert. Je nach Erkennungsgrad des Hautfilters haben die erkannten Bereiche glatte Konturen oder weisen Löcher und ausgefranste Konturen auf. Um dem vorzubeugen, wird auf das Binärbild ebenfalls eine morphologische Schließung (siehe 2.1.2) angewandt. Die Abbildungen in 4.10 zeigen eine Hand in einem Farbbild sowie das Ergebnis nach Anwendung des Hautfilters und der Schließung.

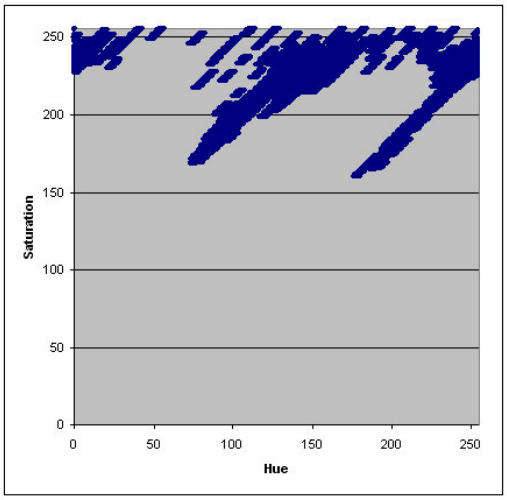
4.4.4 Vor- und Nachteile

Durch die Kalibrierung stellt sich das System sehr gut auf die spezielle Hautfarbe des Anwenders und den Hintergrund ein. Der Hautfilter arbeitet sehr schnell und ressourcenschonend.

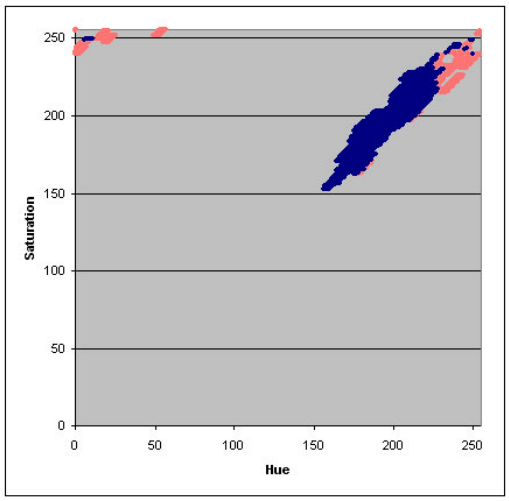
Die in 3.4.1 beschriebenen Verfahren sind allgemeiner gehalten und lieferten schlechtere Ergebnisse als die Kalibrierung. Jedoch sind der Kalibrierung Grenzen gesetzt. Kommen zu viele Hautfarben im Hintergrund vor, kann das System nicht mehr die ganze Hand erfassen. Die hautfarbenen Werte im Hintergrund werden ja von der Vordergrundmatrix entfernt. Im schlimmsten Fall kommen sämtliche Hautfarben auch in Objekten des Hintergrundes vor. Das System erkennt dann keine Hand mehr. Ein weiteres Problem sind Überstrahlungen



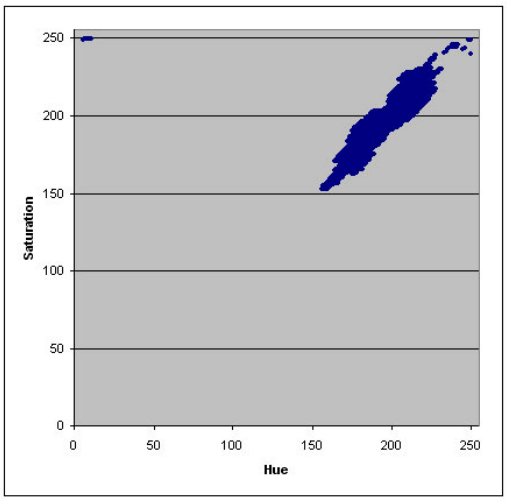
(a) Vordergrund (Hautfarbe)



(b) Hintergrund



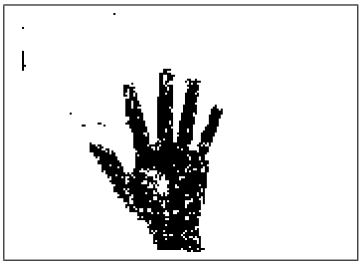
(c) Überschneidungen



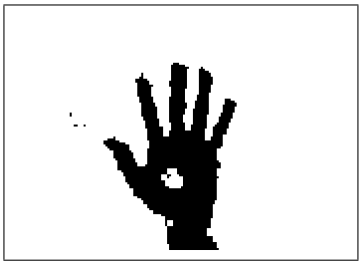
(d) Differenz

Abbildung 4.9: Schließung und Differenzbildung

(a) Original



(b) nach Hautfilter



(c) nach Schließung

Abbildung 4.10: Anwendung des Hautfilters

(siehe auch 4.2.3) durch direktes Licht. Werden diese strahlend weißen Bereiche auf der Hand bei der Kalibrierung mit eingeschlossen, so wird die Hand durch den Hautfilter in der Regel nicht vollständig erfasst.

4.5 Fingerspitzenerkennung

Eine zentrale Rolle bei der Positionsbestimmung und Mausklickerkennung spielt die Erkennung der Positionen der Fingerspitzen im Videobild. Voraussetzung für die Fingerspitzenerkennung ist eine Kontur, die die Randpunkte der Hand beinhaltet. Die Abbildungen 4.11, 4.12 und 4.13 in diesem Abschnitt sind als zusammengehörig zu betrachten.

4.5.1 Mittelpunkt der Handfläche

Der Mittelpunkt einer Handfläche befindet sich bei einer ausgestreckten Hand horizontal etwa in der Mitte und vertikal etwa an der Grenze zwischen dem mittleren und unteren Drittel. Im ersten Schritt wird um die Handkontur ein Rechteck (bounding box) gezogen. Die Eckpunkte (x_1, y_1, x_2, y_2) ergeben sich durch die Konturpunkte mit der minimalen und maximalen Position in horizontaler und vertikaler Richtung. Die horizontale Koordinate wird durch den horizontalen Mittelpunkt der bounding box, also $\frac{x_2-x_1}{2}$ bestimmt. Die vertikale Koordinate errechnet sich aus $y_1 + ((y_2 - y_1) * \frac{2}{3})$. Die Abbildung 4.11 zeigt eine Handkontur und die Ermittlung des Mittelpunktes. Natürlich ist dies nur eine Annäherung an den wirklichen Mittelpunkt. Es hat sich aber herausgestellt, dass die Genauigkeit für die weiteren Berechnungen vollkommen ausreicht.

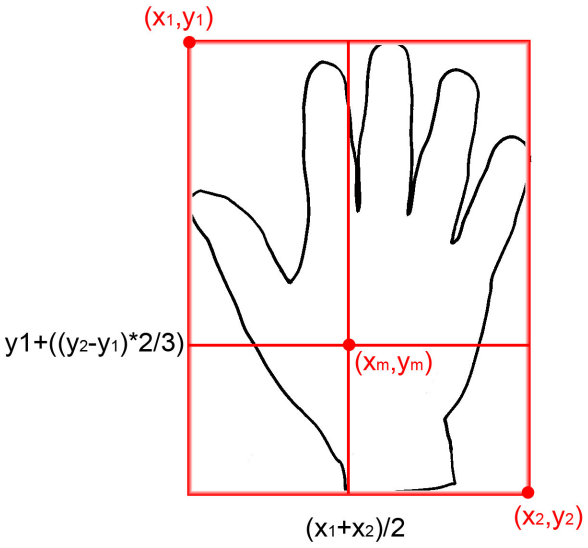


Abbildung 4.11: Ermittlung des Mittelpunktes der Handfläche

4.5.2 Polare Distanz

Mithilfe des Mittelpunktes der Handfläche (x_m, y_m) kann nun zu jedem Konturpunkt die *polare Distanz*, d.h. die Entfernung vom Mittelpunkt zum Konturpunkt, berechnet werden. Die Abbildung 4.12 zeigt die Konstruktion eines Dreiecks aus Kontur- und Mittelpunkt für die Anwendung des Satzes des Pythagoras. Begonnen wird mit dem untersten Konturpunkt, der sich meist am Handgelenk befindet. Dann wird im Uhrzeigersinn fortgefahren, bis man wieder beim Ausgangspunkt angelangt ist. Die polare Distanz d jedes Konturpunktes (x, y) wird ins Verhältnis zur geringsten polaren Distanz d_{min} in der Kontur gesetzt und um denselben Wert verringert. Als Ergebnis erhält man die relative polare Distanz d_r:

$$d = \sqrt{(x_m - x)^2 + (y_m - y)^2} \tag{4.2}$$

$$d_r = \frac{d - d_{min}}{d_{min}} \tag{4.3}$$

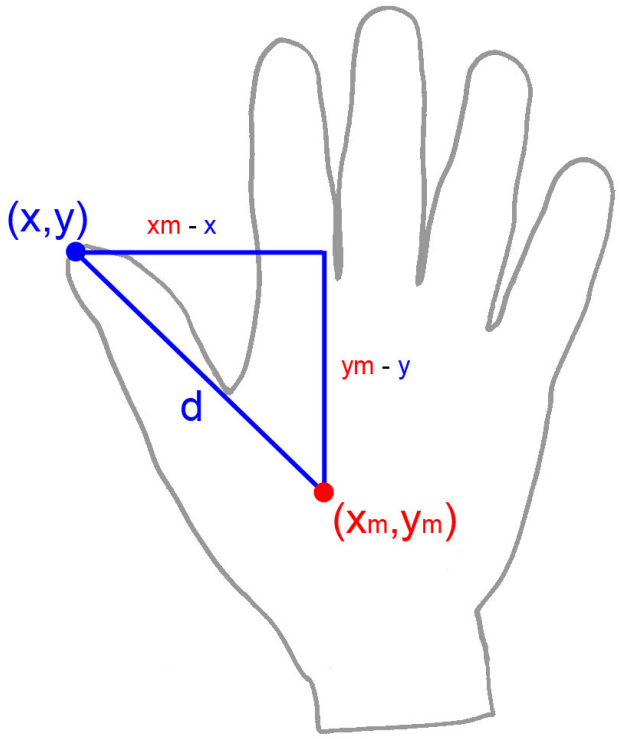


Abbildung 4.12: Konstruktion eines Dreiecks aus Kontur- und Mittelpunkt zur Berechnung der polaren Distanz

4.5.3 Minima- und Maxima-Bestimmung

Trägt man die relativen polaren Distanzen der Kontur in ein Koordinatensystem ein (siehe Abbildung 4.13), so erkennt man an den lokalen Maxima die Fingerspitzen und dazwischen an den lokalen Minima die Fingerzwischenräume. Eine ähnliche Vorgehensweise wird in [3] beschrieben. Folgende Schritte bestimmen die Konturpunkte, an denen lokalen Minima und Maxima vorkommen:

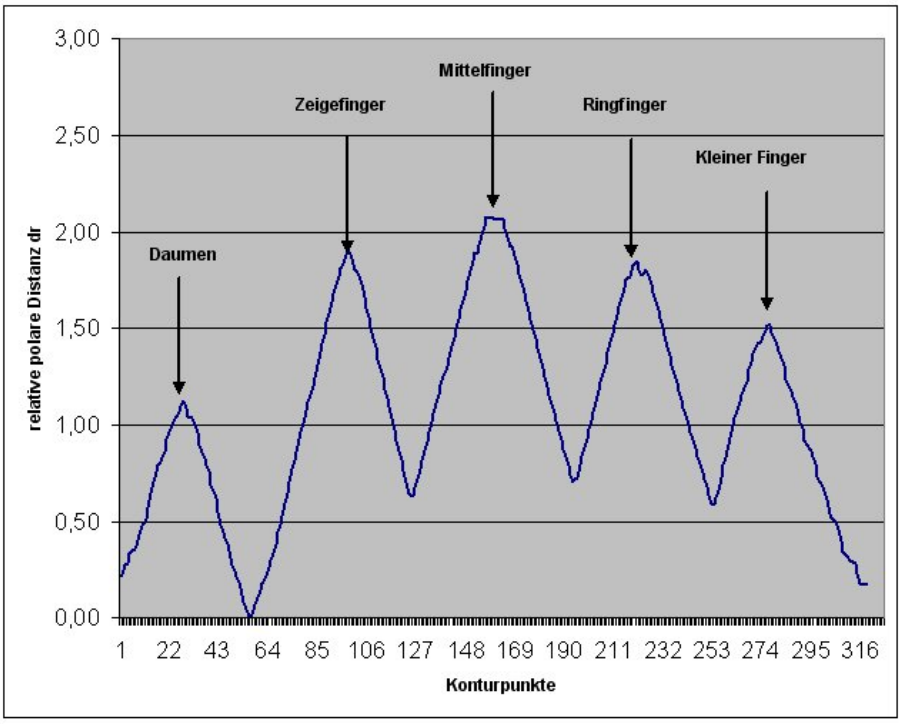


Abbildung 4.13: Polare Distanzen aller Konturpunkte im Koordinatensystem

1. Begonnen wird mit dem untersten Konturpunkt.
2. Der erste Konturpunkt wird als lokales Minimum notiert
3. Für jeden Konturpunkt wird nun die relative polare Distanz $d_r(i)$ errechnet. i gibt dabei den Index des Konturpunktes an.
4. Die Differenz $g = d_r(i) - d_r(i-1)$ wird berechnet. Diese entspricht der ersten Ableitung bzw. dem Gradienten g.
5. Zur Glättung einer eventuell sehr ungleichmäßigen Kontur wird der Mittelwert aus mehreren, umliegenden Gradienten gebildet. Die Anzahl der verwendeten Gradienten bestimmt sich aus der Anzahl der Konturpunkte c.
6. Sofern der Mittelwert der Gradienten vom Positiven ins Negative wechselt, wird dieser Konturpunkt als ein lokales Maximum notiert. Bei einem Wechsel vom Negativen ins Positive wird umgekehrt ein lokales Minimum notiert.
7. Der letzte Konturpunkt wird ebenfalls als lokales Minimum notiert.

Berechnungsbeispiel

Die Tabelle 4.1 zeigt ein Berechnungsbeispiel. Aus Platzgründen wurden nicht für alle Konturpunkte die Berechnungen aufgeführt, sondern nur jene, bei denen ein Vorzeichenwechsel bzw. ein Extremum gefunden wurde. Die Spalte „Gradienten-Glättung“ erschließt sich nicht aus der Tabelle, da jeder Wert auf umliegende Gradienten zurückgreift, die nicht mit abgedruckt wurden. Die Spalte wurde aber aufgeführt, weil in ihr der Vorzeichenwechsel ersichtlich ist.

Konturpunkt (x, y)	Polare Distanz d	Relative polare Distanz d_r	Gradient g	Gradienten-Glättung
...				
(78,32)	38,2099	0,8622	0,0485	-0,0087
(78,31)	39,2046	0,9107	0,0485	**-0,0003**
Vorzeichenwechsel von - nach + → relatives Minimum				
(77,30)	40,3113	0,9647	0,0539	**0,0074**
(77,29)	41,3038	1,0130	0,0484	0,0148
...				
(84,8)	62,0322	2,0233	-0,0487	0,0098
(84,9)	61,0328	1,9746	-0,0487	**0,0065**
Vorzeichenwechsel von + nach - → relatives Maximum				
(85,10)	60,0750	1,9279	-0,0467	**-0,0005**
(86,10)	60,1332	1,9307	0,0028	-0,0040
...				
(93,33)	38,6005	0,8813	0,0467	-0,0143
(93,32)	39,5601	0,9280	0,0468	**-0,0070**
Vorzeichenwechsel von - nach + → relatives Minimum				
(93,31)	40,5216	0,9749	0,0469	**0,0003**
(93,30)	41,4849	1,0219	0,0470	0,0076
...				
Berechnungsgrundlage:				
Mittelpunkt der Handfläche (x_m, y_m): (82,70)				
Minimaler polarer Abstand d_{min}: 20,52				
Anzahl Konturpunkte c: 410				
Glättung über $\frac{c}{30} = 13$ Gradienten				

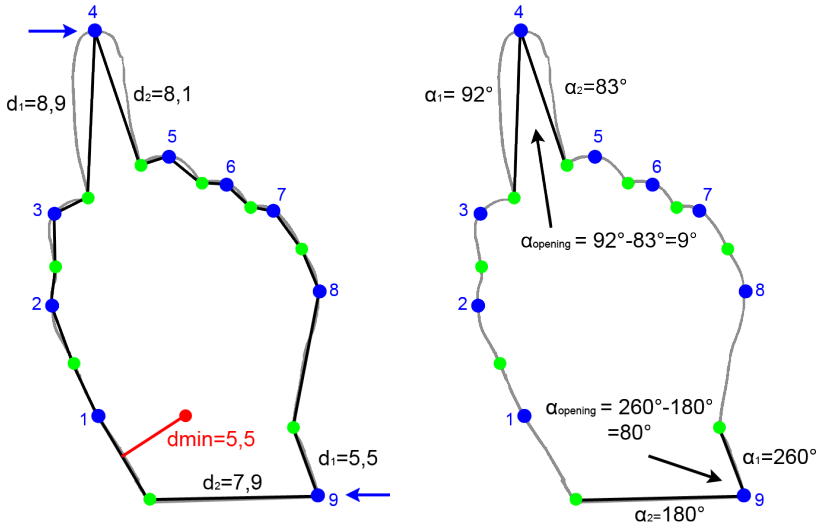
Tabelle 4.1: Berechnungsbeispiel zur Bestimmung von relativen Minima und Maxima

4.5.4 Filterung der Maxima

Nicht bei jedem gefundenen Maxima handelt es sich um eine Fingerspitze. Die Fingerknochen der gebeugten Finger oder auch Ungenauigkeiten in der Kontur führen schon zu einem lokalen Maximum. Daher müssen die gefundenen Maxima weiter untersucht und gefiltert werden:

Filterung durch Fingerlänge

Hierbei werden die Maxima im Verhältnis zu ihren umgebenden Minima untersucht. Dadurch, dass auch beim ersten und letzten Konturpunkt je ein lokales Minimum notiert wurde, liegt vor und hinter jedem Maximum stets ein Minimum. Nun wird die polare Distanz zwischen jedem Maximum und seinen zwei umliegenden Minima berechnet. Übersteigen beide Distanzen einen gewissen Schwellenwert, so handelt es sich bei diesem Maximum mit hoher Wahrscheinlichkeit um eine Fingerspitze. Dieser Schwellenwert wird durch d_{min} bestimmt. Durch einen Faktor, z.B. zwischen 0,5 und 1,5 auf d_{min}, kann auf unterschiedliche Fingerlängen von Anwendern reagiert werden. Die Abbildung 4.14(a) zeigt eine Handkontur, bei der der Zeigefinger ausgestreckt wurde. Die blauen Markierungen geben die gefundenen Maxima, die grünen die Minima an. Die Maxima sind durch schwarze Linien mit ihren umliegenden Minima verbunden. Die rote Linie gibt die kleinste polare Distanz d_{min} an. Wie man nun erkennen kann, kommen bei einem Faktor von 1 als Fingerspitzen die Maxima 4 und 9 in Frage, da die Distanzen zu den Minima d_{min} übersteigen. Maximum 9 könnte durch einen höheren Faktor, d.h. durch eine größere vorgegebene Fingerlänge, herausgefiltert werden.



(a) durch Fingerlänge (b) durch Öffnungswinkel

Abbildung 4.14: Filterung der Maxima

Filterung durch Öffnungswinkel

Eine weitere Möglichkeit, falsch erkannte Fingerspitzen wie beim Maximum 9 zu verhindern, ist die Filterung durch den Öffnungswinkel zwischen jedem Maximum (x_{max}, y_{max}) und den beiden umliegenden Minima (x_{min1}, y_{min1}) und (x_{min2}, y_{min2}). Dazu werden die Steigungen m_1 und m_2 der zwei Geraden durch das Maximum und seine umgebenden Minima errechnet. Aus den Steigungen bestimmen sich der Winkel α_1 und α_2 der Geraden. Der Öffnungswinkel $\alpha_{opening}$ berechnet sich durch die absolute Differenz der beiden Winkel:

$$m_1 = \frac{y_{max} - y_{min1}}{x_{max} - x_{min1}} \tag{4.4}$$

$$m_2 = \frac{y_{max} - y_{min2}}{x_{max} - x_{min2}} \tag{4.5}$$

$$\alpha_1 = \frac{\arctan m_1}{\pi} * 180 \tag{4.6}$$

$$\alpha_2 = \frac{\arctan m_2}{\pi} * 180 \tag{4.7}$$

$$\alpha_{opening} = |\alpha_1 - \alpha_2| \tag{4.8}$$

Wie in Abbildung 4.14(b) ersichtlich, ergibt sich beim Maximum 4 ein Öffnungswinkel von 9 Grad, beim Maximum 9 von 80 Grad. Durch den zu großen Öffnungswinkel beim Maximum 9 handelt es sich daher nicht um eine Fingerspitze.

4.5.5 Einschränkungen

Natürlich steht und fällt das Verfahren mit der Qualität der Kontur. Weist die Kontur große Ein- oder Ausbuchtungen auf oder umfließt sie nicht die gesamte Länge des Fingers, so werden Finger nicht korrekt erkannt. Im gewissen Maße kann auf ungleichmäßige Konturen mit einer stärkeren Glättung reagiert werden, aber hier sind natürlich Grenzen gesetzt. Voraussetzung für eine korrekte Fingerspitzenerkennung ist auch, dass der Anwender ein langärmeliges Kleidungsstück trägt, da ansonsten Teile des Arms mit als Hand interpretiert werden und die Berechnung des Mittelpunktes der Handfläche von falschen Voraussetzungen ausgeht. Umfließt die Kontur die Hand aber korrekt, funktioniert die Erkennung zuverlässig. Im Kapitel Test ab Seite 89 finden sich einige Beispiele und Testergebnisse zur Fingerpitzenerkennung.

4.6 Detail-Fingerspitzenerkennung

Auf das Videobild wurde zu Beginn eine Unterabtastung vorgenommen. Das heißt, dass alle Filter und Verarbeitungsschritte auf ein niedrig aufgelöstes Bild angewendet werden und die Rechenzeit dadurch drastisch reduziert wird. Die Reduzierung der Auflösung hat zwar keine Auswirkung auf die Ermittlung der Fingeranzahl, die Position der Fingerspitzen ist jedoch ungenauer. Daher wird die Fingererkennung in Originalauflösung wiederholt. Sie

wird jedoch auf einen kleinen Ausschnitt rund um die gefundene Fingerspitze beschränkt. Die Rechenzeit, die für die Unterabtastung und die weitere Fingererkennung anfällt, ist bei weitem geringer, als würde man alle Verarbeitungsschritte auf die Originalauflösung anwenden.

Die Abbildung 4.15 zeigt den Ablauf anhand eines Beispiels. Nachdem das Bild unterabgetastet und der Hautfilter angewandt wurde, ermittelt die Fingerspitzenerkennung die Position der Fingerspitze in der reduzierten Auflösung von 160 mal 120 Bildpunkten bei (77,16) bzw. skalierte aber ungenaue (154,32) in der Auflösung von 320 mal 240 Bildpunkten. In einem Ausschnitt von 40 mal 40 Bildpunkten rund um (154,32) wird erneut eine Fingerspitzenerkennung in der Original-Auflösung durchgeführt. Die Position der Fingerspitze in diesem Ausschnitt liegt bei (25,14). Im Ergebnis befindet sich die genaue Position der Fingerspitze bei (159,26). Die Abweichung der ungenauen Position (154,32) beträgt also 5 bzw. 6 Bildpunkte.

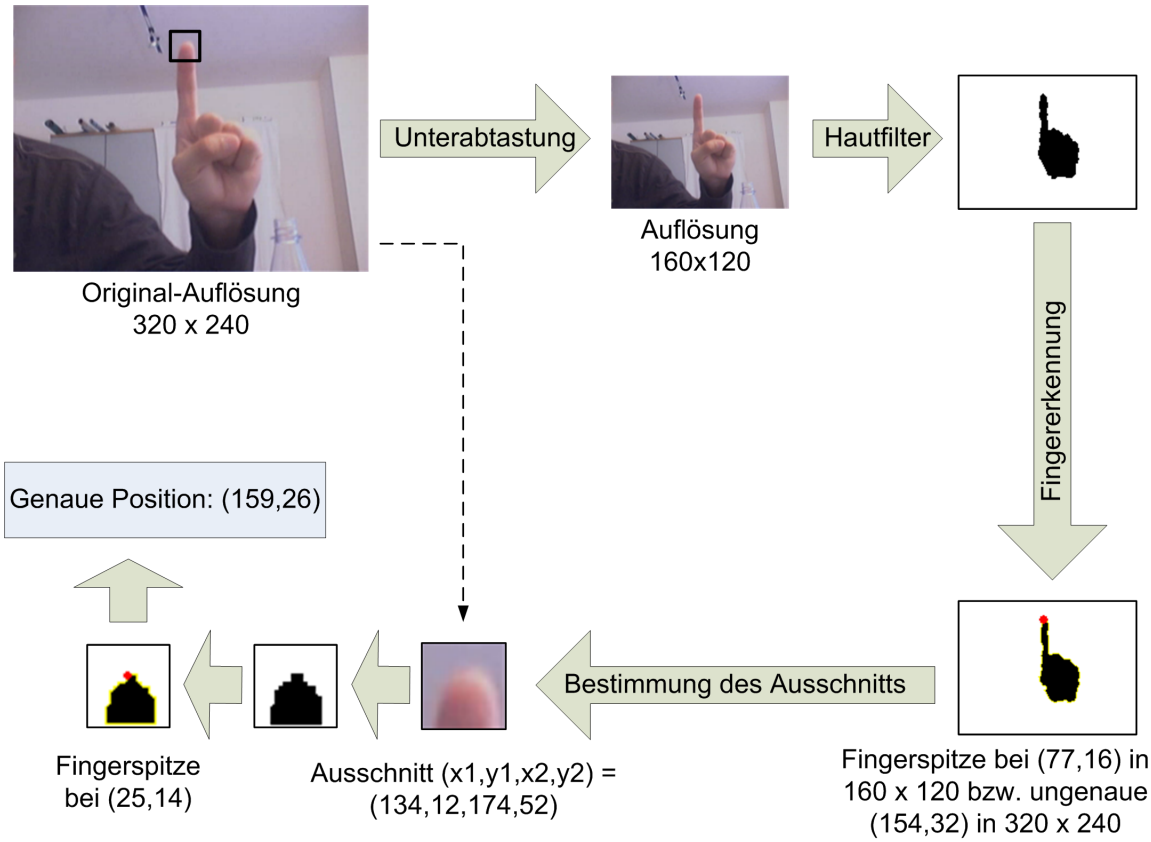


Abbildung 4.15: Beispiel-Berechnung der Fingerspitzen-Position

4.7 Bestimmung der Cursor-Position

Nachdem die Positionen der Fingerspitzen im Bild feststehen, wird für die Cursor-Position der längste Finger ausgewählt. Für die natürliche Zeigegeste benutzt der Mensch den

gestreckten Zeigefinger, während die restlichen Finger leicht zur Faust geballt sind. Daher wird der Zeigefinger als längster Finger erkannt, der Cursor lässt sich aber auch mit jedem anderen ausgestreckten Finger steuern. Die Position der Fingerspitze kann nun auf zwei Arten für die Cursor-Steuerung benutzt werden:

4.7.1 Direkte Steuerung (Maus-Modus)

Beim Maus-Modus wird die Position der Fingerspitze im Videobild direkt auf die Auflösung des Computermonitors umgerechnet (skaliert). Zu berücksichtigen ist, dass eine Position im unteren Bereich des Bildschirms so noch nicht erreicht werden kann. Bewegt man die Fingerspitze zu weit nach unten, verschwindet die Hand aus dem Sichtbereich der Kamera. Die Kontur ist nicht mehr vollständig und die Fingerspitzenerkennung liefert falsche Ergebnisse. Um dem zu entgegnen, wird der Aktionsbereich der Hand in den oberen Teil des Videobildes verlagert, so dass stets die ganze Hand im Videobild erfasst werden kann.

4.7.2 Indirekte Steuerung (Joystick-Modus)

Beim Joystick-Modus wird die Position der Fingerspitze im Videobild in eine Bewegungsrichtung umgerechnet. Verharrt die Fingerspitze in der Mitte des Videobildes (Ruhebereich), so bewegt sich die Maus nicht. Bewegt man nun die Fingerspitze beispielsweise nach rechts, wird der Cursor kontinuierlich nach rechts bewegt. Verbleibt die Fingerspitze an dieser Position, so wird sich der Cursor bis zum rechten Bildschirmrand bewegen. Je mehr man die Fingerspitze aus der Mitte bewegt, desto stärker wird der Cursor bewegt. Rund um die Mitte des Videobildes bewegt sich der Cursor nur sehr langsam. Je mehr man den Finger zum Rand des Videobildes bewegt, desto schneller ist die Bewegung. Die Beschleunigung lässt sich individuell einstellen. Um den Cursor anzuhalten, bewegt man die Fingerspitze wieder in die Mitte des Videobildes.

Rund um den Ruhebereich befindet sich der langsame Bereich. Befindet sich die Fingerspitze in diesem Bereich, wird der Cursor nur sehr langsam, d.h. nur Pixel für Pixel bewegt. So ist eine exakte Positionierung des Cursors möglich. Verlässt man den langsamen Bereich, gelangt man in den Aktionsbereich, in dem der Cursor je nach Position beschleunigt wird. Die Abbildung 4.16 veranschaulicht die verschiedenen Bereiche.

4.7.3 Vergleich der Modi

Wird ein Anwender das System zum ersten Mal benutzen, wird er den Maus-Modus als die natürlichere Art der Cursor-Steuerung empfinden. Jedoch besitzt der Maus-Modus einen Nachteil, den es zu beachten gibt:

Je nach Auflösung des Videobildes muss die Position der Fingerspitze mehr oder weniger stark auf die Bildschirmauflösung skaliert werden. Die Tabelle 4.2 zeigt den rechnerischen Skalierungsfaktor für verschiedene Auflösungen.

Bei einer Video-Auflösung von 160 mal 120 und einer Bildschirm-Auflösung von 1280 mal 1024 Pixeln, welche auch während der Entwicklung des Systems benutzt wurden, muss

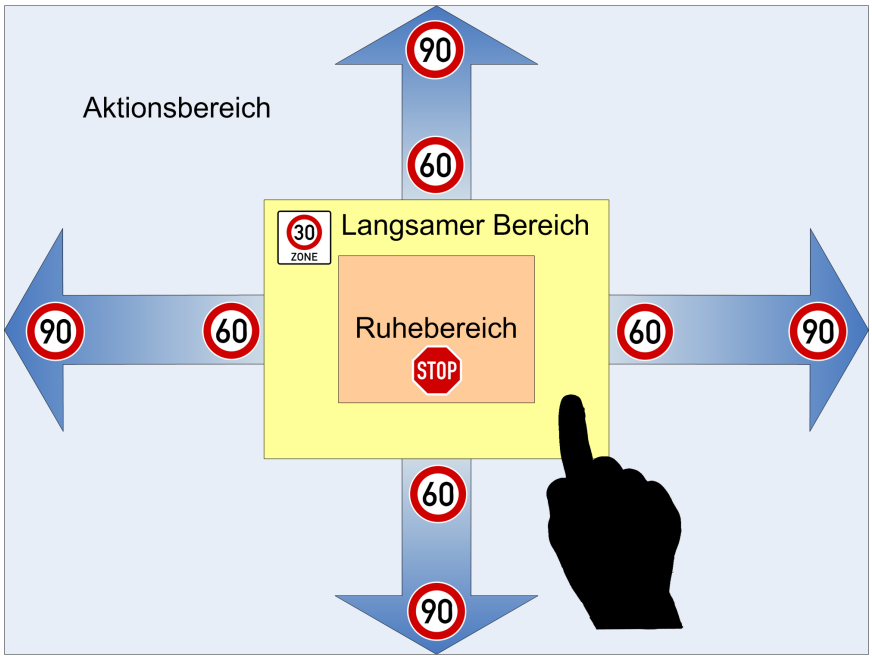


Abbildung 4.16: Joystickmodus

Video-Auflösung (Pixel)	Monitor-Auflösung (Pixel)	Skalierungsfaktor (rechnerisch)	Skalierungsfaktor (tatsächlich)
640 x 480	640 x 480	1	2
640 x 480	1024 x 768	1,6	3,2
640 x 480	1280 x 1024	2	4
320 x 240	1280 x 1024	4	8
160 x 120	1280 x 1024	8	16

Tabelle 4.2: Skalierungsfaktoren bei verschiedenen Video- und Monitor-Auflösungen

also eine Skalierung um den Faktor 8 vorgenommen werden. Bewegt man den Finger im Videobild um einen Pixel, so bewegt er sich auf dem Bildschirm gleich um volle 8 Pixel. So ist es aber fast unmöglich, die sehr kleinen Steuerelemente einer heutigen grafischen Benutzeroberfläche zu treffen. Bei einer Verdopplung der Video-Auflösung auf 320 mal 240 sind es immer noch 4 Pixel.

Dazu kommt, dass gar nicht die ganze Größe des Videobildes zur Verfügung steht. Bewegt man den Zeigefinger an den unteren Rand des Videobildes, so wird die Hand nicht mehr vollständig erfasst, da der Großteil der Hand sich außerhalb des Videobildes befindet. Für die Fingererkennung muss jedoch die gesamte Handkontur sichtbar sein. Ähnlich verhält es sich mit dem linken und rechten Rand des Videobildes. Die rote Umrandund in Abbildung 4.17 zeigt, dass weniger als die Hälfte des Videobildes als Aktionsbereich bei der Positionssteuerung mit dem Zeigefinger übrig bleibt.

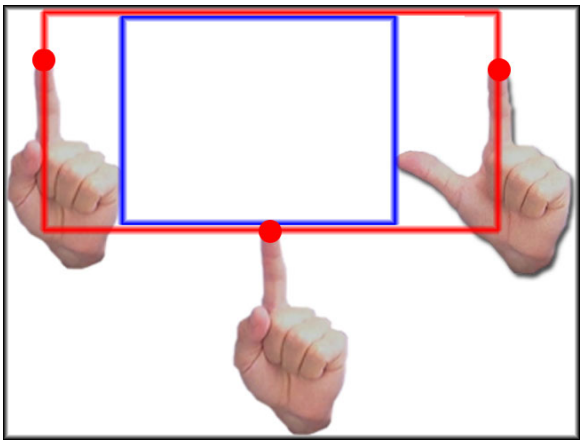


Abbildung 4.17: Aktionsbereich bei Positionssteuerung (rot) und Gestenerkennung (blau)

Für eine Gestenerkennung von z.B. zwei Fingern schränkt sich der Aktionsbereich auf ca. ein Viertel ein (blaue Umrandung). Dabei wird davon ausgegangen, dass der Anwender am Schreibtisch sitzt, die Kamera in der Nähe des Monitors steht und die Entfernung von der Hand zur Kamera ca. 50 cm beträgt. Je näher der Anwender die Hand an die Kamera führt, umso größer wird die Hand im Videobild und umso kleiner wird der Aktionsbereich. Durch die Halbierung in der Breite als auch in der Höhe wächst der tatsächliche Skalierungsfaktor auf über 16 Bildpunkte an. Selbst wenn nun die Video-Auflösung auf 640 mal 480 Bildpunkte erhöht wird, beträgt der Skalierungsfaktor immer noch 4. Die Abbildung 4.18 zeigt die Dimensionen bei einer Skalierung von einer realistischen Kamera-Auflösung von 320 mal 240 Bildpunkten, von der ca. 160 mal 120 Bildpunkte übrig bleiben, auf eine heute gebräuchliche Bildschirm-Auflösung von 1280 mal 1024 Bildpunkte.

Damit die Cursorsteuerung möglichst präzise ist und einzelne Bildpunkte ansteuerbar sind, muss die Video-Auflösung mindestens doppelt so groß sein wie die Bildschirmauflösung. Bei einer heutzutage gebräuchlichen Bildschirm-Auflösung von 1280 mal 1024 Bildpunkte muss die Video-Auflösung mindestens 2560 mal 2048 Bildpunkte betragen. Abgesehen davon, dass es kaum Videokameras für den Heimgebrauch mit solch einer Auflösung zu einem vertretbaren Preis gibt, fallen derart hohe Datenmengen an, die mit einem normalen Computer nicht zu bewältigen sind. Zusammenfassend lässt sich sagen:

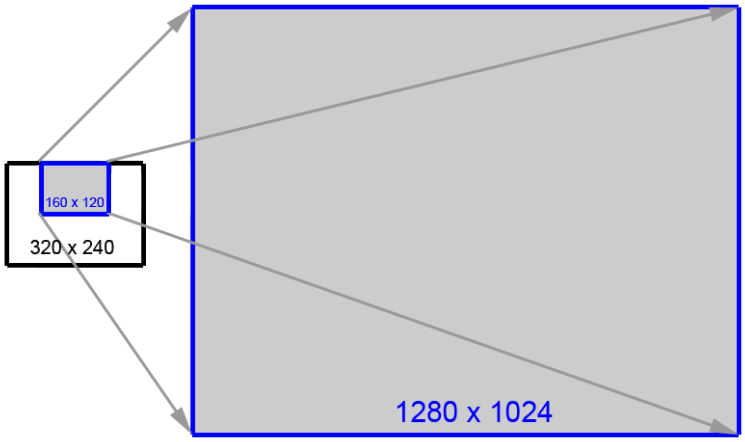


Abbildung 4.18: Skalierung der Video- auf die Bildschirmauflösung um den Faktor 8

- Je näher die Hand zur Kamera gehalten wird, desto stärker wird der Aktionsbereich eingeschränkt, und desto geringer wird die resultierende Video-Auflösung
- Je niedriger die Video-Auflösung, desto weniger Datenmengen fallen an, aber desto gröber ist die Cursorbewegung
- Je höher die Video-Auflösung, desto präziser ist die Cursorsteuerung, aber desto höher ist das Datenvolumen

Einen geeigneten Mittelweg zu finden, gestaltet sich schwierig. Die Lösung liegt im Joystick-Modus, da das Problem mit der Auflösung dort nicht so deutlich zum Tragen kommt. Durch die indirekte Steuerung reicht eine geringe Auflösung aus. Es spielt keine Rolle, ob man den Cursor in einem Bereich von 80 oder 800 Bildpunkten Breite in eine Richtung beschleunigt. Die 80 bzw. 800 Bildpunkte entsprechen dabei der Strecke von der Mitte des Videobildes bis zum linken oder rechten Rand bei einer Breite des Videobildes von 160 bzw. 1600 Bildpunkten. Die Steuerung des Cursors kann so pixelgenau erfolgen. Dafür ist die indirekte Steuerung im Joystick-Modus schwieriger und bedarf viel Übung. Der Vorteil der intuitiven Steuerung des Cursors mit dem bloßen Zeigefinger wird dadurch wieder geschmälert.

Denkbar wäre auch eine Funktionsweise wie bei der herkömmlichen Computermaus. Bei ihr wird die Bewegungsänderung in absolute Bildschirmkoordinaten umgesetzt. Bewegt man die Maus nur ein wenig, erfolgt eine langsame Cursorbewegung. Wird die Maus dagegen ruckartig bewegt, beschleunigt der Cursor schnell über den Bildschirm.
Leider lässt sich diese Funktionsweise nicht auf das Motion Tracking-System übertragen. Befindet sich z.B. der Cursor in der Mitte des Bildschirmes, der Finger aber bereits ganz links im Videobild, ist es nicht möglich, den Cursor weiter nach links zu bewegen, da die Hand dann das Videobild verlässt. Eine Computermaus würde man in diesem Fall einfach kurz vom Mauspad anheben und etwas weiter rechts wieder absetzten. Die Hand müsste man komplett aus dem Videobild entfernen und von rechts wieder ins Videobild bewegen, was wenig praktikal erscheint.
Eine Computermaus hat im Gegensatz zur Videokamera eine sehr viel höhere Auflösung.

Auch wenn man die Maus sehr langsam bewegt, reicht das Mauspad für die gesamte Bildschirmbreite aus. Dagegen ist bei langsamen Bewegungen der Hand der Rand des Videobildes schnell erreicht.

4.8 Erkennung von Mausklicks

Als erstes stellt sich die Frage, welche Handgeste als Klick mit der linken Maustaste interpretiert werden soll.

4.8.1 Linke Maustaste

Bei einer Computermaus benutzt man zum Klicken den Zeigefinger. Die Beugung des Zeigefingers betätigt die Maustaste und löst den Mausklick aus. Dieses Beugen stellt also schon eine Handgeste dar und könnte einfach auf das System übertragen werden. Denkt man jedoch etwas weiter, wird einem klar, dass diese Handgeste so nicht funktionieren kann, denn der Zeigefinger wird schon für die Positionsbestimmung benutzt. Würde man ihn beugen, um den Mausklick auszulösen, verändert sich gleichzeitig die Position des Cursors. Das System kann zu diesem Zeitpunkt nicht feststellen, an welcher Stelle der Mausklick ausgelöst werden soll.

Eine Alternative ist das Benutzen von zwei Fingern. Während man mit einem Finger die Position des Cursors bestimmt, streckt man kurz einen zweiten Finger aus. Am besten haben sich dabei Zeigefinger und Daumen bewährt. Der gestreckte Zeigefinger bestimmt die Positions, der Daumen wird kurz abgespreizt und löst damit den linken Mausklick aus (siehe Abbildung 4.19). Einen anderen Finger als den Daumen zu benutzen, ist zwar ohne weiteres möglich, jedoch sehr anstrengend. Es fällt einem sehr schwer, einen der verbleibenden drei anderen Finger unabhängig vom Zeigefinger zu bewegen und dabei den Zeigefinger in seiner Position unverändert zu lassen. Der Daumen ist einfach viel beweglicher. Für einen Doppelklick wird die Handgeste zweimal kurz hintereinander durchgeführt.

4.8.2 Zählen der Finger

Das Erkennen des Mausklicks ist mit einem einfachen Zählen der gestreckten Finger zu realisieren. Wenn sich die Anzahl der Finger von 1 zu 2 und dann wieder zu 1 ändert (Sequenz *1-2-1*), wird ein Mausklick ausgeführt. Dabei wird bei jeder Änderung der Fingeranzahl die aktuelle Fingeranzahl und die Dauer, wie lange sie bestand, in einer Liste gespeichert. Anschließend werden die letzten drei Einträge der Liste auf Übereinstimmung mit der Sequenz *1-2-1* geprüft. Zur Verdeutlichung zeigt die Tabelle 4.3 eine solche Liste.

In der Liste fällt auf, dass die Sequenz *1-2-1* mehrfach auftaucht. Jedoch wurden zwei Mal die Veränderungen in der Fingeranzahl nicht aufgezeichnet, da die Dauer jeweils kleiner als 100 oder größer als 1000 Millisekunden war. Bei der kurzen Dauer von nur 54 Millisekunden handelte es sich mit großer Wahrscheinlichkeit um Fehler in der Fingerspitzenerkennung, verursacht z.B. durch plötzlich auftretende Löcher in der Kontur. Ein normaler Anwender schafft es in der Regel nicht, einen Finger in unter 100 Millisekunden abzuspreizen und

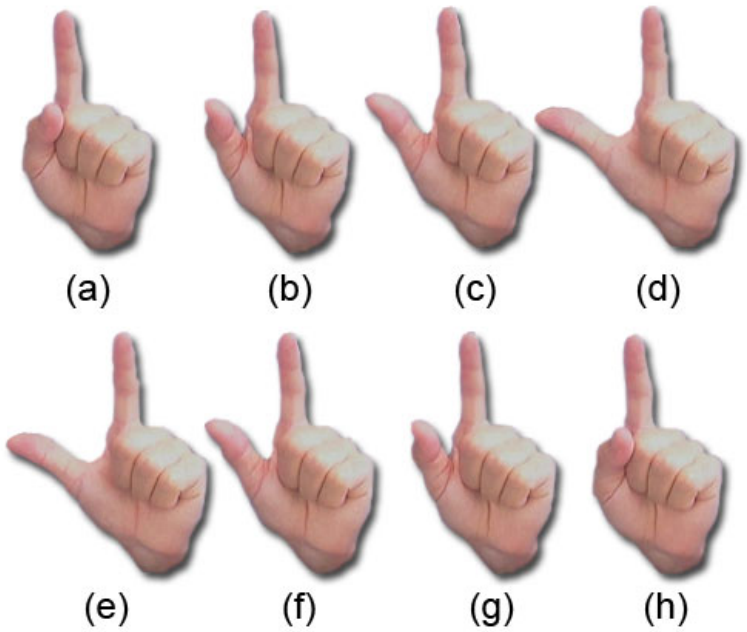


Abbildung 4.19: Handgeste zum Auslösen eines Klicks mit der linken Maustaste

Zeile	Anzahl Finger	Dauer (ms)	Position (x,y)	Kommentar
0	0	2.343	(88,40)	geballte Faust
1	1	24.237	(100,12)	24 Sekunden wird der Cursor mit dem Zeigefinger bewegt
2	2	54	(100,12)	Diese Aktion wird nicht aufgezeichnet, da Dauer < 100
3	1	1.298	(120,90)	12 Sekunden wird der Cursor mit dem Zeigefinger bewegt
4	2	1.566	(12,20)	Diese Aktion wird nicht aufgezeichnet, da Dauer > 1000
5	1	25.551	(15,12)	25 Sekunden wird der Cursor mit dem Zeigefinger bewegt
6	2	256	(50,88)	Diese Aktion wird nicht aufgezeichnet, da der Klickbereich verlassen wurde
7	1	9.501	(60,15)	9 Sekunden wird der Cursor mit dem Zeigefinger bewegt
8	2	210	(60,16)	Daumen wird kurz abgespreizt
9	1	-	(59,16)	Daumen wurde wieder zurückbewegt - Mausklick wird ausgeführt

Tabelle 4.3: Aufzeichnung der Anzahl und Dauer von gestreckten Fingern einer Hand

wieder zurückzuführen. Daher werden solche Aktionen herausgefiltert, sie würden sonst zu unerwünschten Mausklicks führen. Auch wenn ein zweiter Finger länger als 1000 Millisekunden abgespreizt wird, wird nicht von einem Mausklick ausgegangen. In der Zeile 8 wurden zwei Finger für 210 Millisekunden in die Kamera gehalten in Zeile 9 ein Finger wieder zurückgeführt. Der Mausklick wird demnach ausgeführt.

4.8.3 Klickbereich

Bei der normalen Computermaus muss die Position des Cursors während des Klickens in einem definierten Bereich gehalten werden. Bewegt sich der Cursor während des Klickens zu stark, wird der Mausklick nicht ausgeführt bzw. ein Drag- und Drop-Vorgang ausgelöst. Dies ist z.B. bei Anwendern zu beobachten, die das erste Mal eine Maus in die Hand nehmen und sich wundern, dass die Maustaste nicht funktioniert. Sie bewegen die Maus, während sie auf die Maustaste drücken, zu sehr und aus dem Klickbereich heraus. Dieser Klickbereich muss auch beim Klicken mit dem abgespreizten Finger eingehalten werden. In der Tabelle 4.3 erscheint zwar in den Zeilen 5 bis 7 die Sequenz *1-2-1* und auch die Dauer von 100 bis 1000 Millisekunden wird eingehalten, während des Klickens bewegt sich der Cursor jedoch zu stark. Dagegen bewegt sich der Cursor in den Zeilen 7 bis 9 nur um einen Bildpunkt nach unten und dann um einen Bildpunkt nach links. Diese Bewegung liegt in der Toleranzgrenze. Zur Berechnung, ob der Klickbereich verlassen wurde, werden die minimalen und maximalen Positionen in waagerechter und senkrechter Richtung gespeichert. Daraus wird eine bounding box gebildet. Übersteigt die Größe der bounding box eine bestimmte Größe nicht, wurde der Klickbereich nicht verlassen. Während der Entwicklung des Systems hat sich eine Größe von 5 bis 10 Bildpunkten bewährt.

Der Klickbereich funktioniert des Weiteren als ein Filter, der ungewollte Mausklicks vermeidet. Durch Ungenauigkeiten in der Handkontur kann per Zufall die Handgeste für einen Mausklick ausgelöst werden, auch wenn man in diesem Augenblick vielleicht nur die Position des Mauszeigers verändern wollte. Neben dem zeitlichen Filter (Dauer zwischen 100 und 1000 Millisekunden) sorgt nun auch mit dem Klickbereich ein örtlicher Filter dafür, dass ein Mausklick nur in einem fest definierten Bereich auftritt.

4.8.4 Rechte Maustaste und Mausrad

Für den Klick mit der rechten Maustaste wird ein dritter Finger benutzt. Während Zeige- und Mittelfinger gestreckt in die Kamera gehalten werden, wird kurz der Daumen abgespreizt und wieder zurückgeführt.

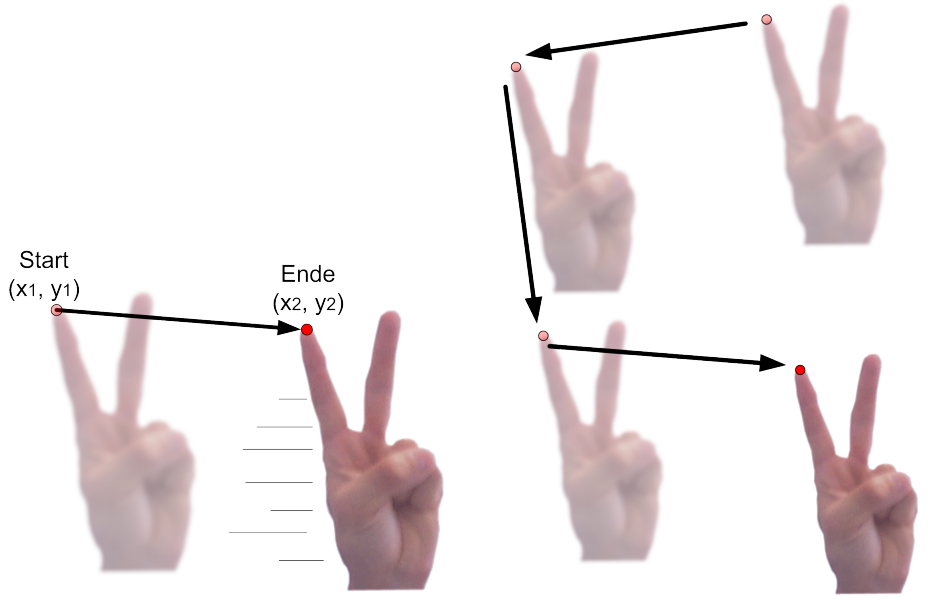
Inzwischen verfügt so gut wie jede Maus über ein Scroll- bzw. Mausrad, mit dem der Fensterausschnitt verändert werden kann, ohne die Bildlaufleisten benutzen zu müssen. Für die Benutzung des Mausrads werden alle fünf Finger gestreckt. Durch eine Bewegung der Hand nach oben und unten wird der Fensterausschnitt entsprechend verschoben.

4.9 Erkennung komplexer Handgesten

Die Informationen über die Position und Anzahl der Fingerspitzen lassen sich auch für komplexe Handgesten ausnutzen. Das Verfahren ähnelt den Mausgesten, die z.B. beim Internet-Browser Opera eingesetzt werden können. Durch Drücken der rechten Maustaste und einer Bewegung der Maus nach links lässt sich bei Opera die vorhergehende Internet-Seite aufrufen. Eine Bewegung nach rechts ruft die nächste Seite auf.

Diese Mausgesten können auch mit der Hand ausgeführt werden. Dazu werden zwei Finger ausgestreckt und die Hand ruckartig nach links, rechts, oben oder unten bewegt. Dabei wird vom System die horizontale und vertikale Distanz zwischen dem Start und dem Ende der Geste berechnet. Übersteigt eine der beiden Distanzen eine gewisse Größe, wird die Bewegungsrichtung ermittelt.

Die Abbildung 4.20(a) zeigt eine Handgeste, bei der der Start der Bewegung bei (x_1, y_1) und das Ende bei (x_2, y_2) liegt. Die horizontale Distanz $d_x = |x_2 - x_1|$ übersteigt in diesem Fall die vertikale Distanz $d_y = |y_2 - y_1|$. Der horizontale Start der Bewegung x_1 ist kleiner als das Ende der Bewegung x_2, demnach wird eine Bewegung nach rechts erkannt.



(a) Handgeste nach rechts (b) Handgeste in Form des Buchstabens C

Abbildung 4.20: Handgesten

Dementsprechend werden die anderen Bewegungsrichtungen nach links, oben und unten ermittelt. Die Bewegungen können kombiniert werden, so dass komplexe Handgesten möglich sind. Eine Bewegung der Hand in der Form des Buchstabens C nach links, nach unten und dann nach rechts (siehe Abbildung 4.20(b)) könnte beispielsweise den Taschenrechner (Calculator) starten.

4.10 Benutzeroberfläche

Entsprechend den Anforderungen in 3.6.3 wurde eine Benutzeroberfläche entworfen, die in den Abbildungen 4.21 und 4.22 dargestellt ist. Die einzelnen Kontrollelemente werden im Folgenden erklärt.

4.10.1 Hauptfenster

Im Hauptfenster (siehe Abbildung 4.21) sind die wichtigsten Kontrollelemente zur Bedienung des Systems erreichbar:

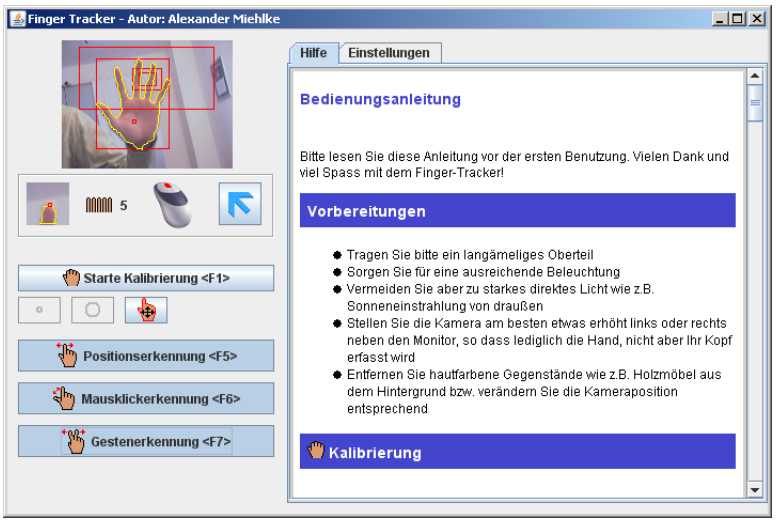


Abbildung 4.21: Benutzeroberfläche (Registerkarte Hilfe)

Videobild Zur Kontrolle der Kameraeinstellung und der Erkennungsgenauigkeit wird dem Anwender das aktuelle Videobild eingeblendet. In das Videobild können über die Einstellungen Markierungen wie die erkannte Handkontur und die Positionen der Fingerspitzen eingeblendet werden.

Detail-Fingerspitzen-Erkennung Sofern das System eine Unterabtastung vornimmt, wird die erkannte Fingerspitze in diesem Fenster eingeblendet.

Fingeranzahl Hier wird die Anzahl der erkannten Finger angezeigt.

Status Anhand einer abgebildeten Maus wird der Status der Positionssteuerung und der erkannte Mausklick visuell dargestellt.

Nur Videofenster anzeigen Mit dieser Schaltfläche verkleinert sich das Fenster auf die Größe des Videofensters. Ein erneuter Klick stellt das Fenster wieder in der ursprünglichen Größe dar.

Starte Kalibrierung Über diese Schaltfläche startet der Anwender die Kalibrierung auf den Hintergrund und seine Hautfarbe.

Größer und Kleiner Zur Kalibrierung der Hautfarbe wird dem Anwender eine Markierung ins Videobild eingeblendet, in der er seine Hand hält. Mit diesen Schaltflächen kann diese Markierung verkleinert und vergrößert werden.

Hautfarbe erweitern Mit dieser Schaltfläche kann die Hautfarbe erweitert werden, falls die Kontur die Hand nicht vollständig umschließt.

Positionserkennung Über diese Schaltfläche wird die Erkennung der Hand im Videobild aktiviert. Ab diesem Zeitpunkt wird der Cursor auf dem Bildschirm mit der Hand gesteuert. Ein nochmaliger Klick auf die Schaltfläche deaktiviert die Erkennung wieder.

Mausklickerkennung Die Schaltfläche startet die Erkennung für den Mausklick. Ein nochmaliger Klick auf die Schaltfläche deaktiviert die Erkennung wieder.

Gestenerkennung Diese Schaltfläche startet die Gestenerkennung. Ein nochmaliger Klick auf die Schaltfläche deaktiviert die Erkennung wieder.

4.10.2 Einstellungen

Über den Reiter *Einstellungen* (siehe Abbildung 4.22) kann das Verhalten des Systems gesteuert werden.

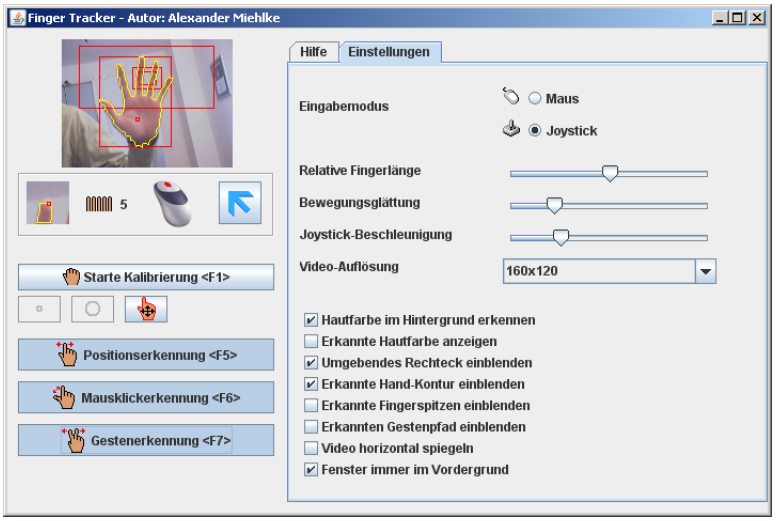


Abbildung 4.22: Benutzeroberfläche (Registerkarte Einstellungen)

Eingabemodus Diese Option bestimmt den Eingabemodus, d.h. ob die Steuerung wie eine Computermaus (siehe 4.7.1) oder ein Joystick (siehe 4.7.2) funktioniert.

Relative Fingerlänge Über diesen Schieberegler lässt sich die relative Fingerlänge einstellen. Falls Fingerspitzen nicht erkannt werden, weil ein Schwellenwert nicht überschritten wird, kann mit dieser Einstellung der Schwellenwert herabgesetzt werden.

Bewegungsglättung Bei einer zittrigen Bewegung des Cursors im Eingabemodus Maus kann mit diesem Schieberegler die Bewegung geglättet werden. Zu beachten ist, dass dadurch die Reaktion des Systems auf die Handbewegung verlangsamt wird.

Joystick-Beschleunigung Je weiter der Schieberegler nach rechts geschoben wird, umso stärker wird der Cursor im Joystick-Modus beschleunigt.

Video-Auflösung In dieser Liste werden die von der angeschlossenen Kamera unterstützten Auflösungen angezeigt und können ausgewählt werden.

Hautfarbe im Hintergrund erkennen Durch diese Einstellung werden Hautfarben, die gleichzeitig im Vorder- und Hintergrund vorkommen, erkannt und ausgefiltert.

Erkannte Hautfarbe anzeigen So kann überprüft werden, wie genau die Hand erkannt wurde und ob sich ggf. hautfarbene Gegenstände im Hintergrund befinden.

Umgebendes Rechteck einblenden Diese Option blendet ein Rechteck um die Hand ein.

Erkannte Hand-Kontur einblenden Diese Option zieht eine Kontur um die erkannte Hand.

Erkannte Fingerspitzen einblenden Diese Option markiert die erkannten Fingerspitzen. Dabei werden der längste Finger, der ausschlaggebend für die Positionssteuerung ist, rot, alle anderen Fingerspitzen hellblau markiert.

Erkannten Gestenpfad einblenden Diese Einstellung blendet im Videobild die erkannten Gesten ein. Dabei wird Anfang und Ende der Geste mit einer weißen Linie markiert. Mehrere aufeinanderfolgende Gesten wurde durch verbundene Linien dargestellt.

Video horizontal spiegeln Diese Einstellung spiegelt das Videofenster. Falls man die Hand nach links bewegt, wird sie sich auch im Videofenster nach links bewegen.

Fenster immer im Vordergrund Diese Einstellung bewirkt, dass dieses Fenster nicht von anderen Fenstern verdeckt werden kann.

5 Implementierung

Die im Entwurf erläuterten Verfahren und Techniken werden nun implementiert. Nachdem die Entscheidung für Software-Framework getroffen wurde, werden in diesem Kapitel ausgewählte Klassen kurz erklärt und deren Methoden mit Code-Ausschnitten beschrieben.

5.1 Wahl des Software-Frameworks

In den Grundlagen unter 2.2 und 2.3 wurden zwei für die Entwicklung dieses Systems in Frage kommenden Frameworks beschrieben. Unter beiden Frameworks wurden zu Beginn dieser Arbeit Testimplementierungen vorgenommen, um festzustellen, welches sich besser eignet. Beide stellen zahlreiche Beispiele zur Verfügung, um so einen schnellen Einstieg zu ermöglichen. Auf Basis dieser Beispiele wurde begonnen, zwei kleine Anwendungen zu programmieren, die letztlich folgendes leisten sollten:

1. Ansteuerung einer Webcam und Anzeige des Live-Kamerabildes
2. Veränderung der Kamera-Auflösung im laufenden Betrieb
3. Auslesen eines einzelnen Frames und Konvertierung des Frames in ein Farbbild
4. Anwendung von verschiedenen Filtern, wie z.B. Mittelwert- oder Binärfilter, auf dieses Farbbild
5. Anzeige des gefilterten Frames

Natürlich traten, wie bei jeder erstmaligen Verwendung eines Frameworks, zahlreiche Probleme auf. So gelang es unter dem Java Media Framework unter anderem nicht, die Kameraauflösung im laufenden Betrieb zu verändern. Es war stets ein Neustart des Systems nötig. Unter OpenCV gab es unter anderem Probleme, den gefilterten Frame wiederzugeben. Mithilfe der Entwicklerforen für JMF [10] und OpenCV [11] wurde versucht, die Probleme in den Griff zu bekommen. Dabei stellte sich die gute Unterstützung seitens der Entwickler im JMF-Forum heraus, während die Fragen im OpenCV Forum wochenlang unbeantwortet blieben. Für beide Frameworks gibt es keine Literatur, insofern ist man auf die Online-Dokumentationen und Internet-Foren angewiesen. Dies gab letztlich auch den Ausschlag für die Verwendung des Java Media Frameworks.

5.2 Entwicklungswerkzeuge

Für die Entwicklung des Systems wurden die nachfolgend aufgeführten Programme und Werkzeuge benutzt:

- Java-Compiler: Sun Java 1.6.0_01
- Entwicklungsumgebung: Eclipse Framework 3.3.0
- Kamera-Ansteuerung: Java Media Framework 2.1.1e
- UML-Diagramme: Omondo EclipseUML 2007 Free Edition
- Logging: Log4J 1.2.14
- Globale Windows-Tastenkombinationen: melloware JIntelligentType 1.2
- Tests: TPTP-4.4.0.1 (Test & Performance Tools Platform Project)
- JAR-Erstellung: Fat Jar Eclipse Plug-In 0.0.25
- Bildverarbeitungsprogramme: Adobe Photoshop CS2, Paint Shop Pro 4 und Ulead GIF Animator LITE Edition 1.0
- Klänge: Nero WaveEditor 3.5.6.0

5.3 Voreinstellungen

Die Voreinstellungen für das System sind in einer *properties*-Datei abgelegt. Bei Änderungen ist es so nicht notwendig, den Quelltext neu zu übersetzen. Die Voreinstellungen können leicht mit einem Texteditor angepasst werden. Die Datei *config.properties* enthält mehrere Abschnitte, die im Folgenden auszugsweise beschrieben werden. Details zu allen Voreinstellungen sind in der Datei selbst ersichtlich.

Sprache

Dieser Abschnitt enthält die Beschriftungen der Kontrollelemente in der Benutzeroberfläche. So ist es ohne weiteres möglich, die Beschriftungen in eine andere Sprache zu übersetzen. Das System enthält die Dateien *config.de.properties* und *config.en.properties* für deutsche und englische Sprache. Je nach gewünschter Sprache muss eine der beiden Dateien nur noch nach *config.properties* umbenannt werden. Hier nun ein Ausschnitt aus der *config.de.properties*:

```
1  #
2  # GUI text
3  #
4
5  frame.title=Finger Tracker - Autor: Alexander Miehlke
6  buttonCalibrate.text.start=Starte Kalibrierung <F1>
7  buttonCalibrate.text.grabBackground=Hintergrund aufnehmen <
       F1>
8  buttonCalibrate.text.grabSkinColor=Hautfarbe aufnehmen <F1>
9  buttonCalibrateSmaller.toolTipText=Markierung verkleinern
10 buttonCalibrateBigger.toolTipText=Markierung vergrößern
```

Sollte die *config.properties* fehlen, gibt das System eine Warnmeldung aus, arbeitet dann aber mit im Quelltext hartkodierten Voreinstellungen weiter.

Benutzeroberfläche

Dieser Abschnitt bestimmt die Voreinstellungen für die Kontrollelemente, an denen Eingaben möglich sind, wie z.B. Checkboxen, Radioboxen oder Schieberegler:

```
#
# GUI settings
#

radioButtonMouse.selected=true
radioButtonJoystick.selected=false

sliderMouseSmooth.minimum=0
sliderMouseSmooth.maximum=10
sliderMouseSmooth.value=5

sliderJoystickAcceleration.minimum=10
sliderJoystickAcceleration.maximum=50
sliderJoystickAcceleration.value=20
```

Die letzten beiden Blöcke beziehen sich auf die Schieberegler für die Glättung der Cursorposition im Maus-Modus und die Beschleunigung des Cursors im Joystick-Modus. Dabei gibt `minimum` den kleinsten und `maximum` den größten auswählbaren Wert des Schiebereglers an. `value` bestimmt die Voreinstellung des Schiebereglers.

Kameraeinstellungen

Dieser Abschnitt bestimmt einige Parameter der Kamera:

```
#
# capture settings
#

monitorSize=160x120
captureSize=320x240
detectionSize=160x120
fingerTipROISize=40x40
framerate=30
```

monitorSize Diese Einstellung bestimmt die Größe des Videobildes in der Benutzeroberfläche.

captureSize Gibt an, mit welcher Auflösung das Videobild von der Kamera erfasst wird. Die Auflösung muss von der Kamera unterstützt werden.

detectionSize Gibt an, auf welche Auflösung das Videobild für die Fingerspitzenerkennung unterabgetastet werden soll. Entspricht die Auflösung der von `captureSize`, wird keine Unterabtastung vorgenommen.

fingerTipROISize Falls das Videobild unterabgetastet wurde, bestimmt dieser Parameter, wie groß der Ausschnitt im Original-Videobild ist, in dem eine detaillierte Fingerspitzenerkennung durchgeführt wird.

framerate Bestimmt, wie oft die Kamera in der Sekunde ein Videobild liefern soll. Die Framerate muss von der Kamera unterstützt werden.

Bestimmung der Cursor-Position

Dieser Abschnitt definiert das Verhalten bei der Bestimmung der Cursor-Position:

```
#
# position detection
#

joystickIdleSize=10
actionRect=10,5,80,50
actionBorder=20
minScreenSpacing=10
```

joystickIdleSize Wird im Joystick-Modus die Fingerspitze im sogenannten Ruhebereich gehalten, so bewegt sich der Cursor nicht. Erst wenn die Fingerspitze aus dem Ruhebereich heraus bewegt wird, bewegt sich der Cursor in die jeweilige Richtung (siehe auch Abbildung 4.16 auf Seite 63). Der Parameter `joystickIdleSize` bestimmt nun die Größe des Ruhebereichs prozentual zur Größe des Videofensters. Bei einer Auflösung von 320 mal 240 Bildpunkten und einem Wert von 10 hat der Ruhebereich eine Größe von 32 mal 24 Bildpunkten und wird (abhängig von `actionRect`) zentriert.

actionRect Bewegt sich die Fingerspitze zu weit in den Randbereich des Videobildes, so wird die Hand nur noch zum Teil erfasst. Die Fingerspitzenerkennung kann aber nur mit einer komplett sichtbaren Hand erfolgen. Die Angabe von `actionRect` erfolgt ebenfalls prozentual zur Größe des Videofensters. Der Wert 10 bei einer Breite von 320 Bildpunkten gibt an, dass der Finger im Maus-Modus bereits bei 32 Bildpunkten den linken Rand erreicht hat.

minScreenSpacing Der Wert 10 gibt an, dass der Cursor stets einen Mindestabstand von 10 Bildpunkten vom Bildschirmrand einhält. So wird das Navigieren im Randbereich erleichtert und verhindert, dass der Cursor am rechten oder unteren Bildschirmrand nicht mehr sichtbar ist.

Erkennung von Mausklicks

Der Abschnitt `click detection` enthält die Einstellungen zur Erkennung von Mausklicks:

```
#
# click detection
#

clickRectSize=20
fingerCountSmooth=5

sequenceLeftClick.0=1,0,0
sequenceLeftClick.1=2,200,1000
sequenceLeftClick.2=1,0,0

sequenceRightClick.0=2,0,0
sequenceRightClick.1=3,200,1000
sequenceRightClick.2=2,0,0
```

clickRectSize Der Wert von `clickRectSize` bestimmt die Größe des Klickbereichs in Bildpunkten (siehe 4.8.3 auf Seite 68). Je größer der Klickbereich, desto mehr kann sich die Position des Zeigefingers während der Geste für den Mausklick verändern.

fingerCountSmooth Dieser Eintrag bestimmt, wie die Anzahl der erkannten Finger geglättet wird. Bei einer sehr instabilen Kontur kann es zu unerwünschten Fehlern bei der Fingerzählung kommen. Mit der Glättung wird in diesem Fall ein Mittelwert über 5 Fingerzählungen berechnet. So werden kurzzeitige Ausreißer unterdrückt.

sequenceLeftClick und sequenceRightClick Für die Betätigung des linken und rechten Mausklicks können hier die entsprechenden Handgesten angegeben werden. Für den linken Mausklick sind es drei Gesten, bei dem zunächst ein Finger gestreckt ist. Die Mindest- und Höchstdauer, wie lange er gestreckt ist, spielt keine Rolle und ist mit 0 angegeben. Sie kommen erst zum Tragen, wenn ein zweiter Finger gestreckt wird. In diesem Fall müssen die zwei Finger mindestens 200 und höchstens 1000 Millisekunden ausgestreckt bleiben. Danach wird einer der beiden Finger wieder gebeugt. Ähnlich verhält es sich bei der Geste für den rechten Mausklick, nur dass sich hier die Anzahl der gestreckten Finger von zwei nach drei und wieder nach zwei verändert.

Handgesten

Dieser Abschnitt enthält die Handgesten, auf die das System reagieren soll:

```
#
# gestures
#

gesture.0=LDR,calc.exe
gesture.1=URD,notepad.exe
gesturesMinMove=10
```

gesture.x Diese Liste enthält die Bewegungsrichtungen der Handgesten und die auszuführenden Programme. Als Bewegungsrichtung kann eine Kombination aus `L` (links), `R` (rechts), `U` (oben) und `D` (unten) angegeben werden. Das Programm `calc.exe` (Windows-Taschenrechner) wird z.B. bei einer Geste ausgeführt, die dem Buchstaben C ähnelt.

gesturesMinMove Diese Einstellung bestimmt, wie weit sich die Hand im Videobild bewegen muss, damit eine Geste erkannt wird. Der Wert 10 bei einer Videoauflösung von 320 mal 240 Bildpunkten gibt an, dass die Hand 32 Bildpunkte überspringen muss.

Bilder und Klänge

In diesen beiden Abschnitten sind die Bilder und Klänge für die Benutzeroberfläche aufgelistet:

```
#
# images
#

imageBigger=images/bigger.gif
imageSmaller=images/smaller.gif
imageExpand=images/expandskin.gif

#
# sounds
#

soundLeftClick=sounds/leftclick.wav
soundRightClick=sounds/rightclick.wav
```

5.4 Klassen

Dieser Abschnitt verschafft einen Überblick über einige der erstellten Java-Klassen. Neben einer kurzen Beschreibung der Klasse werden auch wichtige Code-Ausschnitte abgebildet. Die Code-Ausschnitte sind teilweise gekürzt, um sich auf das Wesentliche zu konzentrieren. Im Anhang ab Seite 105 sind die dazugehörigen Klassendiagramme abgedruckt. Die Details zu allen Klassen sind im JavaDoc auf der beiliegenden CD ersichtlich.

5.4.1 Calibration

Die Klasse Calibration implementiert die Kalibrierung des Systems auf den Hintergrund und die Hautfarbe des Anwenders. Dabei wird ein Frame des Videobildes an den Konstruktor der Klasse übergeben:

```
1 public Calibration(int[] pixels, Dimension size, int
     circleSize)
```

Neben dem Videoframe `pixels` und seinen Abmessungen `size` ist der Durchmesser des auszuwertenden kreisförmigen Bereichs `circleSize` anzugeben. Die Farbwerte für die Kalibrierung werden aus diesem Bereich entnommen und mit Hilfe der Methode `Color.RGBtoHSB()` in den HSB-Farbraum umgerechnet (siehe auch 2.1.3):

```
1 int[] rgb=ImageUtils.int2rgb(pixels[i]);
2 float[] hsb=Color.RGBtoHSB(rgb[0], rgb[1], rgb[2], null);
3 double hue=hsb[0];
4 double sat=hsb[1];
5 double bri=hsb[2];
```

Die Werte `hue` (Farbwert) und `sat` (Farbsättigung) werden auf einen Bereich von 0 bis 255 abgebildet. Dabei wird die Helligkeit `bri` nicht benutzt.
Die Lookup-Tabelle `lookupTableForeground` wird an der aus `hueInt` und `satInt` errechneten Position gesetzt:

```
1 int hueInt=(int) (hue*(float) tableSize); // tableSize=255
2 int satInt=tableSize-(int) (sat*(float) tableSize);
3 lookupTableForeground[hueInt+(satInt*(tableSize+1))]=
     ImageUtils.FOREGROUND_BYTE;
```

Mit dem aufgenommenen Hintergrundbild (siehe 4.4.2) wird analog verfahren. Die Farbwerte des Hintergrundbildes werden mit der Setter-Methode `setBackgroundPixels(int [] backgroundPixels)` an die Klasse übergeben und in einer zweiten Lookup-Tabelle `lookupTableBackground` abgebildet. Überschneidende Bereiche in beiden Tabellen werden anschließend in `lookupTableForeground` entfernt. Über die Methode `inLookupTable()` kann nun einfach ermittelt werden, ob der übergebene RGB-Farbwert `color` hautfarben

ist oder nicht. Die Methode ermittelt wie oben Farbton und -Sättigung, berechnet die Position in der Lookup-Tabelle `lookupTableForeground` und prüft, ob die Tabelle an dieser Stelle gesetzt ist:

```
1 public boolean inLookupTable(int color) {
2   int[] rgb=ImageUtils.int2rgb(color);
3   float[] hsb=Color.RGBtoHSB(rgb[0], rgb[1], rgb[2], null);
4   int hueInt=(int) (hsb[0]*(float) tableSize);
5   int satInt=tableSize-(int) (hsb[1]*(float) tableSize);
6   return (lookupTableForeground[hueInt+(satInt*(tableSize+1)
       )]==ImageUtils.FOREGROUND_RGB_INT);
7 }
```

5.4.2 FingerDetection

Grundlage für die Positionsbestimmung und die Erkennung des Mausklicks stellt die Erkennung der Fingerspitzen dar. Die Klasse FingerDetection ermittelt diese Daten aus der Handkontur. Dabei greift sie auf die Klasse ContourTracer [BB06] zurück, die die benötigte Kontur aus einem Binärbild erstellt.

Die Zählung der gestreckten Finger implementiert die Methode `getFingerTips()`, die die Positionen aller Fingerspitzen in einer Liste zurückliefert. Beginnend mit dem untersten Konturpunkt werden alle Konturpunkte durchlaufen und die polaren Distanzen `distance` berechnet. Durch Einbeziehung der kleinsten Entfernung `minPolarDistance` wird die relative polare Distanz `relDist` ermittelt. Diese wird von der im vorherigen Iterationsschritt ermittelten relativen polaren Distanz `oldRelDist` subtrahiert. Man erhält die Steigung bzw. den Gradienten `gradient`. Durch Vergleich mit dem vorangegangenen Gradienten `oldGradient` werden die Vorzeichenwechsel der Steigungen sichtbar, aus der sich die Minima und Maxima ableiten lassen.

```
1 node=(Node) contour.getNodes().get(i);
2 distance=node.polarDistance(getPalmCenter());
3 oldRelDist=relDist;
4 relDist=(distance-minPolarDistance)/minPolarDistance;
5 oldGradient=gradient;
6 gradient=relDist-oldRelDist;
7 if (oldGradient>=0.0 && gradient<0.0) {
8   maximaNodes.add(node);
9 }
10 else if (oldGradient<0.0 && gradient>=0.0) {
11   minimaNodes.add(node);
12 }
```

In einem weiteren Schritt werden alle Maxima gefiltert. Dazu werden die Fingerlängen `fingerLengthBefore` und `fingerLengthAfter` über die Distanzen zwischen dem Maxi-

mum und seinen zwei umliegenden Minima errechnet. Über die Methode `openingAngle` wird der Öffnungswinkel ermittelt (siehe auch 4.5.4). Nur wenn beide Fingerlängen einen Schwellenwert überschreiten und der Öffnungswinkel kleiner als `maxOpeningAngle` (in diesem Fall 50 Grad) ist, wird das Maximum in die Liste der erkannten Fingerspitzen kopiert. Der Schwellenwert wird durch die kleinste polare Distanz `minPolarDistance` und einer prozentualen Angabe `fingerLength` bestimmt.

```
for (i=0; i<maximaNodes.size(); i++) {
  Node maximaNode=maximaNodes.get(i);
  Node minimaNodeBefore=minimaNodes.get(i);
  Node minimaNodeAfter=minimaNodes.get(i+1);
  double fingerLengthBefore=maximaNode.polarDistance(
     minimaNodeBefore);
  double fingerLengthAfter=maximaNode.polarDistance(
     minimaNodeAfter);

  double threshold=minPolarDistance*fingerLength/100;
  double openingAngle=openingAngle(maximaNode,
     minimaNodeBefore, minimaNodeAfter);

  if (fingerLengthBefore>=threshold &&
    fingerLengthAfter>=threshold &&
    openingAngle<=maxOpeningAngle) {
    fingerTips.add(maximaNode);
  }
}
```

5.4.3 FingerClick

Die Klasse FingerClick verwaltet in der Hauptsache eine Liste, in der die Anzahl der erkannten Finger und die Dauer abgelegt werden. Als Listenelement wird die Unterklasse `FingerCountEntry` verwendet:

```
private class FingerCountEntry {
  public int fingerCount;
  public long duration;
  public Point position;
}
```

Das Attribut `position` gibt zusätzlich an, an welcher Position sich die Fingerspitze des Zeigefingers im Videobild zum Zeitpunkt der Fingerzählung befand. Die Anzahl der durch die Klasse FingerDetection ermittelten Finger und die Position wird bei jedem Videoframe an die Methode `addFingerCount()` übergeben:

```
1  public void addFingerCount(int fingerCount, Point position)
       {
2    if (fingerCount!=this.oldFingerCount) {
3      if (list.size()>0) {
4        GregorianCalendar currentTime=new GregorianCalendar();
5        long duration=currentTime.getTimeInMillis() - this.
             startTime.getTimeInMillis();
6        if (duration<minDuration) {
7          list.remove(list.size()-1);
8        } else {
9          list.get(list.size()-1).duration=duration;
10       }
11     }
12     list.add(new FingerCountEntry(fingerCount,-1,position));
13     if (list.size()>maxListSize) { // maxListSize=7
14       list.remove(0);
15     }
16     this.startTime=new GregorianCalendar();
17   }
18   this.oldFingerCount=fingerCount;
19 }
```

Sobald sich die Anzahl der Finger verändert, wird die zeitliche Differenz `duration` zum letzten Listenelement in der Liste `list` errechnet und im letzten Listenelement abgelegt. Ist jedoch die Dauer zu gering, wird das letzte Listenelement verworfen. Übersteigt die Länge der Liste den Wert `maxListSize`, wird das erste Listenelement gelöscht. Anderenfalls würde die Liste den Speicher der virtuellen Maschine sprengen.

Ob nun der Anwender die Geste für einen Mausklick durchgeführt hat, lässt sich mit Hilfe der Methode `isClicked()` feststellen:

```
1  public boolean isClicked(ArrayList<SequenceEntry> sequence)
       {
2    int sequenceSize=sequence.size();
3    int listSize=list.size();
4    if (sequenceSize<=listSize) {
5      int count=0;
6      for (int i=listSize-sequenceSize; i<listSize; i++) {
7        FingerCountEntry fingerCountEntry=list.get(i);
8        int offset=i-listSize+sequenceSize;
9        SequenceEntry sequenceEntry=sequence.get(offset);
10       if (sequenceEntry.fingerCount==fingerCountEntry.
             fingerCount) {
11         if (offset>0 && offset<sequenceSize-1) {
12           if (fingerCountEntry.duration>=sequenceEntry.
                minDuration &&
```

```
13              fingerCountEntry.duration<=sequenceEntry.
                   maxDuration) {
14              count++;
15            }
16          } else {
17            count++;
18          }
19        }
20      }
21      if (count==sequenceSize) {
22        return true;
23      }
24    }
25    return false;
26  }
```

Der Methode wird eine Liste von Einträgen der Klasse `SequenceEntry` übergeben, die die Fingeranzahl, die Mindest- und Höchstdauer einer Handgeste beinhalten:

```
1 public class SequenceEntry {
2   public int fingerCount;
3   public long minDuration;
4   public long maxDuration;
5 }
```

Für den linken Mausklick hat die Liste die drei Elemente {{1,0,0},{2,200,1000},{1,0,0}}, da sich die Fingeranzahl von 1 auf 2 und wieder auf 1 verändert. Damit der Mausklick erkannt wird, müssen die zwei Finger mindestens 200 und höchstens 1000 Millisekunden gestreckt sein.

Die Methode `isClicked()` prüft nun, ob die letzten drei Einträge der Listen in Bezug auf Fingeranzahl, Mindest- und Maximaldauer übereinstimmen. Zusätzlich wird der Klickbereich geprüft, d.h. ob sich die Position des Zeigefingers während des Klickvorganges nicht wesentlich verändert hat. Aus Gründen der Übersichtlichkeit wurde der Quellcode für die Prüfung des Klickbereichs nicht mit abgedruckt.

5.4.4 GesturesDetection

Diese Klasse implementiert die Erkennung der Handgesten. Sie verwaltet unter anderem eine Liste, in der alle erkannten Gesten enthalten sind. Die Listenelemente sind vom Typ `GestureEntry`:

```
1 public class GestureEntry {
2   public Node start;
3   public Node end;
```

```
4   public String move;
5   public long time;
6 }
```

Die Attribute `start` und `end` speichern die Start- und Endkoordinaten der Geste. `move` kann die Werte L (links), R (rechts), U (oben) und D (unten) annehmen und steht für die Bewegungsrichtung. `time` enthält die Zeit in Millisekunden, an der die Geste ausgeführt wurde.

Erkennt das System, dass der Anwender zwei Finger in die Kamera hält, wird in jedem Frame die Methode `addPosition()` aufgerufen und die Position der ersten Fingerspitze übergeben:

```
1 public void addPosition(Node fingerTip) {
2
3   int xDiff = fingerTip.x-oldFingerTip.x;
4   int yDiff = fingerTip.y-oldFingerTip.y;
5   int xDiffAbs = Math.abs(xDiff);
6   int yDiffAbs = Math.abs(yDiff);
7
8   if (xDiffAbs>=minMove || yDiffAbs>=minMove) {
9       float ratio =((float) xDiffAbs)/yDiffAbs;
10      if (ratio<1) {
11          if (yDiff<0) addMove(oldFingerTip,fingerTip,UP);
12          else addMove(oldFingerTip,fingerTip,DOWN);
13      } else {
14          if (xDiff<0) addMove(oldFingerTip,fingerTip,RIGHT)
                ;
15          else addMove(oldFingerTip,fingerTip,LEFT);
16      }
17      oldFingerTip = fingerTip;
18    }
19 }
```

Im ersten Codeblock werden die horizontale und vertikale Bewegungsänderung berechnet und in `xDiff` und `yDiff` abgelegt. Die Variable `oldFingerTip` gibt dabei die vorherige Position der Fingerspitze an. Sofern die absoluten Bewegungsänderungen `xDiffAbs` bzw. `yDiffAbs` über dem Schwellenwert `minMove` liegen, wird das Verhältnis zwischen diesen beiden Werten berechnet. Überwiegt die vertikale Bewegung, wird der Liste neben dem Start- und Endpunkt der Wert `UP` bzw. `DOWN` hinzugefügt. Überwiegt die horizontale Bewegung, wird der Wert `LEFT` bzw. `RIGHT` hinzugefügt.

Um nun festzustellen, ob eine bestimmte Handgeste, wie z.B. „LDR“ ausgeführt wurde, wird die Methode `hasGesture()` benutzt:

```
1 public boolean hasGesture(String gesture) {
```

```
2
3   String g=this.toGestureString();
4
5   if (g!=null && g.length()>=gesture.length()) {
6     if (g.substring(this.size()-gesture.length(), this.size
        ()).equals(gesture)) {
7       return true;
8     }
9   }
10  return false;
11 }
```

Über die Methode `toGestureString()` werden zunächst alle Bewegungen der verwalteten Liste in einen String kopiert. Sofern das Ende des Strings mit dem übergebenen String übereinstimmt, wurde die Handgeste ausgeführt.

5.4.5 FingerTracker

Diese Klasse stellt den Hauptteil des Systems dar. Neben der grafischen Benutzeroberfläche werden in der Klasse FingerTracker alle bereits behandelten Klassen verwendet und zusammengeführt. Dieser Abschnitt befasst sich mit der Implementierung der bereits in 4.3 vorgestellten Verarbeitungskette. Das laufende Videobild wird in mehreren Stufen gefiltert und ausgewertet, bis schließlich die Position des Zeigefingers und die Anzahl der gestreckten Finger berechnet werden.

Die folgenden Code-Zeilen enthalten Ausschnitte aus der Methode `processFrame()`, die sämtliche Verarbeitungsschritte nacheinander ausführt:

```
1 public void processFrame(Buffer buffer) {
2   createPixels((byte[]) buffer.getData(), pixelsHR,
      captureSize.width, captureSize.height);
3     ColorProcessor cpCapture=new ColorProcessor(captureSize.
        width, captureSize.height,pixelsHR);
4     ColorProcessor cpDetection=cpCapture;
```

Die Methode `processFrame()` erhält vom Java Media Framework (siehe 2.2) die Rohdaten `buffer` des aktuellen Kamerabildes, die über die Methode `createPixels()` in eine Liste von RGB-Werten `pixelsHR` umgewandelt werden. Zusätzlich wird ein ImageJ-Objekt `ColorProcessor` erstellt.

```
1   if (captureSize.width!=detectionSize.width) {
2     cpDetection=(ColorProcessor) cpCapture.resize(
        detectionSize.width,detectionSize.height);
3     pixelsLR=(int[]) cpDetection.getPixels();
4   }
```

Falls die Auflösung der Kamera nicht mit der für die Verarbeitung übereinstimmt, wird über die Methode `resize()` eine Unterabtastung vorgenommen.

```
1  if (calibration!=null && calibration.isCalibrationDone())
       {
2    byte[] outpixByte=new byte[detectionSize.width*
       detectionSize.height];
3    skinColorFilter(pixelsLR, outpixByte);
```

Sofern die Kalibrierung des Systems auf die Hautfarbe und den Hintergrund vorgenommen wurde, wird nun der Hautfilter über die Methode `skinColorFilter()` angewandt. Aus dem RGB-Bild ensteht ein Binärbild, welches die hautfarbenen Regionen beschreibt und in der Liste `outpixByte` abgelegt wird.

```
1    ByteProcessor bp=new ByteProcessor(detectionSize.width,
       detectionSize.height, outpixByte, dcm);
2    for (int i=0; i<closingCount; i++) bp.dilate(1,255);
3    for (int i=0; i<closingCount; i++) bp.erode(1,255);
```

Zur Glättung der Regionen wird mehrfach eine morphologische Schließung (siehe 2.1.2) durchgeführt. Die Variable `closingCount` gibt dabei die Anzahl der Schließungen an. Da die morphologischen Operationen relativ viel Zeit beanspruchen, werden die Klasse `ByteProcessor` und die Methoden `dilate()` sowie `erode()` von ImageJ verwendet. Gegenüber einer zunächst selbst vorgenommenen Implementierung konnte mit ImageJ ein deutlicher Geschwindigkeitszuwachs erreicht werden.

```
1    ContourTracer contourTracer=new ContourTracer(outpixByte
       , detectionSize.width, detectionSize.height,
       ImageUtils.BACKGROUND_BYTE);
2    Contour largestContour=contourTracer.getContours().
       largestContour(minContourPoints);
```

Mithilfe der Klasse `ContourTracer` wird nun eine Konturverfolgung durchgeführt und die Kontur mit der größten Fläche ausgewählt.

```
1    fingerDetection=new FingerDetection(largestContour);
2    Node foreFinger=fingerDetection.getMaxPolarDistanceNode
       ();
3    int fingerCount=fingerDetection.getFingertips().size();
4    if (captureSize.width!=detectionSize.width) {
5      foreFinger=detectFingerTipHR(foreFinger, cpCapture);
6    }
```

Nun kann die Fingerspitzenerkennung vorgenommen werden. Sie berechnet die Positi-

on des längsten Fingers `foreFinger` und die Anzahl der Finger `fingerCount`. Sofern eine Unterabtastung vorgenommen wurde, wird nun nochmals eine detaillierte Fingerspitzenerkennung in einem kleinen Ausschnitt des Originalbildes `cpCapture` mithilfe der Methode `detectFingerTipHR()` durchgeführt.

```
1    if (radioButtonMouse.isSelected()) moveMouse(fingerCount
        , foreFinger);
2    if (radioButtonJoystick.isSelected()) moveJoystick(
        fingerCount, foreFinger);
3    clickDetection(fingerCount, foreFinger);
4    gesturesDetection(fingerCount, foreFinger);
```

Je nach Einstellung wird nun der Mauscursor über den Maus- oder Joystick-Modus bewegt. Die Methode `clickDetection()` wertet die sich verändernde Anzahl von gestreckten Fingern aus und betätigt eine der Maustasten oder bewegt das Mausrad. Die Methode `gesturesDetection()` erkennt eine ggf. vorgenommene komplexe Handgeste und startet die entsprechende Anwendung.

5.4.6 Robot

Java stellt eine sehr einfach zu handhabende Klasse namens Robot zur Verfügung, die es ermöglicht, den Mauszeiger zu bewegen und Mausklicks auszulösen.

Positionssteuerung der Maus

Mit einem einzelnen Methodenaufruf lässt sich die Position des Mauszeigers verändern. x und y geben dabei die neue Position an:

```
1  new Robot().mouseMove(x,y);
```

Auslösen von Mausklicks

Der folgende Code-Ausschnitt demonstriert einen einfachen Mausklick mit der linken Maustaste.

```
1  Robot robot=new Robot();
2  robot.mousePress(InputEvent.BUTTON1_MASK);
3  robot.mouseRelease(InputEvent.BUTTON1_MASK);
```

Dabei ist es nicht notwendig, das Ziel des Mausklicks, also z.B. ein Handle[1], anzugeben. Der Mausklick wird an der aktuellen Cursorposition ausgeführt und an das Kontrollelement gesendet, welches sich zu diesem Zeitpunkt unter dem Mauscursor befindet.

Die folgenden zwei Code-Ausschnitte lösen einen Doppelklick und eine Bewegung des Mausrads nach unten aus:

```
1  robot.mousePress(InputEvent.BUTTON1_MASK);
2  robot.mouseRelease(InputEvent.BUTTON1_MASK);
3  robot.delay(5);
4  robot.mousePress(InputEvent.BUTTON1_MASK);
5  robot.mouseRelease(InputEvent.BUTTON1_MASK);
```

```
1  robot.mouseWheel(1);
```

[1] engl.: **handle** Griff, Henkel, identifiziert unter Windows ein Kontrollelement wie z.B. Fenster, Schaltfläche oder Bildlaufleiste

6 Test

Das System wurde während der Implementierung und nach Fertigstellung mehreren Tests unterzogen. In diesem Kapitel werden die Ergebnisse des Hautfilters, der Konturfindung und der Fingerspitzenerkennung beschrieben und bewertet. Abschließend wird die Geschwindigkeit des Systems bei verschiedenen Kameraauflösungen gemessen.

6.1 Hautfilter

Dieser Abschnitt zeigt, wie der Hautfilter auf verschiedene Kleidung und Hintergründe reagiert. Die Abbildungen in 6.1 zeigen jeweils auf der linken Seite das Videobild und auf der rechten Seite das Binärbild als Ergebnis des Hautfilters. Schwarze Bereiche zeigen dabei Hautfarbe an.

Im Kapitel Entwurf unter 4.2.4 wurde bereits darauf hingewiesen, dass nicht jede Kleidung geeignet ist. In der Abbildung 6.1(a) wird ein brauner Pullover getragen. Die Farben des Pullovers kommen auch in der menschlichen Hautfarbe vor, so dass sie vom Hautfilter falsch zugeordnet werden. Im Ergebnis verschmilzt die Hand mit dem Ärmel des Pullovers zu einem Objekt.

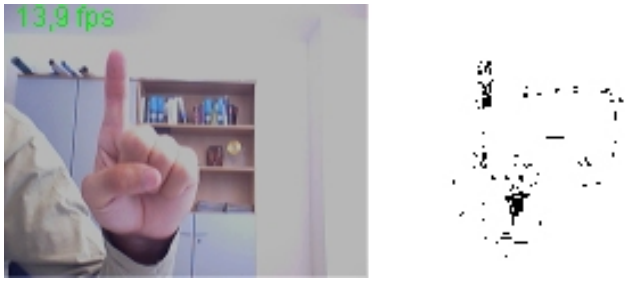
Gleichermaßen abhängig sind die Ergebnisse des Hautfilters vom Hintergrund. Die Abbildung 6.1(b) enthält im Hintergrund ein Holzregal. Die Farbe des Holzes ähnelt der Hautfarbe und wird als solche identifiziert. Der Hautfilter wurde bei dieser Abbildung so konfiguriert, dass gemeinsame Farben des Vordergrundes (Hand) und Hintergrundes (Schrank) nicht vom Vordergrund entfernt werden (siehe auch 4.4.3). So ist die komplette Hand erkennbar. Würde man die Hand etwas weiter Richtung Schrank bewegen, würde sie mit ihm zu einem Objekt verschmelzen.

Bei der folgenden Abbildung 6.1(c) wurde wieder die Standardkonfiguration des Hautfilters verwendet, so dass die gemeinsamen Farben vom Vordergrund entfernt wurden. Die Struktur der Hand wird dadurch nun leider unkenntlich. Tests haben ergeben, dass ca. 15 Prozent Hautfarben im Hintergrund tolerierbar sind. Bei über 15 bis 30 Prozent „franzt“ die Handkontur zwar aus, ist aber noch als Hand erkennbar. Bei über 30 Prozent bleibt von der Hand nicht mehr viel übrig, die nachfolgende Verarbeitung, wie z.B. die Fingererkennung, liefert vollkommen falsche Ergebnisse.

Bei der Abbildung 6.1(d) wurde der Holzschrank abgedeckt. Auch trägt der Anwender geeignete Kleidung. Die Hautfarbe wurde einwandfrei erkannt. Im linken oberen Bereich der Abbildung hat der Hautfilter den nicht abgedeckten schmalen Holzrahmen des Schrankes als Haut identifiziert, jedoch sind diese Fehler so gering, dass sie bei der weiteren Verarbeitung kaum eine Rolle spielen.

(a) Tragen eines braunen Pullovers

(b) Holzregal im Hintergrund



(c) Farben des Hintergrundes wurden entfernt

(d) Einwandfreies Ergebnis des Hautfilters

Abbildung 6.1: Ergebnisse des Hautfilters bei verschiedener Kleidung und Hintergründen

6.2 Konturfindung

Eine wichtige Rolle bei der Erkennung der Hautfarbe und somit die fehlerfreie Bestimmung der Handkontur spielt eine korrekte Beleuchtung. Die Abbildungen in 6.2 zeigen die Ergebnisse der Konturfindung bei verschiedenen Beleuchtungsverhältnissen.

Abbildung 6.2(a) wurde bei Abenddämmerung aufgenommen. Im Raum wurden alle Lichtquellen ausgeschaltet. Die Kamera hat sich durch Anhebung der Empfindlichkeit der Lichtsensoren gut an die geringe Beleuchtung angepasst, daher ist auch ein starkes Rauschen im Videobild sichtbar. Dennoch wurde die Handkontur relativ gut erkannt. Bessere Ergebnisse zeigt Abbildung 6.2(b). Hier wurde die Kontur bei zusätzlicher Aufstellung einer künstlichen Lichtquelle (in diesem Fall ein Deckenfluter) ohne Fehler erkannt. Abbildung 6.2(c) wurde in einem Raum ohne Tageslicht aufgenommen und zeigt ähnlich gute Ergebnisse. Dagegen ist die Kontur in den Abbildungen 6.2(d) und 6.2(e) unbrauchbar. Zu starke Überstrahlungen, verursacht durch starke Sonneneinstrahlung bzw. direktem künstlichen Licht, überdecken die Hautfarbe und lassen sie weiß erscheinen. Weiße Farbtöne kommen jedoch auch im Hintergrund vor, so dass sie kein Unterscheidungsmerkmal darstellen. An den fehlerhaften Konturen kann man erkennen, dass die gleißend hellen Bereiche nicht von der gelben Konturlinie umschlossen wurden.

6.3 Fingerspitzenerkennung

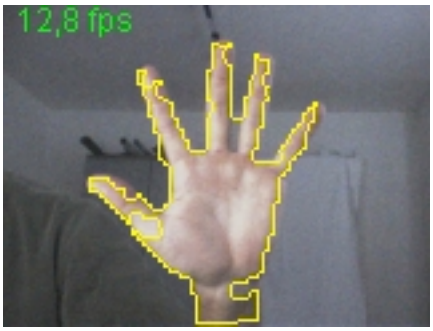
Die Abbildungen in 6.3 zeigen die Ergebnisse der Fingerspitzenerkennung bei einer verschiedenen Anzahl von Fingern. Die Konturen der Hände sind gelb umrandet. Die erkannten Fingerspitzen sind hellblau, der längste Finger rot markiert. In den Abbildungen ist zu erkennen, dass die Kontur nicht immer exakt an der Hand entlangläuft. Gerade bei Abbildung 6.3(e) beim Mittel- und kleinem Finger weist die Kontur Einbuchtungen auf, dennoch funktioniert die Fingererkennung durch eine implementierte Konturglättung zuverlässig.

In der Abbildung 6.3(d) könnte man vermuten, dass das Verfahren an dem gebeugten kleinen Finger (grüne Markierung) eine Fingerspitze erkennt, da sich dort ein lokales Maximum befindet. Das Verfahren hat jedoch dieses Maximum herausgefiltert, da der Öffnungswinkel zu groß ist (siehe 4.5.4).

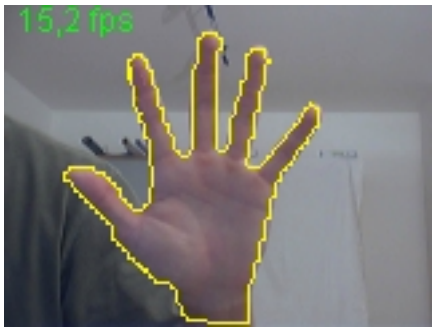
Die Abbildungen in 6.4 zeigen, wie stark das Verfahren von einer korrekten Kontur abhängig ist. In Abbildung 6.4(a) fehlt die Trennung zwischen Mittel- und Ringfinger, so dass beide Finger als nur einer erkannt werden. Dagegen erinnert die Kontur in Abbildung 6.4(b) nur noch wenig an eine Hand. Es wurden nur zwei Finger erkannt. Während die eine Position den Zeigefinger markiert, entstand die andere rein zufällig durch eine ungewollte Ausbuchtung der Kontur.

6.4 Geschwindigkeit

In diesem Abschnitt wird geprüft, ob die in den Anforderungen verlangte Echtzeitfähigkeit des Systems gegeben ist, d.h. ob sich der Cursor in angemessener Zeit ohne merkliche

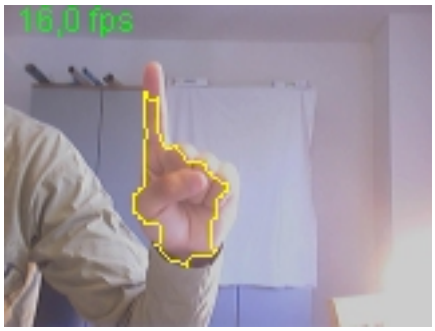


(a) Gute Konturfindung bei wenig Beleuchtung



(b) Sehr gute Konturfindung bei abgedunkeltem Raum und indirektem künstlichen Licht

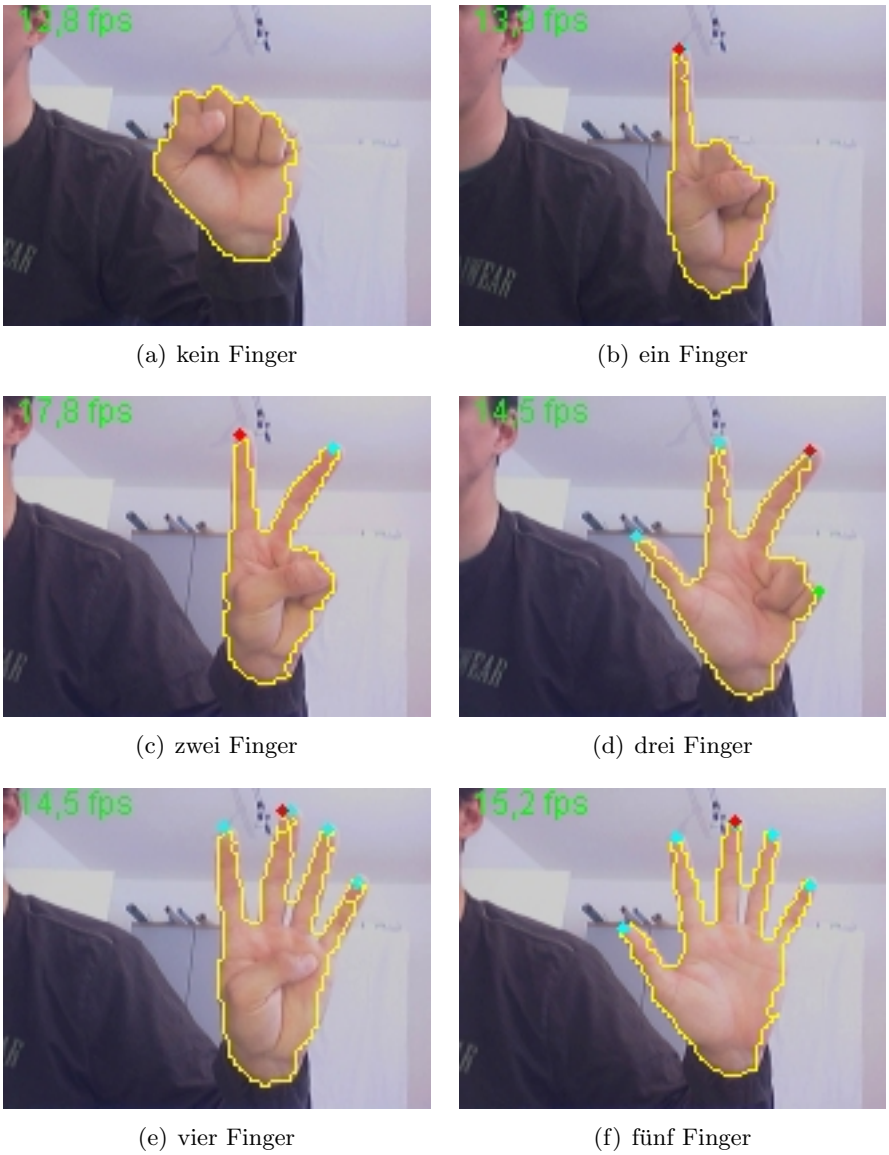
(c) Sehr gute Konturfindung bei ausschließlich indirektem künstlichen Licht



(d) Sehr schlechte Konturfindung bei starker Sonneneinstrahlung

(e) Sehr schlechte Konturfindung bei direktem künstlichen Licht

Abbildung 6.2: Konturfindung bei verschiedenen Beleuchtungssituationen

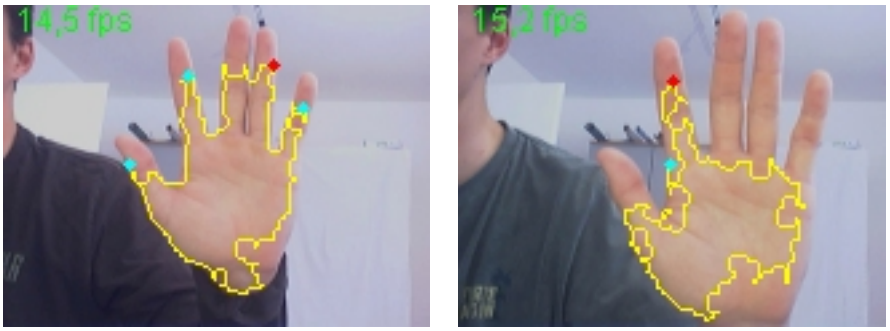


(a) kein Finger (b) ein Finger

(c) zwei Finger (d) drei Finger

(e) vier Finger (f) fünf Finger

Abbildung 6.3: Fingerspitzenerkennung



(a) vier Finger (b) zwei Finger

Abbildung 6.4: Fingerspitzenerkennung bei fehlerhafter Kontur

Verzögerung mit der Hand steuern lässt. Getestet wurde unter verschiedenen Kamera-Auflösungen auf einem Desktop-Computer und einem Laptop.

6.4.1 Testaufbau

Desktop-Computer

- Prozessor: Intel Pentium 4
- Taktfrequenz: 2,4 GHz
- Arbeitsspeicher: 1 GB DDR-RAM
- Betriebssystem: Microsoft Windows XP Professional

Laptop

- Typ: IBM Thinkpad X31
- Prozessor: Intel Pentium M
- Taktfrequenz: 1,5 GHz
- Arbeitsspeicher: 2 GB SO-DIMM
- Betriebssystem: Microsoft Windows XP Professional

Aktive Software

Auf beiden Computern liefen zur Zeit des Tests folgende Anwendungen im Hintergrund:

- Logitech Quickcam Steuerungssoftware
- Outpost Firewall
- Antivirenprogramm AntiVir

Webcam

- Hersteller: Logitech
- Modell: QuickCam Pro 3000
- Auflösungen: 640 mal 480, 320 mal 240 und 160 mal 120 Bildpunkte
- Framerate: 30 Bilder pro Sekunde
- Farbraum: RGB

Untersuchte Methode

Die Laufzeiten des Systems wurden anhand der Methode `processFrame()` (Klasse FingerTracker) gemessen. Diese Methode wird jedes Mal dann ausgeführt, wenn die Kamera einen neuen Frame bereitstellt und enthält alle Verarbeitungsschritte vom Einlesen eines Frames bis zur Positionierung des Mauszeigers.

Profiling-Werkzeuge

Die Messung der Verarbeitungszeit für einen Frame wurde mit Hilfe einer selbst implementierten Methode vorgenommen. Für den Vergleich der Rechenzeit der einzelnen Verarbeitungsschritte wurde das Eclipse-PlugIn *Test & Performance Tools Platform (TPTP)* verwendet.

Testverlauf

Auf beiden Computern wurde das System gestartet, eine Kalibrierung auf den Hintergrund und die Hautfarbe vorgenommen und der Maus-Modus aktiviert. Für ca. eine Minute wurde der Cursor auf dem Bildschirm mit der Hand bewegt und Mausklicks ausgeführt. Es wurden dabei keine weiteren Anwendungen gestartet. Aus den Messergebnissen wurden dann Durchschnittswerte berechnet.

6.4.2 Messwerte und Ergebnisse

In diesem Abschnitt wird zunächst die Geschwindigkeit der Verarbeitung eines Frames auf dem Desktop-Computer und dem Laptop gegenübergestellt. Dann wird die Rechenzeit der einzelnen Verarbeitungsschritte miteinander verglichen. Die einzelnen Messwerte werden anhand mehrerer Diagramme veranschaulicht. Zudem werden die Ergebnisse analysiert und die Ursachen erläutert.

Rechenzeit für einen Frame

Die Diagramme in der Abbildung 6.5 zeigen die Messergebnisse des Desktop-Computers und des Laptops. Auf der horizontalen Achse sind die verschiedenen Kamera-Auflösungen,

auf der vertikalen die Geschwindigkeit des Systems in Bildern pro Sekunde angetragen. Bei den linken drei Säulen erfolgte die Verarbeitung des Videobildes in der Auflösung, wie die Kamera sie liefert. Bei den rechten drei Säulen wurde eine Unterabtastung vorgenommen. Für jede Auflösung wurde über den Säulen der konkrete Messwert vermerkt.

Erwartungsgemäß sinkt die Framerate mit der Erhöhung der Kamera-Auflösung. Während bei der Auflösung von 160 mal 120 Bildpunkten die Geschwindigkeit mit 31 Bildern pro Sekunde der Framerate der Kamera entspricht, bricht die Framerate bei 320 mal 240 auf 14 (Desktop-Computer) bzw. 15 (Laptop) Bildern pro Sekunde ein. In der höchsten von der Kamera unterstützten Auflösung von 640 mal 480 Bildpunkten werden sogar nur noch 3 Bilder pro Sekunde erreicht.

Zu erklären ist dies mit der größeren Datenmenge, die bei höheren Auflösungen anfällt. Die Tabelle 6.1 zeigt, dass sich bei Verdopplung der Auflösung die Datenmenge vervierfacht. So muss das System bei 320 mal 240 Bildpunkten mit 6.750 Kilobyte pro Sekunde viermal soviel Daten verarbeiten wie bei 160 mal 120 Bildpunkten mit nur 1.688 Kilobyte. Bei 640 mal 480 Bildpunkten fallen mit 27.000 Kilobyte sogar 16 mal soviel Daten an.

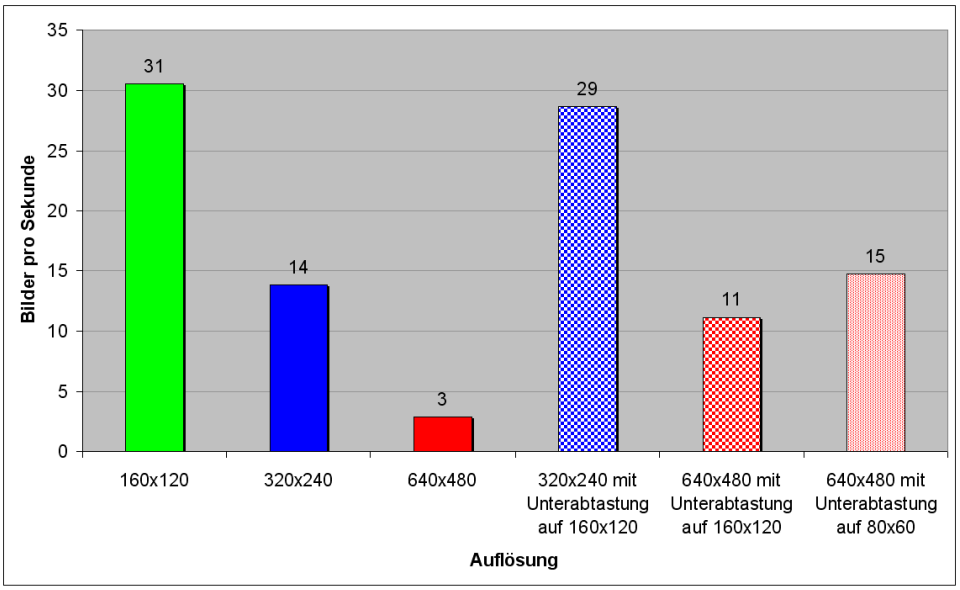
Video-Auflösung (Pixel)	Datenmenge pro Frame (Kilobyte)	Datenmenge pro Sekunde (Kilobyte)	Faktor
160 x 120	56	1.688	1
320 x 240	225	6.750	4
640 x 480	900	27.000	16

Tabelle 6.1: Anfallende Datenmengen bei verschiedenen Auflösungen (24 Bit Farbtiefe, 30 Bilder pro Sekunde)

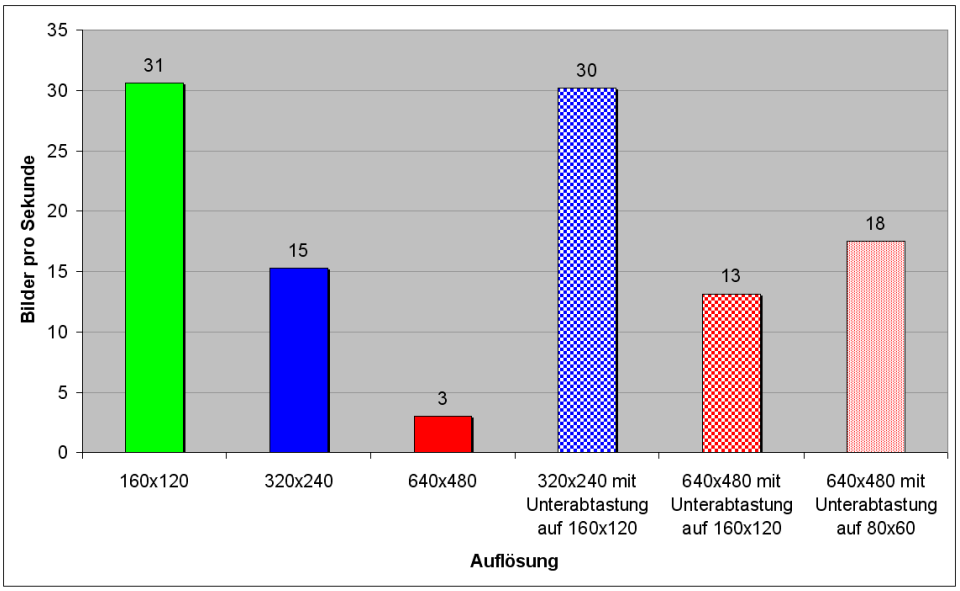
Die niedrigen Frameraten wirken sich natürlich drastisch auf die Bedienbarkeit des Systems aus. Bei 14 bzw. 3 Bildern pro Sekunde wird der Mauszeiger alles andere als flüssig über den Bildschirm bewegt. Die hohen Datenmengen können von den verwendeten Filtern und Methoden nicht in angemessener Zeit bewältigt werden.

Ein anderes Bild zeigt sich bei der Unterabtastung. Wird für die Verarbeitung die Auflösung von 320 mal 240 auf 160 mal 120 Bildpunkte reduziert, so ist die Framerate mit 29 bzw. 30 Bildern pro Sekunde nahezu identisch mit der Kamera. Der Mauszeiger bewegt sich flüssig und fast ohne Verzögerung über den Bildschirm. Wird die Kamera-Auflösung auf 640 mal 480 Bildpunkte erhöht und ebenso mit 160 mal 120 Bildpunkten unterabgetastet, so steigen die Frameraten mit 11 bzw. 13 Bildern pro Sekunde auf in etwa den vierfachen Wert gegenüber den Werten ohne Unterabtastung. Da die Werte für eine flüssige Bewegung des Mauszeigers zu gering waren, wurde testweise auf 80 mal 60 Bildpunkte unterabgetastet. Dadurch erhöhte sich die Framerate jedoch nur um 4 bzw. 5 Bilder pro Sekunde. Dazu kam, dass die Fingerspitzenerkennung bei einer solch groben Auflösung nicht mehr genügend Informationen erhält, um einwandfrei zu arbeiten.

Die unterschiedlichen Frameraten zwischen dem Desktop-Computer und dem Laptop sind zwar gering, der Laptop erreicht aber teilweise bis zu drei Frames mehr. Zurückzuführen ist dies auf den etwas leistungsfähigeren Prozessor des Laptops. Mobile Prozessoren erreichen



(a) Desktop-Computer



(b) Laptop

Abbildung 6.5: Rechenzeit für einen Frame

im Verhältnis zu ihrer meist niedrigen Taktfrequenz höhere Rechenleistungen als gleich getaktete Desktop-Computer.

Neben der Logitech QuickCam Pro 3000 wurde das Sytem auch mit zwei weiteren Webcams getestet. Die Webcams *Mustek GSmart Mini 2* und *Logitech QuickCam Pro 5000* lieferten jedoch nahezu identische Werte, so dass ihre Messwerte nicht mit abgedruckt wurden.

Rechenzeit einzelner Verarbeitungsschritte

Das Säulen-Diagramm in Abbildung 6.6 zeigt die Messergebnisse der einzelnen Verarbeitungsschritte von der Umwandlung der Kamera-Rohdaten nach RGB bis zur Fingerspitzenerkennung. Auf der horizontalen Achse sind die Verarbeitungsschritte, auf der vertikalen die Zeit in Millisekunden angetragen. Durch den durch TPTP verursachten Overhead sind die Zeiten nicht realistisch, lassen aber einen Vergleich untereinander zu. Das Kreis-Diagramm in Abbildung 6.7 stellt die durchschnittliche Verarbeitungszeit im Verhältnis zur Gesamtzeit in Prozent dar.

In beiden Diagrammen lässt sich ablesen, welche Verarbeitungsschritte am meisten Zeit benötigen. So benötigen der Hautfilter mit 40 bis 111 und die Schließung mit 32 bis 58 Millisekunden weit mehr Zeit als andere Verarbeitungsschritte. Beide Filter zusammen nehmen rund 65% der Gesamtzeit in Anspruch. Beide Filter arbeiten bildpunktorientiert, was sehr viele Iterationsschritte zur Folge hat.

Im Säulendiagramm ist der Vorteil der Unterabtastung gut zu erkennen. Während sich die Rechenzeit bei steigender Auflösung erhöht, bleibt sie durch die Unterabtastung in fast allen Verarbeitungsschritten im Bereich der Auflösung von 160 mal 120 Bildpunkten. Je weiter die Auflösung durch die Unterabtastung verringert wurde, umso größer ist die Zeitersparnis. Die Zeitersparnis kompensiert dabei den zusätzlichen Aufwand für die Unterabtastung und die Detail-Fingerspitzenerkennung.

Die Konturfindung, die Wahl der größten Kontur und die Fingerspitzenerkennung laufen relativ schnell ab. Zusammen belegen sie durchschnittlich 21% der Rechenzeit. Der Grund dafür ist, dass im Verhältnis zur Bildauflösung nur wenige Konturpunkte verarbeitet werden müssen. So hat eine ausgestreckte Hand in einem Videobild von 160 mal 120 Bildpunkten (=19.200 Bildpunkte) lediglich ca. 300 Konturpunkte.

Zusammenfassend lässt sich sagen, dass das System bis zur Auflösung von 320 mal 240 Bildpunkten schnell genug arbeitet. Bei der durch die Unterabtastung erreichte Framerate von 30 Bildern pro Sekunde lässt sich das System flüssig bedienen.

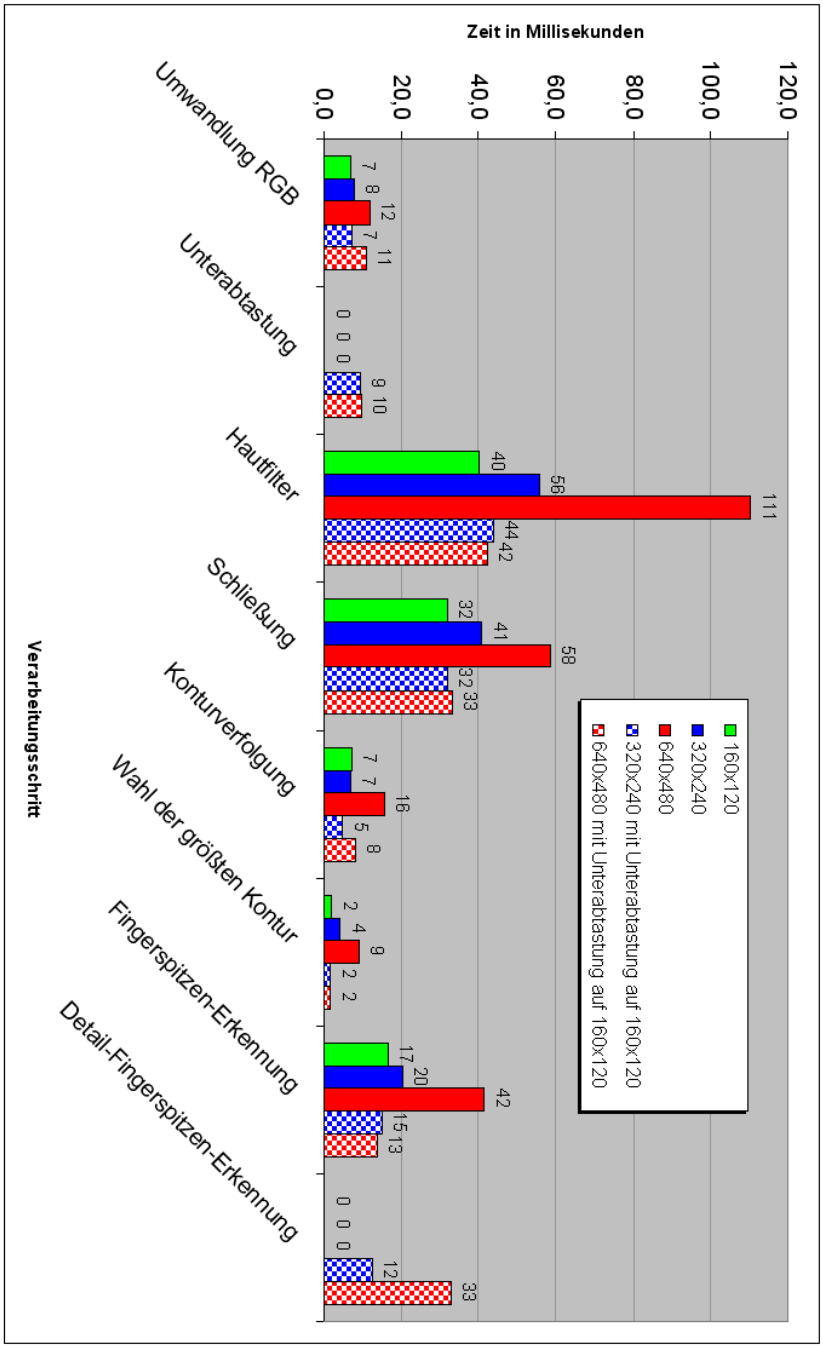


Abbildung 6.6: Rechenzeit einzelner Verarbeitungsschritte

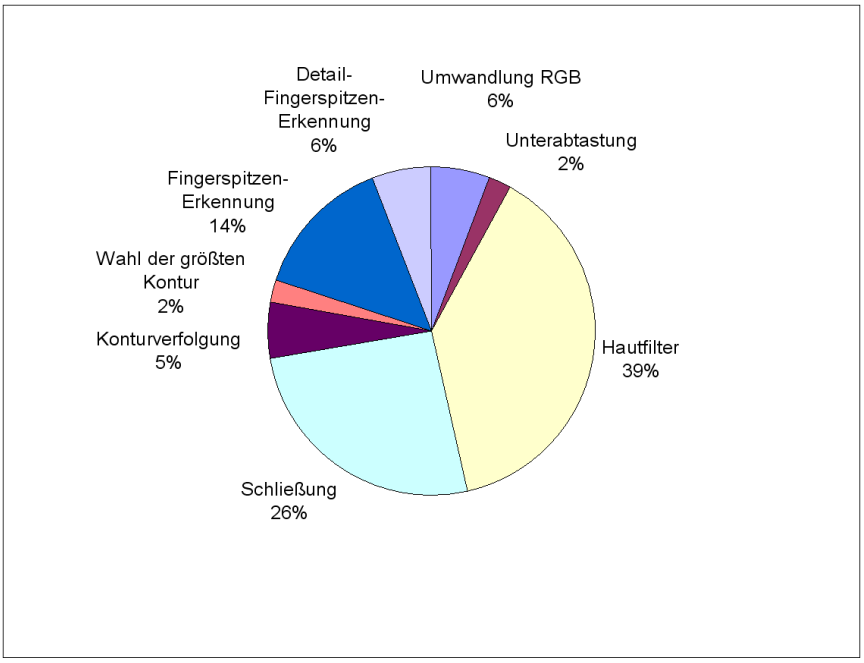


Abbildung 6.7: Anteil der Verarbeitungsschritte an der Rechenzeit für einen Frame

7 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse dieser Arbeit zusammen und nimmt eine Bewertung vor. Anschließend werden Vorschläge für eine Weiterentwicklung gemacht und ein Ausblick auf mögliche Eingabeformen der Zukunft gegeben.

7.1 Zusammenfassung

In dieser Arbeit wurde ein System entwickelt, mit der der Mauszeiger auf dem Bildschirm über die Hand gesteuert werden kann. Über den Zeigefinger wird die Position bestimmt. Alle Eingaben, die mit der Maus möglich sind, können über einfache Handgesten vorgenommen werden. So lässt sich die linke Maustaste durch das kurze Abspreizen des Daumens auslösen. Für einen Klick mit der rechten Maustaste wird dabei zusätzlich der Mittelfinger gestreckt. Für die Bedienung des Mausrads werden alle fünf Finger gestreckt und die Hand nach oben oder unten bewegt. Weitere Gesten in Form von bestimmten Handbewegungen ermöglichen das Ausführen beliebiger Anwendungen oder Aktionen.

Das System begnügt sich mit einem herkömmlichen Computer und einer einfachen Webcam. Die Beschaffung teurer Spezialhardware ist nicht notwendig. Der Anwender muss keinen Handschuh tragen oder spezielle Sensoren an der Hand befestigen, die bloße Hand reicht aus.

Dadurch, dass die Programmiersprache Java verwendet wurde, konnte das System sowohl unter Windows als auch unter Linux erfolgreich getestet werden. Dabei wird jede Software unterstützt, bei der eine Computermaus zur Eingabe benutzt werden kann, da das System die Position des Mauszeigers und den Mausklick über den Maustreiber an die Software schickt. Die Benutzeroberfläche zur Steuerung des Systems erfüllt die geforderten Grundsätze für die Dialoggestaltung.

Die Erkennungsgenauigkeit der Hand steht und fällt mit der Beleuchtung und der Wahl des Hintergrundes. Bei Überstrahlungen durch direktes Licht oder hautfarbenen Objekten im Hintergrund ist eine Erkennung der Hand über die Hautfarbe und somit eine Positionssteuerung des Cursors leider nur bedingt möglich. Die Erkennung des Mausklicks und die Gestenerkennung funktionieren dann durch eine unvollständige Handkontur meist nur unzureichend.

Stimmt jedoch die Beleuchtung und wird der Hintergrund von hautfarbenen Objekten befreit, lässt sich der Cursor durch die bloße Handbewegung ohne Verzögerung positionieren. Über den Joystick-Modus ist jeder Bildpunkt ansteuerbar. Im Maus-Modus lässt sich die Maus schnell, wenn auch je nach Video-Auflösung und Erkennungsgenauigkeit etwas grober und unruhiger, positionieren. Die Erkennung des Mausklicks und die Gestenerkennung funktionieren durch die saubere Handkontur robust und fehlerfrei.

Diese Arbeit hat gezeigt, dass mit einer einfachen Webcam ein interessantes neues Eingabegerät realisiert werden kann. Das System will die Computermaus, die seit Jahrzehnten als Standard-Eingabegerät neben der Tastatur benutzt wird, nicht ersetzen. Vorstellbar wäre aber z.B. der Einsatz in interaktiven Informationsständen in Museen oder Ausstellungen. Dort könnten Besucher mit der bloßen Hand anstatt per Touchscreen oder Maus zwischen den einzelnen Informationsangeboten navigieren. Sicherlich würden Besucher, die noch nie eine Maus in der Hand gehabt haben, diese neue intuitive Eingabeform schnell verstehen und ihr positiv gegenüberstehen.

7.2 Weiterentwicklungen

Trotzdem das Ziel, den Computer mit der bloßen Hand zu steuern, mit dem System erreicht wurde, ist dennoch Platz für Erweiterungen und Weiterentwicklungen.

Trennung von Gesicht und Hand

Bisher sucht sich das System den größten hautfarbenen Bereich im Videobild und geht davon aus, dass es sich um die Hand handelt. Wird das Gesicht mit erfasst und erscheint es im Videobild größer als die Hand, muss zur Zeit noch der Fokus der Kamera verändert werden. Das System könnte so modifiziert werden, dass es anhand der Form erkennt, ob es sich bei dem hautfarbenen Objekt um ein Gesicht oder eine Hand handelt. Ebenso wäre es denkbar, dass das System bei Tragen eines kurzärmeligen Kleidungsstückes automatisch den Arm von der Hand trennt und durchs Videobild laufende Personen erkennt.

Hautfarbene Objekte im Hintergrund

Zum Trennen von Vorder- und Hintergrundobjekten wird bislang ausschließlich die Hautfarbe benutzt. Objekte wie Holzmöbel, deren Farbe der Hautfarbe ähneln, müssen daher vom Anwender aus dem Hintergrund entfernt werden. Da dies nicht immer möglich ist, würde eine Erkennung solcher Gegenstände die Benutzerfreundlichkeit erhöhen. Um dies zu erreichen, könnte die Bewegung im Videobild mittel Motion Segmentation (siehe 3.3) als zusätzliches Erkennungsmerkmal dienen.

Erweiterte Fingerspitzenerkennung

Die Fingerspitzenerkennung berechnet derzeit lediglich die Position und die Anzahl der gestreckten Finger. Welcher der Finger, also ob z.B. der Zeigefinger oder Mittelfinger gestreckt wurde, ist nicht bekannt. Auch kann der Anwender nur eine Hand benutzen. Eine Erweiterung der Fingerspitzenerkennung in diese Richtungen würden weitere und detailliertere Gesten ermöglichen. Als Beispiel könnte der Anwender mit dem Zeigefinger der einen Hand die Maus steuern, während er mit der anderen Hand ein Fenster verschiebt. Mit dem Spreizen von Daumen und Zeigefinger wäre es möglich, Bildausschnitte zu vergrößern und zu verkleinern.

Erweiterte Gestenerkennung

Die derzeit implementierte Gestenerkennung erfasst lediglich Start- und Endpunkt einer Handbewegung und lässt nur vier Bewegungen (links, rechts, oben, unten) zu. Würde man alle Positionen einer Handbewegung erfassen, ließen sich komplexere Gesten realisieren. So könnte die Hand geometrische Figuren wie Kreis und Dreieck beschreiben. Auch wäre die Anbindung einer Handschriftenerkennung denkbar.

Erkennen von Überstrahlungen

Damit es durch Überstrahlungen nicht zu fehlerhaften Handkonturen kommt, muss direktes Licht z.B. durch die Abdunkelung des Raumes verhindert werden. Eine Alternative wäre die Erkennung von Überstrahlungen im Videobild. Dazu könnte entlang der vorhandenen, aber unvollständigen Handkontur nach weißen Bereichen gesucht werden. Durch eine Flutfüllung (floodfill) könnten diese Bereiche markiert und mit in die Handkontur einbezogen werden.
Eine Testimplementierung erzielte teilweise recht gute Ergebnisse. Sobald sich jedoch hinter der Hand ebenfalls Überstrahlungen oder sehr helle Objekte befanden, kam es zu Fehlerkennungen.

7.3 Ausblick

Wird die heutige Computermaus auch die nächsten Jahrzehnte als *das* Eingabegerät neben der Tastatur überstehen? Die Hersteller werden nicht müde, immer neue Technologien in eine Maus zu stecken. So hat Apple kürzlich ein Patent veröffentlicht, in dem eine vollkommen tastenlose „Multitouch-Maus“ [13] beschrieben wird. Alle Aktionen des Benutzers, wie z.B. der Mausklick, werden über eine Gestenerkennung durch eine Kamera im Inneren des Gehäuses erfasst. Aber auch wenn die Bedienung der Multitouch-Maus vielleicht noch einfacher wird, bleibt es doch ein Gerät, an das sich der Anwender erst einmal gewöhnen muss.

Oder wird es demnächst vielleicht keine Eingabegeräte, so wie wir sie heute kennen, mehr geben? Sicher ist, dass sie näher an den Menschen heranrücken, d.h. dass sie intuitiver werden und sich mehr an den menschlichen Kommunikationsformen orientieren. Vielleicht werden wir sie nicht mehr so wahrnehmen wie heute. Eine nicht sichtbare Kamera, die nicht nur die Hand, sondern den Menschen als Ganzes erfasst und alle nur denkbaren Ausdrucksformen versteht, könnte alle bisherigen Eingabegeräte ersetzen. Diese Kamera - wenn man es denn so nennen möchte - würde sich nicht nur am Arbeitsplatz befinden, sondern wäre allgegenwärtig und könnte immer und überall Eingaben entgegennehmen. In dem Film „2001 - Odyssee im Weltraum“ besitzt der Computer „HAL“ diese Eigenschaft. Er versteht über die Sprache, Mimik und Gestik auch die Gefühle seines Gegenübers und ersetzt fast einen realen Menschen. Und können wir nicht am besten mit einem Menschen kommunizieren?

A Anhang

A.1 Klassendiagramme

In den folgenden Abbildungen sind die Klassendiagramme der wichtigsten Klassen des Systems abgedruckt:

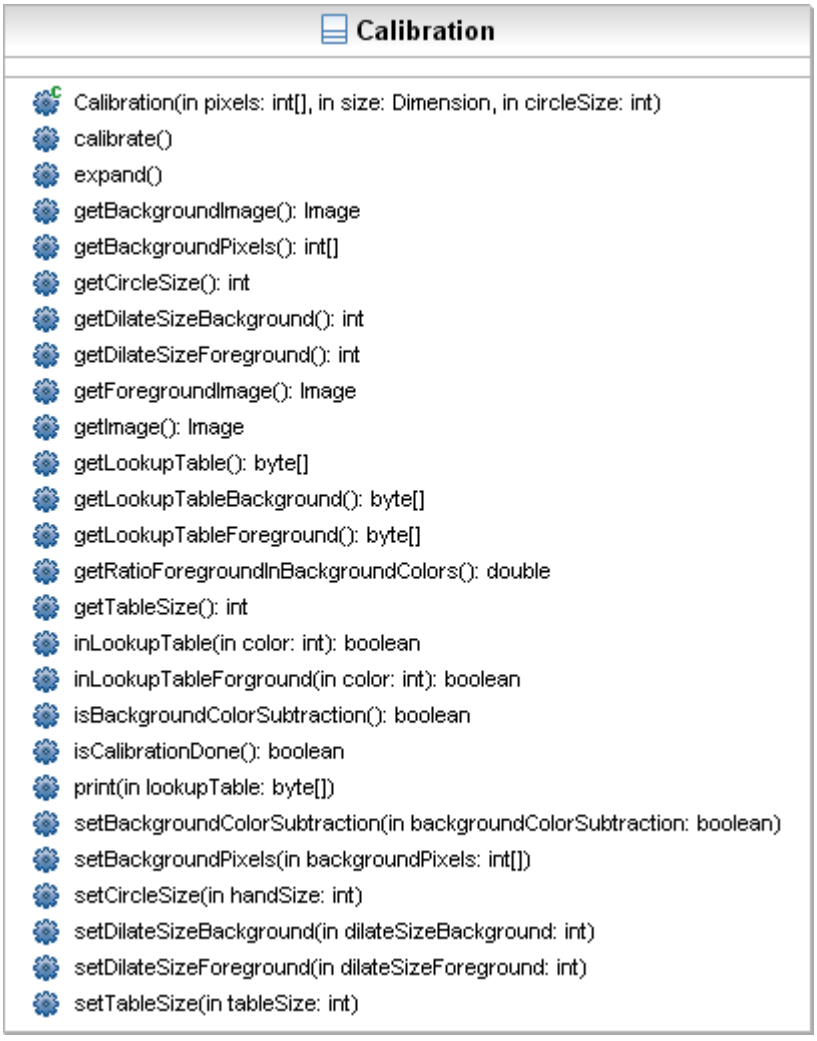


Abbildung A.1: Klassendiagramm Calibration

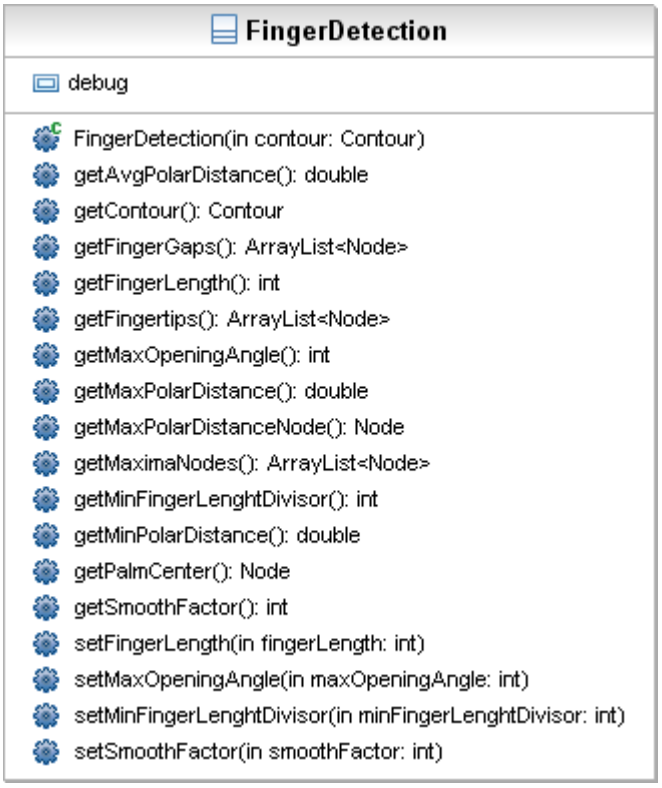


Abbildung A.2: Klassendiagramm FingerDetection

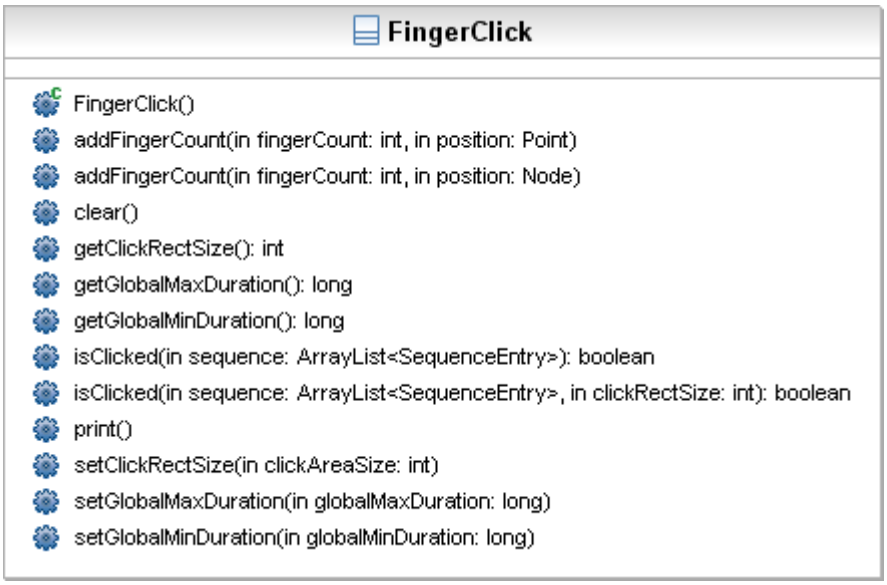


Abbildung A.3: Klassendiagramm FingerClick

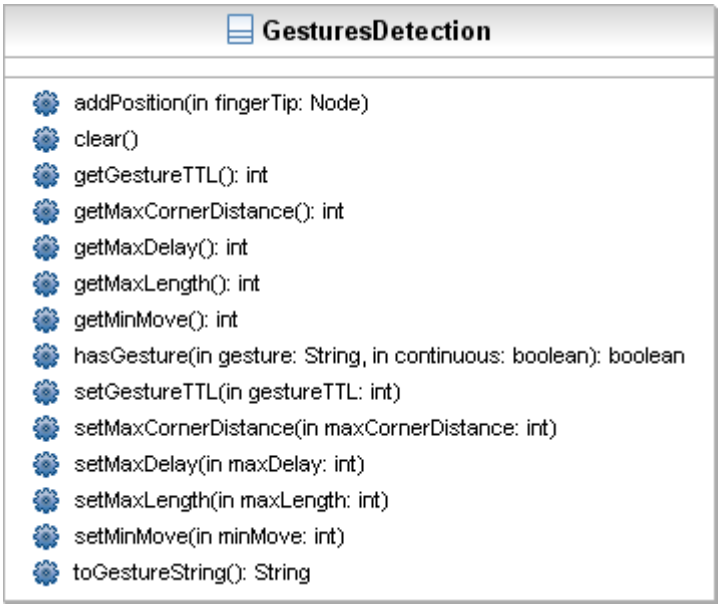


Abbildung A.4: Klassendiagramm GesturesDetection

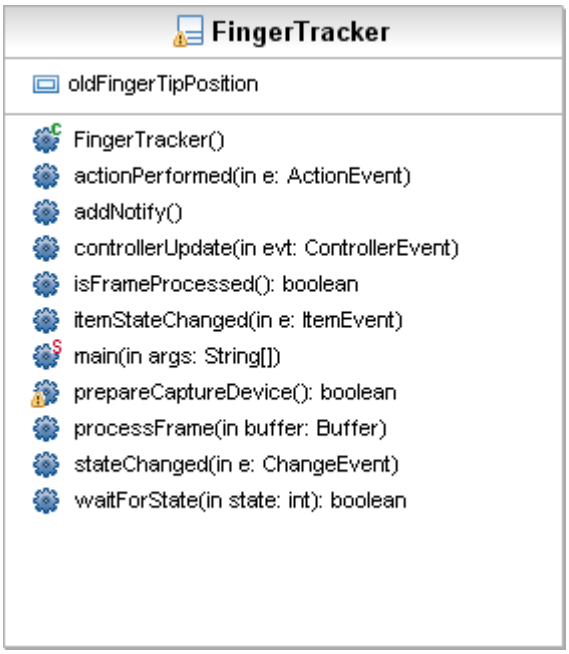


Abbildung A.5: Klassendiagramm FingerTracker

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[BB06] Wilhelm Burger and Mark James Burge. *Digitale Bildverarbeitung; Eine Einführung mit Java und ImageJ*, pages 79–104,112–133,170–188,255–267. Springer Verlag Berlin Heidelberg, 2006.

[BK98] Henning Bässmann and Jutta Kreyss. *Bildverarbeitung AdOculos, Dritte, vollständig überarbeitete und erweiterte Ausgabe*. Springer Verlag Berlin Heidelberg, 1998.

[Hab95] Peter Haberäcker. *Praxis der Digitalen Bildverarbeitung und Mustererkennung*, pages 51–56. Carl Hanser Verlag München Wien, 1995.

[Her05] Thorsten Hermes. *Digitale Bildverarbeitung; Eine praktische Einführung*, pages 58–65. Carl Hanser Verlag München Wien, 2005.

Onlineverzeichnis

[1] Henry Chang and Ulises Robles. *Face Detection, EE368 Final Project Report - Spring 2000.* http://www-cs-students.stanford.edu/~robles/ee368/skincolor.html, http://www-cs-students.stanford.edu/~robles/ee368/skinsegment.html, 11.06.2007.

[2] Intel Corporation. *OpenCV Reference Manual*, pages 21–22. pdf-document, www.cs.unc.edu/Research/stc/FAQs/OpenCV/OpenCVReferenceManual.pdf, 03.04.2007.

[3] Stephen C. Crampton and Margrit Betke. *Counting Fingers in Real Time: A Webcam-Based Human-Computer Interface with Game Applications*, page 3. pdf-document, http://cs-people.bu.edu/stevec/research/Crampton-Betke.pdf, 30.05.2007.

[4] Christopher K. Eveland, Diego A. Socolinsky, and Lawrence B. Wolff. *Tracking Human Faces in Infrared Video*, pages 1–5. pdf-document, http://www.equinoxsensors.com/publications/model.pdf, 30.05.2007.

[5] Y. Hung, Y. Yang, Y. Chen, I. Hsieh, and C. Fuh. *Free-hand pointer by use of an active stereo vision system.* pdf-document, http://ippr.csie.ntu.edu.tw/Publication/15_Free-HandPointerbyUseofanActiveStereo.pdf, 30.05.2007.

[6] Fraunhofer Institut. *Stereo Hand Tracker.* pdf-document, http://www.hhi.fraunhofer.de/german/im/produkte/fingerposition/art/Flyer-Fraunhofer-HHI-Videotracker.pdf, 04.06.2007.

[7] Jay P. Kapur. *Face Detection in Color Images, EE499 Capstone Design Project Spring 1997.* http://www.geocities.com/jaykapur/face.html, 29.05.2007.

[8] Johannes Matiasch. *Motion Detection: Eine Einführung*, pages 3–5. pdf-document, http://www.prip.tuwien.ac.at/Teaching/WS/ProSemSab/prosem05/pdf/Matiasch.pdf, 29.05.2007.

[9] Sun Microsystems. *Java Media Framework API Guide*, pages 11–15. pdf-document, http://sdlc-esd.sun.com/ESD4/JSCDL/jmf/2.0/jmf2_0-guide.pdf, 29.05.2007.

[10] Sun Microsystems. *Sun Developer Network, Developer Forums, Multimedia and Imaging APIs - Java Media Framework.* http://forum.java.sun.com/forum.jspa?forumID=28, 12.06.2007.

[11] OpenCV. *OpenCV Forums.* http://sourceforge.net/forum/forum.php?forum_id=72228, 12.06.2007.

[12] OpenCV. *OpenCV Library Wiki.* http://opencvlibrary.sourceforge.net, 03.04.2007.

[13] Die Presse.com. *Apple: Patent für tastenlose Multitouch-Maus eingereicht.* http://www.diepresse.com/home/techscience/hightech/apple/315353, 25.07.2007.

[14] Christiaan M. van der Walt. *Optimization Methods for Skin Based Face Detection, Poster papers from the Sixteenth Annual Symposium of the Pattern Recognition Association of South Africa*, pages 56–59. pdf-document, `http://www.prasa.uct.ac.za/PRASA2005_posters.pdf`, 29.05.2007.

[15] Wikipedia. *Java Media Framework*, pages 10–20. `http://de.wikipedia.org/wiki/Java_Media_Framework`, 31.12.2007.

[16] Gesellschaft Arbeit und Ergonomie online e.V. Wolfgang Schneider. *Ergonomische Dialoggestaltung*. `http://www.ergo-online.de/site.aspx?url=html/software/ergonomische_dialoggestaltung/titel.htm`, 14.06.2007.

Glossar

Computer Vision	Computergestütztes „Sehen“, umfasst mehrere Teilgebiete wie z.B. Mustererkennung, Bildverarbeitung, Bewegungserkennung, Objekterkennung und Objektklassifikation. Typische Anwendungen sind Gesichtserkennung, Gestenerkennung, Qualitätsprüfung und Videoüberwachung
Frame	Einzelbild im kontinuierlichen Datenfluss einer Videokamera, eines Videofilms oder Computerspiels
Framework	Rahmenstruktur, Design-Grundstrukur, die einen beim Erstellen einer Anwendung unterstützt
Handle	engl.: *handle* Griff, Henkel, identifiziert unter Windows ein Kontrollelement wie z.B. Fenster, Schaltfläche oder Bildlaufleiste
HSB	Farbraum, bei dem die Farbe über Farbton (hue), Farbsättigung (saturation) und Helligkeit (brightness) identifiziert wird
JMF	Java Media Framework API, Programmierschnittstelle zur Handhabung von zeitbasierten Medien
Motion Detection	Erkennen und Erfassen von Bewegung im Verhältnis zu einem als statisch betrachteten Hintergrund
Motion Segmentation	Erkennung von bewegten Bildausschnitten, Trennung vom Hintergrund und Einteilung in Klassen von Objekten
OpenCV	Open Source Computer Vision, freie Bibliothek für die Entwicklung von Computer Vision-Anwendungen

RGB	Farbraum, bestehend aus den drei Komponenten Rot, Grün und Blau, die sich zu Weiß addieren (additiv)
USB	Universal Serial Bus, mit dem externe Peripheriegeräte wie Maus, Tastatur oder Webcam an den Computer angeschlossen werden
Webcam	Preisgünstige Videokamera zum Anschluss an den Computer, die ursprünglich für das fortlaufende Einstellen von Bildern ins Internet entwickelt wurde
Workstation	Leistungsfähiger und zuverlässiger Rechner für die Ausführung von anspruchsvollen Anwendungen

Eigenständigkeitserklärung

Fachhochschule für Technik und Wirtschaft Berlin
Fachbereich Wirtschaftswissenschaften II
Studiengang Angewandte Informatik

Diplomarbeit

„Entwicklung eines Motion Tracking-Systems zur Positionssteuerung per Handbewegung“

Eingereicht von Alexander Miehlke

1. Betreuer: Prof. Dr.-Ing. Thomas Jung
2. Betreuer: Prof. Dr. Horst Hansen

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel verfasst habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Berlin, 4. September 2007