Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Daniel Schulze

Eine Processing-Erweiterung um kamerabasierte Gestenerkennung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Daniel Schulze

Eine Processing-Erweiterung um kamerabasierte Gestenerkennung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. rer. nat. Kai von Luck
Zweitgutachter : Prof. Dr. rer. nat. Gunter Klemke

Abgegeben am 15. September 2010

Daniel Schulze

Thema der Bachelorarbeit

Eine Processing-Erweiterung um kamerabasierte Gestenerkennung

Stichworte

Gestenerkennung, Processing, Mensch-Maschine-Interaktion, Handerkennung, Hauterkennung

Kurzzusammenfassung

Diese Arbeit beinhaltet die Realisierung einer Bibliothek für Processing, die es ermöglicht kamerabasiert Gesten zu erfassen. Ziel ist es, Menschen ohne gute Programmierkenntnisse zu ermöglichen, die Gestenerkennung in ihre Programme zu integrieren. Es werden alle Schritte beschrieben, die durchlaufen werden - vom Einlesen des Kamerabildes bis zur erkannten Geste. Die entwickelte Bibliothek wird anschließend in ein Programm zur Bildbetrachtung eingebunden um die Funktionalität zu demonstrieren.

Daniel Schulze

Title of the paper

An extension for Processing to include camera-based gesture recognition

Keywords

gesture recognition, Processing, human-computer interaction, skin color detection, hand detection

Abstract

This paper describes the implementation of a software extension for Processing which includes camera-based gesture recognition. The intention is to help people without good knowlege of programming languages to integrate gesture recognition into their programs. All required steps will be described - from getting the image from the camera to returning the recognized gestures. The developed library is then integrated into a program for viewing images to show its capabilities.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

1.1 Motivation

Die Vergangenheit hat gezeigt, dass Computer inzwischen überall vertreten sind - von der Informationstafel im Supermarkt bis zum Entertainment-System im Auto. Dabei lassen sich nicht alle über das klassische Bedienkonzept mit Maus und Tastatur steuern. Für diese neue Situation bedarf es also neuer Bedienkonzepte.

Im Bereich der Touchscreens spielen Multitouch und Gestenerkennung schon seit einiger Zeit eine Rolle. Inzwischen sind sie in den meisten neueren Mobiltelefonen implementiert und damit auch der breiten Öffentlichkeit zugänglich. Allerdings ist der Anwendungsrahmen von Touchscreens begrenzt und somit lassen sich nicht alle Probleme lösen, indem man einen Touchscreen einbaut.

In Filmen ist immer wieder zu beobachten, wie einer der Schauspieler einen Computer mit Gesten steuert. Beispiele sind hier die Filme Minority Report oder auch IronMan 2. Zwar ist das dort Gezeigte eher in die Kategorie „Science-Fiction“ einzuordnen, aber das Konzept der Steuerung über Gesten ist auch in der realen Welt, jenseits von Hollywood, denkbar.

Weiterhin sieht man immer wieder, dass die Besucher von Kunstinstallationen gefilmt werden und dass dann auf ihre Bewegungen reagiert wird. Dies ist gewissermaßen eine Vorstufe der Gestenerkennung. Die Bewegungen werden aufgenommen und mit Animationen auf einer Leinwand wird darauf reagiert. Die Bewegungen werden allerdings nicht weiter interpretiert.

Es ist also zu beobachten, dass Gestensteuerung zwar bei Touchscreens durchaus verbreitet ist, ansonsten aber nur recht vereinzelt anzutreffen ist. Dies hat sicherlich mehrere Gründe. Einer davon ist, dass die technische Umsetzung nicht trivial ist. Ferner sind die Künstler, die das benutzen würden, in der Regel keine ausgebildeten Informatiker und verfügen damit nicht über das Wissen, wie kamerabasierte Gestenerkennung technisch umzusetzen ist.

Das ist der Punkt, an dem Informatiker ins Spiel kommen. Sie verfügen über Kenntnisse von Programmiersprachen, mit deren Hilfe es möglich ist, kamerabasierte Gestenerkennung in die Realität umzusetzen. Dabei muss allerdings sichergestellt werden, dass die Anwendung am Ende von Jedem benutzt werden kann - auch von Personen, die nur über wenige Programmierkenntnisse verfügen.

Die Aufgabe dieser Bachelorarbeit wird es also sein, eine Bibliothek zu entwickelt, die es Anderen erlaubt, ihre Programme durch Gestenerkennung zu erweitern.

1.2 Gliederung

Im Anschluss an dieses Kapitel werden in den Grundlagen einige Begriffe erklärt, die für den weiteren Verlauf wichtig sind.
In der darauf folgenden Analyse geht es dann um die bestehenden Problemstellung und die dafür anvisierte Lösung. Desweiteren werden bestehende Projekte vorgestellt, bei denen Gestenerkennung eingesetzt werden könnte.
Danach geht es weiter mit den Algorithmen, die benötigt werden, um die Gestenerkennung umzusetzen. Dabei wird der Ablauf der Gestenerkennung Schritt für Schritt erklärt und die benutzten Verfahren werden erläutert.
Im Kapitel Umsetzung wird die entwickelte Erweiterung dann an Hand eines Beispielprogramms erläutert und es wird Verbesserungspotenzial aufgezeigt.
Im Schluss wird eine kurze Zusammenfassung gegeben und es wird ein Fazit gezogen.

2 Grundlagen

2.1 Definitionen

2.1.1 Gesten

Gesten sind eine Art der non-verbalen Kommunikation, bei der mit Händen oder dem Gesicht Bewegungen ausgeführt werden, um dem Gegenüber etwas mitzuteilen. Wir benutzen Gesten ganz selbstverständlich im Alltag - von der Zeigegeste, um jemand auf etwas aufmerksam zu machen, bis hin zur Mimik, um Gefühle auszudrücken.

Bei der Interaktion zwischen Menschen und Computersystemen ist die Gestensteuerung bisher selten anzutreffen, allerdings steigt die Verbreitung, vor allem bei Touchscreens, stetig an. Neben Touchscreens können Gesten aber auch über Kameras erfasst werden. Dabei gibt es die Möglichkeit, die Hände des Benutzers im Kamerabild zu suchen oder aber auf Hilfsmittel zurückzugreifen. Dabei können z. B., wie bei [Rödiger, 2010] Infrarotsensoren zur genauen Positions- und Lageerkennung beitragen.

[Pavlovic u. a., 1997] teilt Gesten in manipulative und kommunikative Gesten ein, geht dann allerdings nur auf die kommunikativen weiter ein. Im Bereich der Mensch-Maschine-Interaktion kommen aber eher die manipulativen zum Einsatz. Der Anwender versucht nicht mit dem System zu kommunizieren, sondern er versucht das System zu manipulieren, indem er z. B. Objekte auf einem Bildschirm hin- und herschiebt.

Auf Basis dessen teilen [Boetzer u. a., 2008] die manipulativen Gesten in drei Grundarten ein:

> **Bewegungsverfolgung** Damit sind Gesten gemeint, die aus der physikalischen Welt übernommen wurden. Dahinter verbirgt sich z. B. das Hin- und Herschieben von Objekten auf einer virtuellen Oberfläche. Diese Gesten sind intuitiv benutzbar.

kontinuierliche Gesten Hier finden sich keine direkten Vorbilder in der realen Welt, aber trotzdem sind diese Art der Gesten leicht zu lernen. So fällt z. B. das Vergrößern und Verkleinern durch das Auseinanderziehen und Zusammenschieben zweier Finger, wie es bei diversen Produkten der Firma Apple umgesetzt ist, in diese Kategorie.

symbolisch-manipulative Gesten Diese Art der Gesten beschreibt abstrakte Gesten, für deren Anwendung ein gewisser Lernaufwand notwendig ist. Darunter fällt z. B. die Bewegung mit drei Fingern auf einem Touchscreen, um ein gezeigtes Objekt umzudrehen.

Des Weiteren teilt [Rödiger, 2010] Gesten noch in statische und dynamische Gesten ein. Der Name ist dabei selbsterklärend. In die Kategorie der statischen Gesten fallen die, bei denen sich die Hand des Anwenders nicht bewegt. Dazu gehören Zeigegesten oder ein „Daumen hoch“. Zu den dynamischen Gesten gehört dementsprechend alles, bei dem die Hand eine Bewegung macht. Diese Bewegung wird über einen gewissen Zeitraum beobachtet und daraus wird dann geschlossen, um welche Geste es sich handelt.

2.1.2 Processing

Hinter dem Wort Processing verbirgt sich eine Programmiersprache. Allerdings handelt es sich hierbei nicht um eine eigenständige Sprache, sondern eigentlich nur um einen Aufsatz auf die Programmiersprache Java (vgl. [Processing, 2010]). Dabei läuft es so, dass der Preprocessor aus dem Processing-Code erst einmal Java-Code generiert. Dieser durchläuft dann im Folgenden die übliche Java-Toolchain und wird anschließend in ausführbaren Code umgewandelt.

Da Processing auf Java aufsetzt, sind dort auch alle Möglichkeiten von Java verfügbar. So stellt Java z. B. die Funktionalität für Dateioperationen oder Netzwerkverbindungen bereit und diese können somit auch in Processing genutzt werden. Die Funktionsvielfalt von Java ist zwar ein großer Vorteil, allerdings kann es für Anfänger sehr unübersichtlich und komplex wirken. Es bedarf einer gewissen Zeit, bis man sich mit Java vertraut gemacht hat und damit produktiv arbeiten kann.

An dieser Stelle setzt Processing nun an. Hauptziel von Processing ist es, dem Benutzer Möglichkeiten an die Hand zu geben, mit möglichst wenigen und einfachen Befehlen etwas auf dem Bildschirm auszugeben. Gerade grafische Effekte sind in Java nicht ganz einfach zu realisieren. In Processing hingegen lassen sie sich mit wenigen Befehlen umsetzen. Aus

diesem Grund eignet sich die Sprache auch gut für „Nicht-Informatiker“, die bisher keine oder nur geringe Programmierkenntnisse besitzen.

Entwickelt wurde Processing ab 2001 am Massachusetts Institute of Technology (MIT) in Cambridge, USA. Im November 2005 wurde es unter GNU Public License gestellt und ist damit als frei verfügbare Software erhältlich (Open Source).

2.1.3 OpenCV

Das „CV“ in OpenCV steht für „Computer Vision“ und das lässt darauf schließen, dass es etwas mit maschinellem Sehen zu tun haben muss. In der Tat handelt es sich hierbei um eine Bibliothek, die dem Anwender über 500 Funktionen aus dem Bereich Bildverarbeitung und Bilderkennung bereitstellt (vgl. [Bradski und Kaehler, 2008]). Das reicht vom einfachen Gauß-Filter, mit dem ein Bild unscharf gemacht werden kann, über die Hough-Transformationen, um Linien in einem Bild zu detektieren, bis hin zur Gesichtserkennung. Für den Benutzer hat dies den Vorteil, dass er teilweise sehr komplexe Verfahren einfach benutzen kann, ohne sich über die genaue Funktionsweise zu informieren und ohne die mathematischen Hintergründe verstehen zu müssen. Dabei können die verschiedenen Verfahren von OpenCV sowohl auf einzelne Bilder als auch auf Videos angewandt werden.

Die Bibliothek wurde ursprünglich von Intel entwickelt, allerdings wird sie aktuell hauptsächlich von der Firma Willow Garage gepflegt. Der Code steht unter der BSD-Lizenz, was bedeutet, dass er frei verfügbar ist und auch in kommerziellen Produkten verwendet werden darf.

Entwickelt wurde die Bibliothek in C und C++, weil es mit diesen Sprachen möglich war, sehr ressourcensparend und effizient zu programmieren. Für Benutzer, die nicht über C oder C++ auf die Funktionen zugreifen wollen oder können, stehen auch Schnittstellen zu anderen Sprachen wie Python oder Ruby zur Verfügung. Desweiteren stellt OpenCV sogar eine Schnittstelle für MATLAB (ein Programm zur Lösung komplexer mathematischer Probleme) bereit.

2.1.4 Kamerabasierte Erfassung

Das „kamerabasiert“ im Titel dieser Arbeit bedeutet, dass hierbei eine Kamera als Eingabemedium genutzt werden soll. Dabei wird der Benutzer von einer einfachen Kamera (z. B. einer Webcam) gefilmt. In diesem Video werden die Bewegungen des Benutzers auf definierte

Gesten hin überprüft. Dabei entsteht das Problem, dass der Benutzer sich in einer dreidimensionalen Welt bewegt, die Kamera mit ihrem Sensor aber nur ein zweidimensionales Abbild davon erfasst. Dadurch geht die Tiefeninformation unwiederbringlich verloren.

Um dem Informationsverlust entgegen zu wirken, gibt es einige Lösungen, wie z. B. das Berechnen einer Distanz über die Perspektive. Des Weitere existieren Hardwarelösungen, mit denen auch die dritte Dimension erfasst werden kann. Zum einen ist da die Kombination aus zwei Kameras zu nennen. Diese filmen eine Szene aus zwei verschiedenen Blickwinkeln und am Computer wird aus diesen beiden Perspektiven dann ein dreidimensionales Bild berechnet. Eine weitere Hardwarelösung ist die TOF-Kamera (*time of flight*). Diese Kamera sendet einen (meist infraroten) Lichtstrahl aus und misst die Zeit, die dieser braucht, um vom Objekt reflektiert zu werden. Durch die Laufzeit des Lichts kann dann die Entfernung zum Objekt berechnet werden. Die genannten Lösungen sind allerdings sehr rechenintensiv und die TOF-Kameras sind zudem auch noch sehr teuer.

2.2 Verwandte Arbeiten an der HAW

Boetzer [2008] beschreibt die gestenbasierte Steuerung auf Basis eines Motion Trackers. Dieser besteht aus vier Infrarotkameras, die die Position der Hand erkennen und verfolgen. Es werden verschiedene Gesten entwickelt und getestet.

Rahimi und Vogt [2008] beschreiben die Gestenerkennung bei Multi-Touch-Oberflächen. Sie entwerfen dafür verschiedene Anwendungen und entwickeln eine intuitive Bedienung anhand von Gesten.

Rödiger [2010] beschreibt dreidimensionale Gesten an der Powerwall der HAW. Dafür nutzt er Marker, die von infrarotem Licht angeleuchtet werden und die dann von einer Kamera erfasst werden. Durch die feste Anordnung der Marker ist es möglich diese auch im Raum zu lokalisieren und Bewegungen in alle drei Dimensionen festzustellen.

3 Analyse

3.1 Ist-Analyse die Erste

3.1.1 Situationsbeschreibung

Gesten sind eine Teil unserer alltäglichen Kommunikation. Wir zeigen auf etwas, um darauf hinzuweisen, und auch bei der verbalen Kommunikation setzen wir häufig Gesten zur Unterstützung ein. Dies geschieht eher unbewusst und ohne, dass wir diese Gesten großartig lernen müssen. Bei der Kommunikation mit Maschinen bedienen wir uns eines extra dafür erlernten Konzepts mit Maus und Tastatur. Andere Eingabemethoden, die uns natürlicher erscheinen, wie z. B. die Spracheingabe, werden zwar seit einiger Zeit erprobt, haben sich aber auch auf Grund ihrer teils hohen Fehlerquote bisher nicht durchsetzen können.

Daneben taucht auch der Ansatz, Computersysteme über Gesten zu steuern, seit Jahren immer wieder auf. Wie so vieles war er als erstes im universitären Umfeld zu finden. Als einer der Vorreiter ist Jefferson Han zu nennen, der mit seinen Multitouch-Anwendungen an der Universität in New York (siehe [Han, 2010]) immer wieder für Aufsehen gesorgt hat.

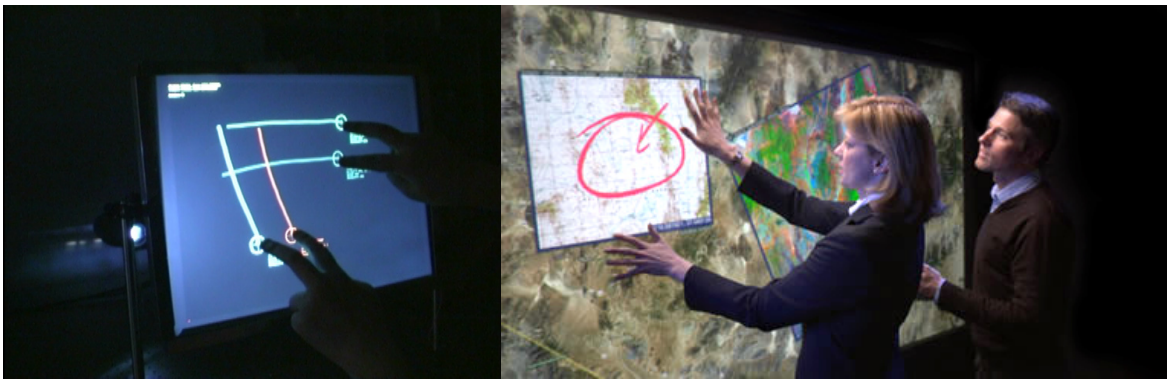


Abbildung 3.1: Multitouch-Anwendungen von Jefferson Han

Im Bereich der Kunst sind Gesten ebenfalls immer mal wieder anzutreffen. Hauptsächlich sieht man in diesem Bereich aber eher die einfache Reaktion auf Bewegungen aller Art. Das liegt vermutlich hauptsächlich daran, dass Bewegungserkennung deutlich einfacher umzusetzen ist als Gestenerkennung.

Zu nennen ist hier sicherlich Scott Snibbe, der mit diversen Kunstinstallationen bereits auf sich aufmerksam gemacht hat (siehe [Snibbe, 2010b]). In vielen seiner Projekte gibt es eine Leinwand, auf der Animationen ablaufen (siehe auch Kap. 3.1.2). Tritt nun ein Besucher vor diese Fläche, so wird er von Kameras erfasst und die Animationen reagieren auf seine Anwesenheit und seine Bewegungen. Eine weitere Interpretation der Bewegungen, insbesondere Gestenerkennung, findet allerdings nicht statt.

In eine ähnliche Richtung geht das Projekt Digital Urban Living [DUL, 2010a]. Diese dänische Gruppe von Kreativen hat einige Projekte verwirklicht, bei denen große Häuserwände als Projektionsfläche dienten, um diverse Animationen darzustellen. Die Besucher wurden von Kameras gefilmt und konnten durch ihre Bewegungen in das Geschehen auf der Wand eingreifen (siehe Kap. 3.1.2). Des Weiteren wurde die Außenfassade des dänische EXPO-Pavillons auf der Weltausstellung 2010 in Schanghai, China, von Digital Urban Living gestaltet und mit Technik ausgestattet, um darauf Animationen ablaufen zu lassen.

Die beiden genannten Projekte beschränken sich allerdings auf die Erkennung von Bewegungen der Besucher. Dies kommt daher, dass eine komplexe Gestenerkennung doch um einiges schwieriger umzusetzen ist als eine einfache Erkennung des Menschen und seiner Bewegungen. Die Interpretation der Bewegungen, und nichts anderes sind Gesten, ist aber bereits in Ansätzen zu beobachten. Daraus lässt sich schließen, dass es Bedarf in diese Richtung gibt und dass die Lösung eher an der Komplexität der Aufgabe scheitert. Dies wird deutlich, wenn man sich anschaut, wie kompliziert die Erkennung von Gesten sein kann. Was für uns Menschen so einfach erscheint, ist, wie so oft, für die Technik relativ schwierig. Dazu gehört es eben auch zu erkennen, ob sich ein Mensch im Kamerabild befindet und ob er winkt oder auf etwas bestimmtes zeigt. Da diese Erkennung recht spezifisches Wissen über Softwaretechnik und Bilderkennung voraus setzt, fehlen vielen Nicht-Informatikern die Fähigkeiten, um so ein Projekt ohne fremde Hilfe umzusetzen.

3.1.2 Projekte

Einige Künstler wurden bereits im vorherigen Kapitel genannt, werden hier aber noch einmal kurz vorgestellt.

Scott Snibbe

Scott Snibbe ist ein amerikanischer Medienkünstler und Filmemacher. Seine Kunstinstallationen beinhalten häufig eine Wand, auf der Animationen ablaufen. Nähert sich ein Besucher der Wand, so reagieren die Animationen auf seine Anwesenheit und auf seine Bewegungen.

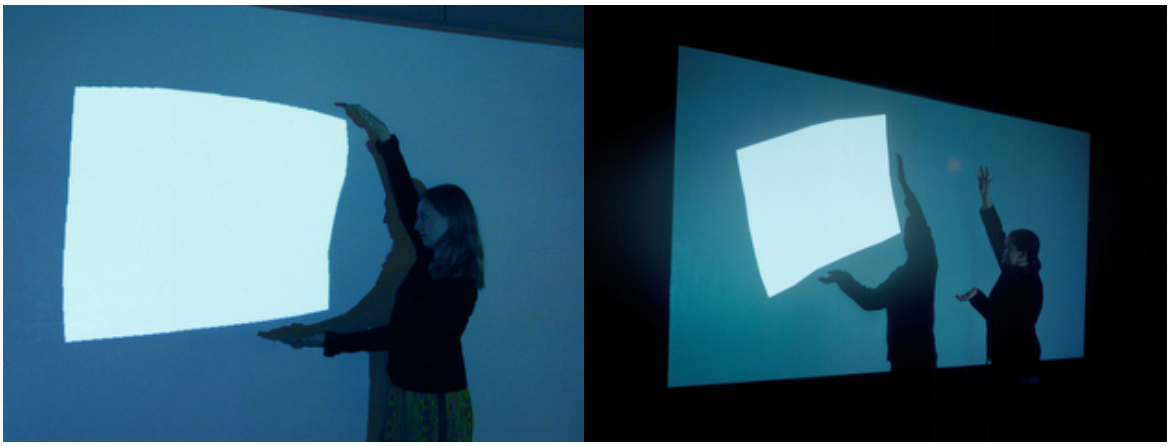


Abbildung 3.2: „Compliant“ von Scott Snibbe [Snibbe, 2010a]

Als Beispiel sei hier das Projekt „Compliant“ aus dem Jahre 2003 genannt. Über einen Beamer wird ein weißes Quadrat von etwa 1 mal 1,5 Metern an die Wand projiziert. Eine Kamera erkennt, wenn nun ein Besucher zwischen Projektor und Wand tritt. Berührt der Schatten des Besuchers das weiße Quadrat, so wird dieses verschoben oder verformt. In Abbildung 3.2 wird dies demonstriert. Im linken Teil des Bildes drückt die Frau mit ihren Armen das Quadrat zusammen und es ist zu erkennen, wie das Quadrat dem Druck nachgibt und sich verformt. Im rechten Teil der Abbildung nähert sich ein Besucher dem Quadrat, verschiebt und dreht es auf diese Weise.

Digital Urban Living

Bei Digital Urban Living handelt es sich um ein Projekt, das an der Universität Aarhus, Dänemark, entstanden ist. Dabei geht es hauptsächlich um Forschung in Richtung Kommunikation und Kunst im öffentlichen Raum. Unter diesem Motto wurde 2008 „Aarhus By Light“ ins Leben gerufen (siehe [DUL, 2010b]). Dafür wurde die Fassade des Konzerthauses in Aarhus für zwei Monate in einen interaktiven Bildschirm verwandelt.

Abbildung 3.3: Das Konzerthaus in Aarhus, Dänemark

Dazu wurden an der Glasfassade halbtransparente Bildschirme mit einer Gesamtfläche von etwa 180 Quadratmetern angebracht. Zu sehen war dort dann eine Art „sich bewegende Skyline“, die bekannte Bauwerke von Aarhus zeigte. Außerdem wurde die Fläche von kleinen Kreaturen bevölkert, die sich dort frei bewegten und miteinander interagierten. Zusätzlich gab es vor dem Gebäude drei farbige Flächen in pink, türkis und gelb (zu sehen in Abb. 3.3). Besucher, die diese Flächen betraten, wurden von einer Kamera erfasst und ihre Konturen wurden, wie in Abbildung 3.4 zu sehen, auf die Fassade übertragen. Somit wurde es dem Besucher unter anderem ermöglicht mit den bunten Kreaturen zu interagieren, indem diese

verschoben oder hochgehoben wurden. Diese Interaktion wurde dadurch gefördert, dass die Kreaturen auf neue Besucher zu gingen, um diese zu begrüßen.

Ein weiteres Ziel, das durch die Installation verfolgt wurde, war die Interaktion der Besucher untereinander. Es war zu beobachten, dass einander fremde Menschen auf den farbigen Flächen miteinander kommunizierten und sich über die Animationen austauschten. Ein weiteres Gesprächsthema waren das Verhalten der Kreaturen. So wurden Theorien zur Intelligenz aufgestellt und diese anschließend diskutiert und gemeinsam getestet.

Abbildung 3.4: Der Mann im Vordergrund wird als Schattenriss auf der Fassade abgebildet und kann dort mit den kleinen Kreaturen interagieren

3.1.3 Lösung

Wenn den Künstlern, oder generell den Nicht-Informatikern, die Fähigkeiten fehlen um Gestenerkennung selber zu implementieren, dann macht es Sinn ihnen diese Funktionen zur Verfügung zu stellen. Dabei muss vor allem auf die einfache Benutzbarkeit geachtet werden. Die Wahl fiel dabei auf eine Bibliothek für die Programmiersprache Processing. Diese Sprache setzt sich dadurch von der Masse ab, dass sie es sehr einfach ermöglicht, grafische

Effekte auf dem Bildschirm auszugeben. Aus diesem Grund wird sie häufig von Künstlern benutzt, die damit ihre Ideen relativ einfach und schnell in die Realität umsetzen können. Und da Processing auf Java basiert, lässt sich das gesamte Repertoire der Java-Möglichkeiten abrufen, was die Sprache wiederum sehr vielseitig einsetzbar macht.

Für die Zwecke der Bildverarbeitung eignet sich Processing, und damit eigentlich Java, allerdings nur bedingt. Java generiert nicht direkt Maschinenbefehle sondern nur den so genannten Bytecode, der dann wiederum in der JavaVM ausgeführt wird. Durch diese weitere Schicht zwischen Programm und CPU ist Java nicht so performant wie es Sprachen sind, die direkt Maschinenbefehle generieren. Aus diesem Grund wird der rechenintensive Teil der Gestenerkennung, also hauptsächlich die Bildverarbeitung, auf C++ ausgelagert. C++ deshalb, weil es hierfür mit OpenCV eine sehr umfangreiche Bibliothek gibt, die dem Anwender viele Probleme der Bildverarbeitung abnimmt.

Die Aufteilung auf zwei Programmiersprachen bringt allerdings nicht nur Vorteile mit sich. Java bzw. Processing zeichnet sich durch seine Plattformunabhängigkeit aus. C++ kann diese Eigenschaft leider nicht komplett für sich verbuchen und aus diesem Grund wird die im Rahmen dieser Bachelorarbeit entwickelte Lösung erst einmal nur mit Windows kompatibel sein. Allerdings sollte es später möglich sein, die Erweiterung auch auf Mac OS oder Linux zu portieren und dort zu nutzen.

3.2 Anforderungen

3.2.1 Performance

Da die Erweiterung genutzt werden soll, um mit ihrer Hilfe andere Software zu steuern, kommt es sehr auf die Performance an. Einem Benutzer fällt es schnell negativ auf, wenn seine Gesten immer erst mit einem deutlichen Zeitverzug angenommen werden und das System somit generell langsam reagiert. Wenn die Quelle des Problems die Gestenerkennung wäre, dann hätte der eigentliche Entwickler der Software keinerlei Chancen den Prozess noch zu beschleunigen.

Um die Erweiterung so performant wie möglich zu machen, werden die rechenintensiven Teile bereits in eine leistungsfähigere Programmiersprache ausgelagert. Allerdings muss auch hier weiterhin darauf geachtet werden, den Code auf Geschwindigkeit zu optimieren.

3.2.2 Einfache Benutzbarkeit

Wie bereits erwähnt, handelt es sich bei Processing um eine Programmiersprache, mit der es sehr einfach ist, grafische Ausgaben zu erzeugen. Das Codebeispiel 3.1 zeigt den Grundaufbau eines einfaches Processing-Programms, welches es ermöglicht, mit der Maus Linien auf eine Fläche zu malen. Sicherlich keine besonders komplizierte Anwendung, aber die Tatsache, dass hierfür nur wenige Zeilen Quellcode benötigt werden, verdeutlicht die Einfachheit dieser Sprache gut.

In dem Beispiel sind drei Methoden zu sehen.
Die setup() wird nur einmal beim Starten des Programms aufgerufen und ist dafür zuständig, die Ein- und Ausgaben zu initialisieren und Variablen anzulegen.
Die draw() wird in Endlosschleife immer wieder aufgerufen. Hier wird das Verhalten des Programms definiert.
Und zu guter Letzt die keyTyped(), welche nur dann aufgerufen wird, wenn der Benutzer eine Taste der Tastatur drückt. Gerade hier wird der Unterschied zu Java deutlich. Es bedarf keiner umständlichen EventListener-Konstruktion, um Tasten- oder Mausevents abzufangen, sondern alles läuft direkt über vordefinierte Methoden und Variablen, deren Funktionen dank Telling Names meist selbsterklärend sind.

```
void setup() {
  size(400, 200);  // neues Fenster erstellen
  background(255); // Hintergrundfarbe auf weiß setzen
}

void draw() {
  if (mousePressed) {  // wenn eine Maustaste gedrückt wird
    // zeichne eine Linie zwischen der aktuellen Mausposition (mouseXY)
    // und der vorherigen Position (pmouseXY)
    line(mouseX, mouseY, pmouseX, pmouseY);
  }
}

void keyTyped() {
  if (key == 'c') {  // wenn zwischendurch die Taste c gedrückt wurde
    background(255);  // dann mach den Hintergrund neu und lösch damit
        die Linien
  }
}
```

Listing 3.1: einfaches Beispiel der Programmiersprache Processing

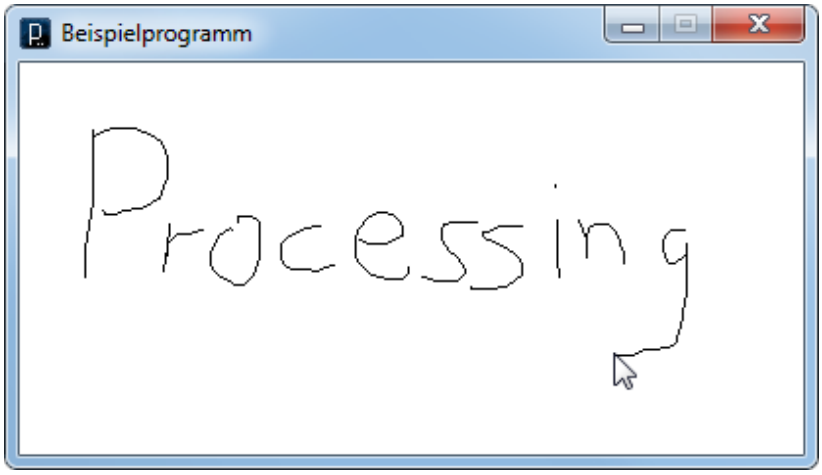


Abbildung 3.5: Das Beispielprogramm, welches in Listing 3.1 beschrieben wird

In diese Umgebung muss sich unsere Erweiterung nahtlos einfügen, weswegen besonders darauf zu achten ist, dass die Benutzung der Processing-Erweiterung mit möglichst wenigen Befehlen möglich ist.

3.3 Einschränkungen

Die Gestenerkennung in Kamerabildern zählt nicht gerade zu den trivialsten Problemen der Informatik. Im Laufe der Planung sind dabei einige Probleme aufgefallen, die sich nicht zufriedenstellend durch Software lösen lassen. Aus diesem Grund gibt es einige einschränkende Anforderungen, die erfüllt sein müssen, damit die Software fehlerfrei funktionieren kann.

3.3.1 Nur eine Person

Für die Gestenerkennung ist es nötig, die Hände einer Person im Bild zu erkennen und sie über die Zeit zu verfolgen. Das an sich ist schon keine einfache Aufgabe und es ist davon auszugehen, dass es noch deutlich komplizierter wird, wenn auf einmal mehr als zwei Hände im Bild zu sehen sind. Von daher wird es erst einmal ratsam sein, die Benutzerzahl auf Eins zu begrenzen.

3.3.2 Bekleidungs- und Hintergrundfarbe

Um die Hände im Bild zu finden, wird u. a. nach Bereichen mit hautähnlicher Farbe gesucht. Erste Tests haben dabei ergeben, dass auch Möbel aus hellem Holz als hautfarben erkannt werden. Und auch Kleidung in Orange- oder Gelbtönen können von der Hauterkennung fälschlicherweise erkannt werden. Aus diesem Grund ist es anzuraten einen Hintergrund zu wählen, bei dem möglichst kein helles Holz zu sehen ist und die Kleidung des Benutzers entsprechend anzupassen.

3.3.3 Lichtverhältnisse

Wie erwähnt, wird im Bild nach Farbtönen gesucht, die der menschlichen Haut ähneln. Die Wahrnehmung von Farben wird aber durch das Licht, mit dem die Umgebung beleuchtet wird, beeinflusst. Aus diesem Grund ist es zu erwarten, dass die Hauterkennung in verschiedenen Lichtsituationen unterschiedlich gut funktioniert. Von daher wird es nötig sein, dass die Kamera am Anfang auf die jeweilige Situation kalibriert wird. Weiterhin sollte darauf geachtet werden, dass sich die Lichtverhältnisse im Laufe der Benutzung nicht verändern.

3.3.4 Gestenbeginn

Da die Hand des Benutzers die ganze Zeit im Bild zu sehen sein wird und sie sich die ganze Zeit bewegt, wird es eine besondere Herausforderung werden, zu erkennen, wann eine Geste beginnt und wann sie endet. Um eine Bewegung deutlich als bewusste Geste zu kennzeichnen wird es notwendig sein, dass der Benutzer direkt vor- und nachher eine Pause von wenigen Sekunden einlegt. Anhand dieser Pause kann die Software dann besser bewusste von unbewussten Bewegungen unterscheiden und die entsprechenden Gesten zuordnen.

3.3.5 Abstand

Des Weiteren ist darauf zu achten, dass der Benutzer der Software einen bestimmten Abstand zur Kamera einhält. Ist der Abstand zur Kamera zu gering, so füllt der Körper das ganze Kamerabild aus. Eine sichere Erkennung der Gesten kann dann nicht mehr gewährleistet werden, weil schon kleinste Bewegungen der Hand zu großen Ausschlägen in der Software führen. Auch von einem zu großen Abstand ist abzuraten, weil in diesem Fall die

Hände im Kamerabild zu klein werden um noch erkannt werden zu können.
Aus den genannten Gründen ist ein Abstand zu Kamera von etwa 1 bis 2 Metern akzeptabel.

3.4 Bedürfnisse

Bei den Anforderungen an die Bibliothek für Gestenerkennung gibt es zwei verschiedene Sichtweisen.
Bei der Ersten, der inneren Sicht, ist der Künstler der Benutzer. Er wendet die Bibliothek an und hat so auch seine Erwartungen hinsichtlich der Benutzbarkeit.
Als Zweites ist da die äußere Sicht. Hier stellt der Künstler seine Software zur Schau. Der Benutzer ist in diesem Fall der Besucher der Ausstellung, der die Gestenerkennung anwendet. Die Ansprüche des Künstlers betreffen hier eher die Performance der Bibliothek.

3.4.1 Innere Sicht

Bei der inneren Sicht ist der Künstler der Anwender, dem die Bibliothek bereit gestellt wird. Er wendet sie nur an, ohne wissen zu müssen was im Hintergrund passiert. Dabei kommt es ihm, wie bereits in Kapitel 3.2.2 erwähnt, vor allem auf die einfache Benutzbarkeit an. Dies kommt auch daher, dass sich die Herangehensweise eines Künstlers von dem eines Informatikers unterscheidet. Ein Informatiker beginnt an Anfang eines Projekts (im Idealfall) erst einmal mit der Planung und erstellt dazu detaillierte UML-Diagramme, die den Aufbau und das Verhalten seines Programms beschreiben.
Der Künstler zieht eher die prototypbasierte Entwicklung vor. Er beginnt mit der Programmierung und überprüft dabei fortlaufend die Ergebnisse. Dabei werden so lange diverse Einstellungen ausprobiert, bis die Resultate den Vorstellungen des Künstlers entsprechen. Dieser Weg ist zwar nicht per se schlecht, aber er birgt auch die Gefahr, dass Probleme auftreten, die durch eine gründliche Planung verhindert worden wären. Allerdings spielt dieser Nachteil eher eine untergeordnete Rolle, weil sich aus jedem Fehlverhalten des Programms eine neue kreative Idee entwickeln kann.
Aus diesem Grund wurde bereits die Entscheidung für Processing getroffen. Diese Sprache ist sehr unkompliziert aufgebaut und unterstützt damit die Arbeitsweise der Künstler. Sie ermöglicht es ihm, sehr einfach Änderungen durchzuführen oder ohne viel Aufwand eine neue Idee auszuprobieren.

Ein weiterer Anspekt der Anforderungen ist, dass der Künstler in seinem kreativen Prozess nicht eingeschränkt werden darf. Von daher ist es sinnvoll, dass er nicht nur Zugriff auf das Endergebnis (die gefundene Geste) hat, sondern auch auf die Zwischenschritte. Hierunter fallen etwa die Positionen der Hände oder die Sequenz der Bewegungen. Da diese Daten ohnehin anfallen, ist es kein großes Problem sie verfügbar zu machen.

3.4.2 Äußere Sicht

In der äußeren Sicht bietet der Künstler seine Software an und ist damit in der Rolle des Dienstleisters. Der Benutzer ist derjenige, der sich das Kunstwerk ansieht, bzw. der damit interagiert.

Als Kunstinstallation könnte man sich nun, ähnlich wie bei Scott Snibbe, eine Leinwand vorstellen, auf der Animationen ablaufen. Tritt ein Besucher vor diese Leinwand, wird sein Schattenriss sichtbar und die Animationen reagieren auf seine Anwesenheit. Anstatt, dass der Besucher die Animationen nur mit seinem Schatten beeinflussen kann, ist es nun möglich sie durch gezielte Gesten zu steuern. So ließen sich z. B. Objekte auf der Leinwand durch eine einfache Zeigegeste dirigieren oder manipulieren.

Da sich der Künstler in diesem Fall in einer anderen Rolle befindet, hat er auch andere Anforderungen. Es ist z. B. sehr wichtig, dass die Bibliothek stabil läuft und es im Laufe der Ausstellung nicht zu Abstürzen kommt. Außerdem muss es möglich sein, die Anwendung ohne Kenntnisse des Systems in Betrieb zu nehmen, damit diese Tätigkeit auch von den Mitarbeitern des Ausstellungshauses ausgeführt werden kann. Dabei muss z. B. die Initialisierung der Kamera vollständig automatisch ablaufen.

3.5 Fazit

Das Thema kamerabasierte Gestenerkennung taucht als Konzept der Mensch-Maschinen-Interaktion immer wieder auf, konnte sich aber bisher nicht gegen die etablierten Konzepte durchsetzen. Dies liegt unter anderem daran, dass die Umsetzung recht kompliziert ist. Ziel ist es, die Funktionalität der Gestenerkennung als Bibliothek bereitzustellen, damit sie von Menschen mit weniger Programmiererfahrung genutzt werden kann. Da es sehr auf die einfache Benutzbarkeit ankommt, fiel die Entscheidung auf die Programmiersprache Processing. Diese ist sehr einfach aufgebaut und ermöglicht es auch Programmieranfängern, schnell zu brauchbaren Ergebnisse zu kommen.

4 Algorithmen

4.1 Allgemein Ablauf

Der allgemeine Ablauf für die Gestenerkennung in Kamerabildern beginnt damit, dass das aktuelle Bild der Kamera abgefragt werden muss. Dieses Bild wird dann vorverarbeitet, indem gegebenenfalls der Kontrast erhöht wird, um die folgenden Schritte einfacher und fehlerfreier zu gestalten.

Danach kommt die Merkmalsextraktion, bei der die gewünschten Objekte im Bild gesucht werden. Hier werden nun also die Hände im Bild gesucht und ihre jeweiligen Positionen werden bestimmt. Dafür existieren einige Verfahren, die im Folgenden noch weiter erläutert werden. Wurden Stellen gefunden, bei denen es sich potenziell um Hände handelt, kommt die Merkmalsreduktion, bei der entschieden wird, ob ein Treffer eine Hand darstellt, oder ob es sich um eine Falschmeldung handelt.

Danach wird überprüft, ob sich eine Hand in mehreren aufeinander folgenden Bildern wiederfindet. Ist dies der Fall, so kann untersucht werden, ob sie eine Bewegung ausgeführt hat, die einer Geste entspricht.

4.2 Handerkennung

Um erkennen zu können, welche Geste der Benutzer ausführt, muss erst einmal erkannt werden, wo sich die Hand im Bild befindet und wie sie sich bewegt. Da dieses Problem nicht erst seit dieser Ausarbeitung existiert gibt es bereits einige Lösungsansätze. Im Folgenden sind einige Methoden aufgeführt, die untersucht wurden.

4.2.1 Differenzbilder

Ein sehr einfaches und häufig eingesetztes Mittel um Bewegungen zu erkennen, ist das Erstellen von Differenzbildern. Dafür benötigt man das aktuelle Bild der Kamera und das vorherige Bild, welches zuvor zwischengespeichert wurde.

In OpenCV gibt es nun eine Methode, die für jeden Pixel die Differenz der beiden Bilder berechnet. Dies wird erreicht, indem die Grauwerte aller Pixel des gepufferten Bildes von den Grauwerten der Pixel des aktuellen Bildes abgezogen werden. Da es bei dem Verfahren nur um den Unterschied der beiden Pixel geht und nicht darum, welcher von beiden nun größer oder kleiner ist, wird am Ende noch der Betrag der Differenz gebildet.

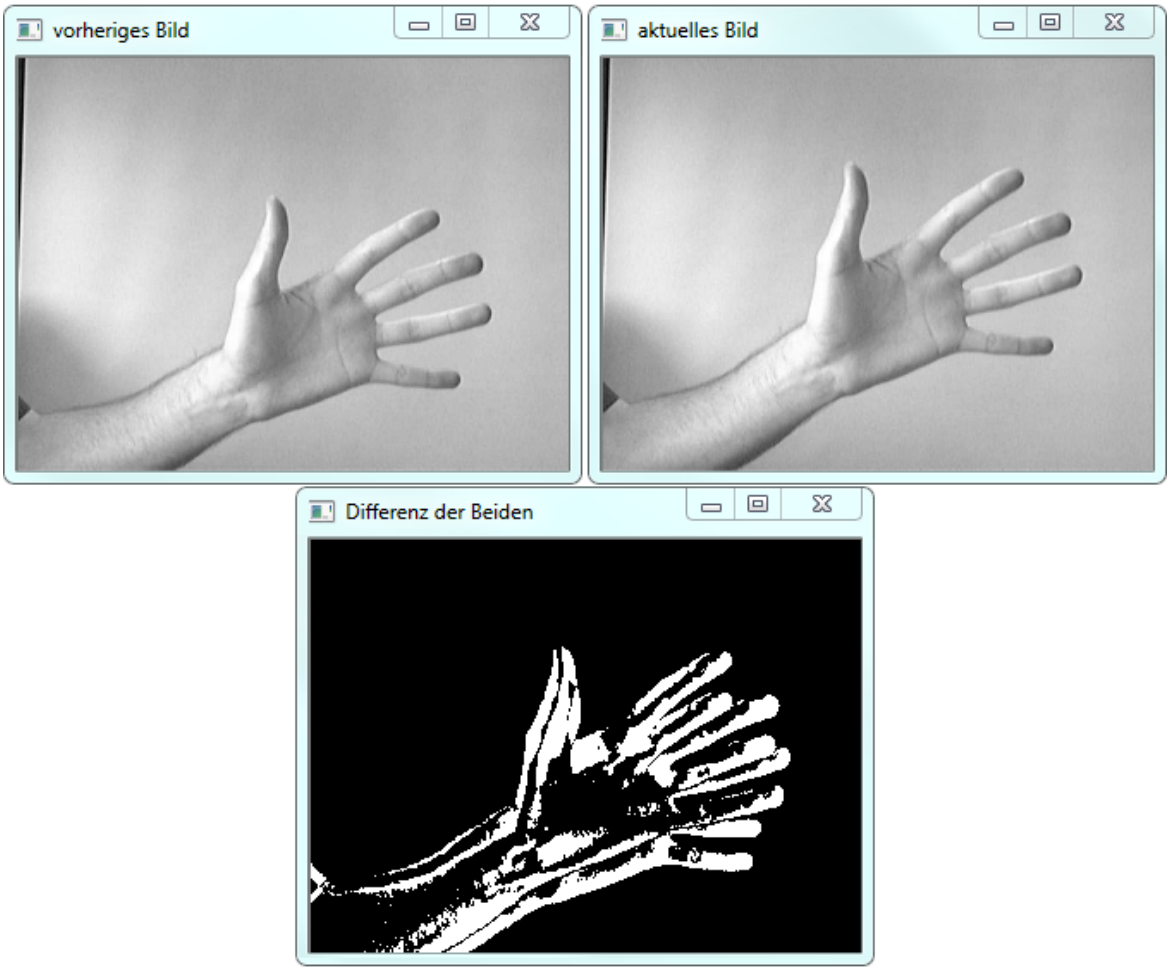


Abbildung 4.1: oben: die Grauwert-Bilder von der Kamera
unten: die Differenz der beiden Bilder

Anhand dieser Zahl kann man dann ablesen, ob sich an der entsprechenden Stelle im Bild etwas bewegt hat oder nicht. In einer idealen Welt wäre die Differenz Null, wenn sich an dieser Stelle nichts verändert hat. Allerdings trifft diese Annahme auf unsere reale Welt nicht zu. Da eine Kamera immer auch etwas rauscht, unterscheiden sich die Grauwerte der Pixel häufig auch dann, wenn sich an der Stelle nichts bewegt hat. Aus diesem Grund legt man einen Grenzwert fest und alles, was kleiner ist, wird definiert als „keine Bewegung“. Bei einer großen Differenz hat sich der Grauwert an dieser Stelle von einem Bild zum Anderen stark verändert, was wiederum auf eine Bewegung im Bild schließen lässt.

In Abbildung 4.1 ist das zu erkennen. Oben sind die Grauwert-Bilder von der Webcam zu sehen und bei näherer Betrachtung fällt auf, dass die Hand im aktuellen Bild (rechts) weiter oben ist als im vorherigen (links). Das bedeutet, dass sich die Hand nach oben bewegt hat. Im unteren Bild ist die Differenz der beiden oberen Bilder abgebildet. Dabei wurde diese Differenz bereits in eine Binärmaske umgewandelt, was bedeutet, dass es für jeden Pixel nur den Wert Eins oder Null, bzw Weiß oder Schwarz, gibt. Ein weißer Pixel steht also für eine Bewegung an dieser Stelle, ein schwarzer für keine Bewegung.

Anhand von Differenzbildern lassen sich Bewegungen in Bildern sehr gut erkennen, allerdings hat es in unserem Fall den Nachteil, dass auch die Bewegung der Kleidung und der Schatten einer Person erkannt wird. Eine sichere Erkennung der Hände im Bild ist mit dieser Methode also nicht möglich.

4.2.2 Hauterkennung

Eine weitere Möglichkeit, wie man die Hände des Benutzers erkennen kann, ist die Hauterkennung. Im Grunde bedeutet das nur, dass im Bild nach Regionen gesucht wird, die von der Farbe her der menschlichen Haut ähneln. Da auch die Hauterkennung kein neues Problem ist, existieren bereits einige Lösungsansätze und auch schon fertig implementierte Lösungen sind zu finden.

Die verschiedenen Algorithmen arbeiten dabei meist nach einem ähnlichen Prinzip. Hauptsächlich unterscheiden sie sich dadurch, dass sie auf verschiedenen Farbräumen, also der Art wie eine Farbe im Computer beschrieben wird, beruhen.

Ein weit verbreiteter Ansatz der Hauterkennung basiert dabei auf dem HSV-Farbraum (vgl. [Oliveira und Conci, 2009]). In diesem Farbraum werden Farben durch ihren Farbwert (*englisch hue*), ihrer Sättigung (*saturation*) und ihrer Helligkeit (*value*) beschrieben. Das ist des-

halb von Vorteil, weil die verschiedenen Hauttöne in diesem Farbraum einen ähnlichen Farbwert besitzen und sich hauptsächlich nur in der Farbsättigung unterscheiden.

Ein weiterer Algorithmus, der auch in der frei verfügbaren OpenCV-Erweiterung OpenCVx implementiert ist, basiert z. B. auf dem YCbCr-Farbmodell (vgl. [Hsu u. a., 2002]). Bei diesem wird eine Farbe nicht durch ihre Rot-, Grün- und Blauanteile beschrieben sondern durch ihre Helligkeit und ihre Rot- und Blauanteile.

Am besten funktionierte aber der Algorithmus, der auch in [Kovac u. a., 2003] beschrieben wird und der ebenfalls bereits in OpenCVx implementiert wurde. Für jeden einzelnen Pixel wird dabei die Abfrage aus dem Codebeispiel 4.1 durchgeführt. Es wird überprüft, ob die Farbanteile des Pixels in einem bestimmten Bereich liegen und ob der Rotanteil stärker ist als die Blau- und Grünanteile. Durch das richtige Verhältnis der Farbanteile zueinander kann so relativ zielsicher erkannt werden, ob es sich bei dem entsprechenden Pixel um einen hautfarbenen handelt oder nicht.

```
if( rot > MIN_ROTANTEIL &&
    gruen > MIN_GRUENANTEIL &&
    blau > MIN_BLAUANTEIL && // Farbanteile sind in ausreichender Menge
       vorhanden
    MAX( rot, gruen, blau ) - MIN( rot, gruen, blau ) > MIN_ABSTAND &&
       // Farbanteile sind weit auseinander - bei zu geringem Abstand
       erscheint der Pixel grau
    abs( rot - gruen ) > MIN_DIFFERENZ && // Rot- und Grünanteile dürfen
        nicht zu dicht beieinander liegen
    rot > gruen && rot > blau ) // Rot ist am stärksten vertreten
{
    // markiere diesen Pixel als Hautfarbe
}
```

Listing 4.1: Abfrage ob ein Pixel hautfarben ist oder nicht

Ein Vergleich der drei beschriebenen Algorithmen ist in Abbildung 4.2 zu sehen. Gezeigt wird dort oben links die Farberkennung im HSV-Raum, oben rechts in der YCbCr-Farbdarstellung und unten die Variante im RGB-Raum. Es ist deutlich zu erkennen, dass der Algorithmus, der auf dem RGB-Raum basiert, in dieser Situation die besten Ergebnisse liefert. Daraus lässt sich zwar keine Allgemeingültigkeit ableiten, aber es ist hilfreich bei der Entscheidungsfindung.

Die Hauterkennung erkennt Hände und Gesicht des Benutzers zuverlässig. Es offenbaren sich allerdings auch Nachteile. So haben z. B. Möbel aus hellem Holz eine ähnliche Farbe

wie die menschliche Haut und damit werden sie ebenfalls erkannt. Dem kann nur begegnet werden, indem man einen Hintergrund verwendet, in dem möglichst keine holz- oder hautähnlichen Farben vorkommen (siehe auch Kapitel 3.3.2).

Des Weiteren existiert eine starke Beeinflussung durch das Umgebungslicht, weil dadurch auch die Farben im Bild anders erscheinen. Das kann dazu führen, dass z. B. bei blaustichigem natürlichem Licht keine hautfarbenen Bereiche mehr gefunden werden können. Das farbige Licht kann zu Teilen über den Weißabgleich der Kamera ausgeglichen werden. Standardmäßig regelt die Kamera, bzw. deren Treiber, den Weißabgleich automatisch und liegt damit auch in den meisten Fällen richtig. In einigen Situationen, z. B. wenn sich natürliches Licht und Neonlicht mischen, kann ein manuelles Nachjustieren des Weißabgleichs zu besseren Ergebnissen führen.

Abbildung 4.2: Die verschiedenen Algorithmen zur Hauterkennung

4.2.3 Formerkennung

Darüber hinaus bietet OpenCV auch die Möglichkeit nach bestimmten Formen im Bild zu suchen. Formerkennung ist allerdings ungleich komplexer und damit rechenaufwändiger als die bisher vorgestellten Verfahren. Der Rechenaufwand macht sich vor allem in einem starken Rückgang der Framerate bemerkbar. Da die Gestenerkennung möglichst in Echtzeit erfolgen soll, wurde diese Alternative dann verworfen.

4.2.4 Objekterkennung

Des Weiteren gibt es noch die Möglichkeit der Objekterkennung. Verbreitete Algorithmen für diese Aufgabe sind z. B. SIFT (*Scale-Invariant Feature Transform*) [Lowe, 2004] oder dessen Nachfolger SURF (*Speeded Up Robust Features*) [Bay u. a., 2006]. Bei Beiden werden markante Punkte im Bild gesucht und dann überprüft, ob es sich dabei um eine Hand handelt. Wie zumindest der Name von SIFT bereits andeutet, sind beide Verfahren dabei unabhängig von Skalierung und Drehung des Objekts.

Allerdings ist keines der beiden genannten Verfahren in den Standardbibliotheken von OpenCV bereits enthalten und der Aufwand, es selber zu implementieren, würde den Rahmen dieser Bachelorarbeit sprengen. Aus diesem Grund wird von dieser Art der Handerkennung abgesehen, auch wenn die Chance bestünde, am Ende eine sehr performante und fehlerfreie Erkennung zu erhalten.

4.2.5 Lösung

Als beste Lösung hat sich eine Mischung aus Haut- und Bewegungserkennung herauskristallisiert. Erst wird im Bild nach hautfarbenen Flächen gesucht und die entsprechenden Bereiche werden in einer Binärmaske markiert. In dieser wird jeder hautfarbenen Pixel mit einer Eins markiert und alle anderen Pixel mit einer Null. Danach wird dann in einer zweiten Binärmaske markiert, wo Bewegung festgestellt wurde.

Die beiden Masken sind im oberen Teil der Abbildung 4.3 zu sehen. Eine Eins ist dort durch einen weißen Pixel gekennzeichnet, eine Null durch einen schwarzen. Wie zu erkennen ist, zeigt sich bei der Hauterkennung erneut das Problem, dass teilweise auch Hintergründe erfasst werden. Obwohl ein weißer Hintergrund verwendet wurde, erscheint die Fläche durch die Beleuchtung durch Glühlampen leicht orange und wird deshalb als Haut markiert.

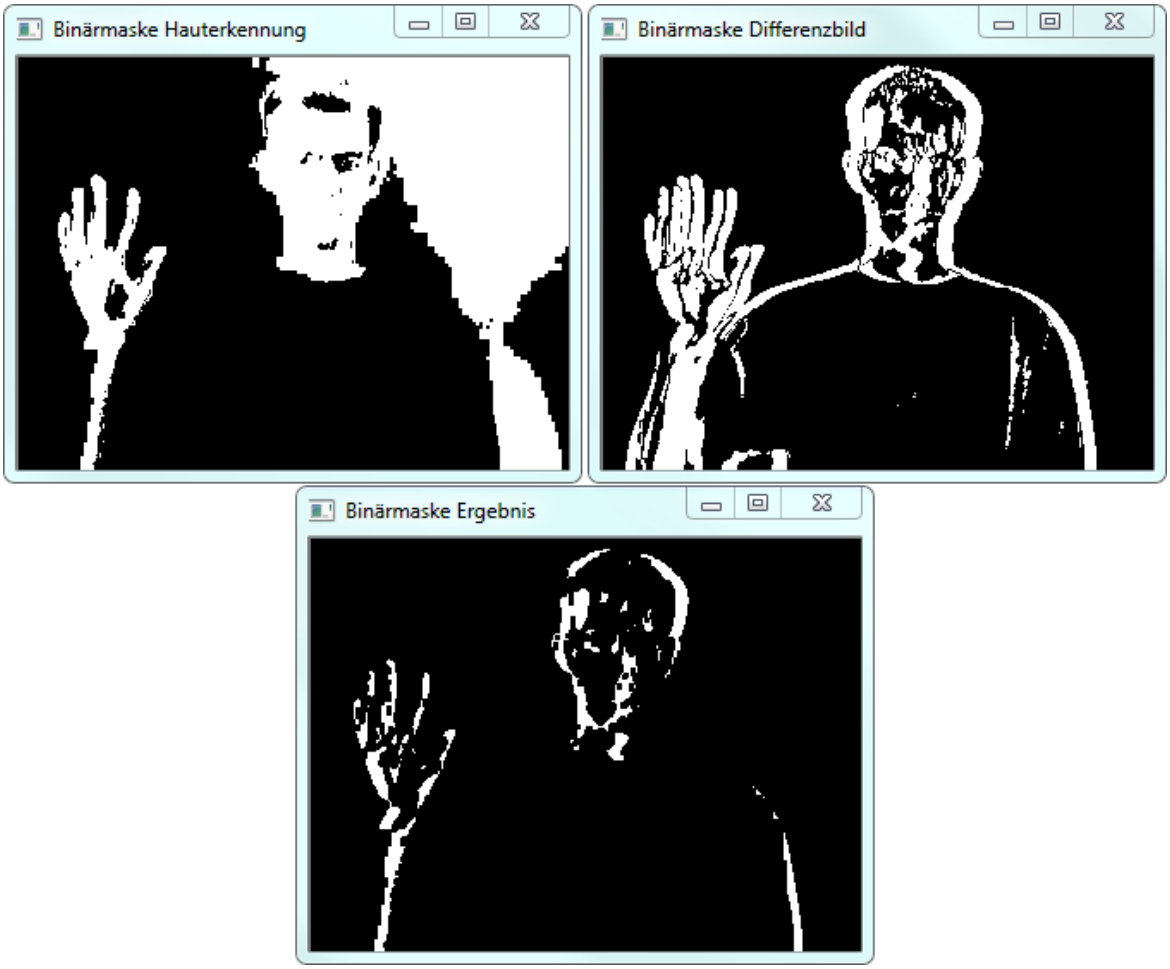


Abbildung 4.3: oben: die Binärmasken der Haut- und Bewegungserkennung
unten: die resultierende Maske nach der UND-verknüpfung

Aus diesen beiden Binärmasken wird zum Schluss eine weitere Maske erstellt, indem die beiden schlicht logisch UND-verknüpft werden. Das heißt, dass das Ergebnis Eins ist, wenn die jeweiligen Pixel in den anderen beiden Masken ebenfalls Eins sind, sonst Null. Zu sehen ist das Ergebnis im unteren Teil in Abbildung 4.3.

Auf diese Art und Weise sind am Ende nur die Pixel markiert, die hautfarben sind und an deren Stelle Bewegung festgestellt wurde. Der fälschlicherweise als hautfarben erkannte Hintergrund wird am Ende nicht mehr erfasst, weil an dieser Stelle keine Bewegung statt findet. Ausnahme ist der linke Ärmel, der sich vor dem vermeintlich hautfarbenen Hintergrund bewegt. Da in dieser Region Bewegung und Hautfarbe auftreten ist es nicht zu verhindern, dass der Ärmel ebenfalls erfasst wird.

Abbildung 4.4: Hauterkennung ohne (links) und mit (rechts) Dilatationsfilter

Des Weiteren entstanden auch durch das Kombinieren der beiden Binärmasken weitere Schwierigkeiten, die es zu lösen galt. So trat bei der Haut- und der Bewegungserkennung jeweils das Problem auf, dass eine Hand nicht als Ganzes erkannt wurde, sondern in mehrere Flächen aufgeteilt wurde. Die Abbildung 4.4 zeigt dieses Phänomen am Beispiel der Hauterkennung. Im linken Teil des Bildes ist der Arm in mehrere hautfarbene Flächen unterteilt. Abhilfe wurde durch die Bearbeitung der einzelnen Flächen mit einem Dilatationsfilter geschaffen. Dieser bewirkt, wie ein Maximumfilter, dass die einzelnen Bereiche vergrößert werden und sich somit hinterher berühren. Das Ergebnis ist im rechten Teil der Abbildung 4.4 zu sehen. Aus den vielen einzelnen Flächen ist nun eine große geworden, die den gesamten Arm beinhaltet.

Abbildung 4.5: Das Ergebnis aus Abb. 4.3 mit einem Dilatationsfilter bearbeitet

Zur Verdeutlichung ist in Abbildung 4.5 noch einmal das mit einem Dilatationsfilter bearbeitete Ergebnis aus Abbildung 4.3 zu erkennen. Im ursprünglichen Bild ist deutlich zu sehen, dass die Hand und der Kopf jeweils in bis zu 20 einzelne kleine Flächen aufgeteilt wurde. Nach der Bearbeitung ist aus Hand und Kopf jeweils eine gleichmäßige Fläche geworden, die sich viel besser zur Weiterverarbeitung eignet als dutzende Einzelne.

4.3 Zuordnung

Dank der verschiendenen Verfahren existiert nun also eine Binärmaske, in der die Stellen markiert sind, an denen eine Bewegung stattfindet und an der hautähnliche Farben gefunden wurden (siehe Abb. 4.5). Wenn man sich nun aber eine Hand im Bild vorstellt, dann stellt man relativ schnell fest, dass diese aus mehreren Pixeln besteht. Es gilt nun also herauszufinden welche Pixel zu einer Hand gehören und welche eventuell nur in der Maske markiert sind, weil es Fehler bei der Hauterkennung gab.

Auch für diese Aufgabe liefert OpenCV bereits fertige Hilfestellungen. Eine davon sucht in der Binärmaske nach zusammenhängenden Pixeln mit einer Eins und merkt sich dann die Kontur, also den Umriss, dieses Bereichs. Auf Grund der Dilatationsfilter lässt sich in unserem Fall an der Form der Kontur nicht erkennen, ob es sich um eine Hand oder ein Gesicht handelt, aber es lässt eine Abschätzung der Größe zu, um Hände von einfachem Rauschen zu trennen. Die kleineren Bereiche werden von nun an ignoriert und es wird nur noch mit den größeren gearbeitet. Im Idealfall bleiben nur noch drei Flächen übrig, eine für den Kopf und zwei für die Hände.

Da uns die genaue Form der Kontur nicht weiter hilft, wird nun ein anderer Weg benötigt, um an Informationen über den Bereich zu gelangen. Wieder einmal hilft OpenCV da mit einer seiner vielen Funktionen aus. Diesmal ist es die Möglichkeit, von einem Bereich den Schwerpunkt zu berechnen. Zwar wird so nicht der genaue Mittelpunkt der Hand gefunden, aber zumindestens nähert sich das Ergebnis diesem an.

In Abbildung 4.6 ist links das Kamerabild und rechts die daraus berechnete Binärmaske zu sehen. Der Schwerpunkt wurde jeweils als blauer Kreis markiert. Zur Verdeutlichung wurde mit einem Rechteck der Bereich eingegrenzt, der berücksichtigt wurde. Dabei ist zu sehen, dass der Schwerpunkt der Maske ziemlich genau mit dem Mittepunkt der Hand überein stimmt. Dies ist allerdings nicht immer der Fall. In einigen Situationen kann es sein, dass der Arm zu einem größeren Teil in die Binärmaske einfließt. Damit verschiebt sich der Schwerpunkt weiter in Richtung Handgelenk. Trotzdem handelt es sich bei der Berechnung des

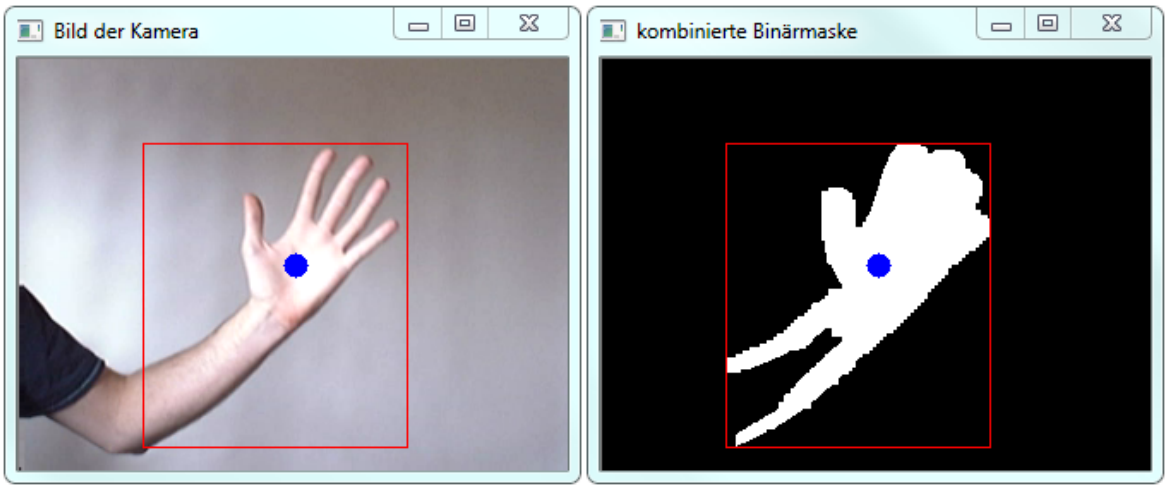


Abbildung 4.6: Links das Bild von der Kamera, rechts die Binärmaske. Die gefundene Fläche wurde umrandet und der Schwerpunkt wurde markiert

Schwerpunkts um eine ausreichend gute Annäherung an den eigentlichen Mittelpunkt der Hand.

Neben der Berechnung des Schwerpunkts gibt es noch weitere Möglichkeiten, um einen markanten Punkt der Fläche zu finden. Die wohl einfachste Option ist es, den Mittelpunkt des umschließenden Rechtecks zu berechnen. Da dies mit zwei Divisionen zu erledigen ist (Breite/2 und Höhe/2), ist das wohl die Möglichkeit mit dem geringsten Rechenaufwand. Allerdings zeigt sich hier, dass es auch deutlich weniger genau ist.
Eine weitere Möglichkeit ist das Berechnen der Maxima in X- und Y-Richtung. Dabei wird für jede Zeile und für jede Spalte gezählt, wie viele weiße Pixel sie enthalten. Der Punkt wird dann an die Stelle gesetzt, an der sich die Zeile und die Spalte mit den meisten weißen Pixeln schneiden. Diese Methode ist deutlich besser als die des Mittelpunkts. Der Nachteil ist aber, dass der berechnete Punkt sehr instabil ist und häufig seine Position innerhalb der Hand ändert. Wenn die Hand z. B. eine gerade Bewegung nach rechts macht, dann bilden die Punkte keine gerade Linie, sonders es ergibt sich eine Zick-Zack-Linie, weil der Punkt seine Position immer wieder ändert und vom oberen in den unteren Teil der Hand springt.
Diese beiden Verfahren sind zwar weniger rechenaufwändig als die Berechnung des Schwerpunkts, zu Gunsten der Genauigkeit wird der größere Rechenaufwand aber in Kauf genommen.

Für die weitere Verarbeitung sind die genauen Konturen nicht mehr relevant und es wird nur noch mit den berechneten Schwerpunkten gearbeitet. Um eine Geste erkennen zu können, muss nun eine Bewegung über einen gewissen Zeitraum erfasst werden. Dabei reicht es,

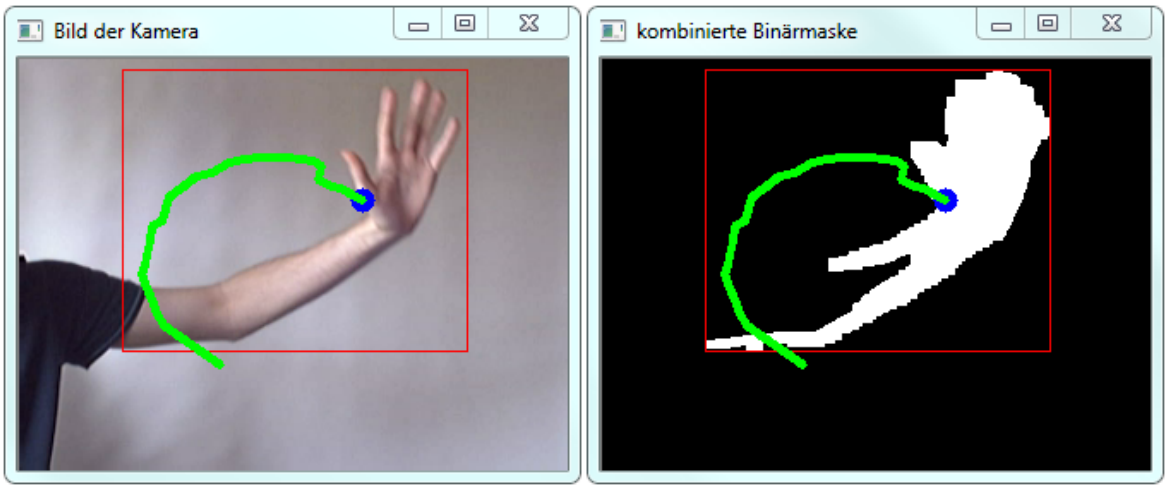


Abbildung 4.7: selber Aufbau wie 4.6. Die letzten 30 Mittelpunkte wurden zu einer Linie verbunden

die letzten n Mittelpunkte einer Hand zu speichern, um ihre Bewegung nachvollziehen zu können. Abbildung 4.7 zeigt was dabei heraus kommt, wenn man die letzten 30 Mittelpunkte speichert und sie mit Linien verbindet.

Das Bild zeigt außerdem das bereits angesprochene Problem, dass es sich beim Schwerpunkt nur um eine Annäherung an den Mittelpunkt der Hand handelt. Der berechnete Schwerpunkt weicht deutlich vom Zentrum der Hand ab. Aus diesem Grund kommt es auch am Ende der Kreisbewegung zu einem abrupten Sprung nach unten. Trotzdem ist die Bewegung der Hand gut wiederzuerkennen.

Da in dem Bild nur eine Hand zu sehen ist, existiert auch nur eine Liste mit Punkten. Wenn jetzt aber eine zweite Hand ins Bild kommt, dann werden auch zwei Mittelpunkte gefunden. Die Herausforderung liegt nun an der Stelle der Zuordnung. Einer der Punkte gehört in die bereits bestehende Liste, der andere in eine Neue. Dabei muss sichergestellt werden, dass der Mittelpunkt der rechten Hand nicht in die Liste der linken Hand oder umgekehrt eingeordnet wird. Gelöst wird diese Problem, indem einzelne Punkte immer in die Liste eingeordnet werden, zu der der Abstand am geringsten ist.

4.4 Gestenerkennung

Jetzt existieren also mehrere Listen mit Punkten, die die Bewegung der Hände repräsentieren. Als letzter Schritt muss jetzt noch herausgefunden werden, ob mit dieser Bewegung eine Geste gemeint war oder ob es sich um unbeabsichtigte Bewegungen handelt.

4.4.1 Start-/Stopp-Erkennung

Diese Unterscheidung, ob die Bewegung einer Geste entspricht oder nicht, ist eines der Hauptprobleme bei kamerabasierter Gestenerkennung. Einfacher hat man es da bei Touchscreens, bei denen es einen definierten Start- und Endpunkt einer Geste gibt. Und zwar genau dann, wenn der Benutzer den Touchscreen berührt bzw. wieder los lässt. Da die Hand allerdings die ganze Zeit im Kamerabild zu sehen ist, muss der Computer auf anderem Wege feststellen, ob die Bewegung der Hand bereits zu einer Geste gehört oder ob sie eher unbewusst geschieht.

Das Problem der Start-/Stopp-Erkennung existiert nicht nur bei kamerabasierten Gesten, bei denen die Hände verfolgt werden müssen. Auch beim sogenannten Eye-Tracking, also dem Verfolgen des Blicks des Benutzers, muss der Computer entscheiden, ob der Benutzer mit seinem Blick ein Ereignis auslösen möchte oder ob er sich ausschließlich umschaut. Hier nennt sich dieses Problem „Midas Touch Problem“ (vgl. [Istance u. a., 2008]) und es lassen sich einige Lösungen in der Literatur finden. Allerdings sind nicht alle auf die kamerabasierte Gestenerkennung übertragbar, weswegen hier nur auf einige relevante eingegangen wird.

Als mögliche Problemlösung wird immer wieder genannt, dass die Geste durch einen Knopf eingegrenzt wird. Der Benutzer würde während seiner Geste einen Knopf gedrückt halten und somit dem Computer eine einfache Identifizierung von Start- und Endpunkt ermöglichen. Dabei könnte es sich auch um einen Fußschalter handeln, so dass der Benutzer weiterhin beide Hände für seine Gesten zur Verfügung hat. Allerdings bedeutet diese Art der Bedienung einen höheren Lernaufwand, da der Benutzer zusätzlich zu den Gesten noch die Koordinierung mit dem Schalter lernen muss. Aus diesem Grund wird von dieser Möglichkeit der Start-/Stopp-Erkennung hier abgesehen.

Des Weiteren findet man in der Literatur häufiger den Ansatz, dass eine Geste durch spezielle Start- und Stopp-Gesten eingegrenzt wird. Dabei könnte man sich z. B. vorstellen, dass der Benutzer kleine Kreise mit der Hand macht, um eine Geste zu starten und/oder zu beenden. Der Computer würde dann die Hand verfolgen und überprüfen, ob sie dabei mehrere,

nahezu konzentrische, Kreise macht. Anstatt den Beginn einer Geste durch Bewegung zu markieren, könnte man dies aber auch durch Stillstand machen. Eine kurze Wartepause vor einer Geste könnte ebenfalls relativ einfach vom System erkannt werden. Aber auch bei diesem Ansatz besteht das Problem, dass dem Benutzer erklärt werden muss, wie er das System bedienen kann.

Es zeigt sich also, dass die Erkennung von Beginn und Ende einer Geste nicht ganz trivial ist und dass auch die Gegenmaßnahmen nicht immer praktikabel sind.

4.4.2 Wischgesten

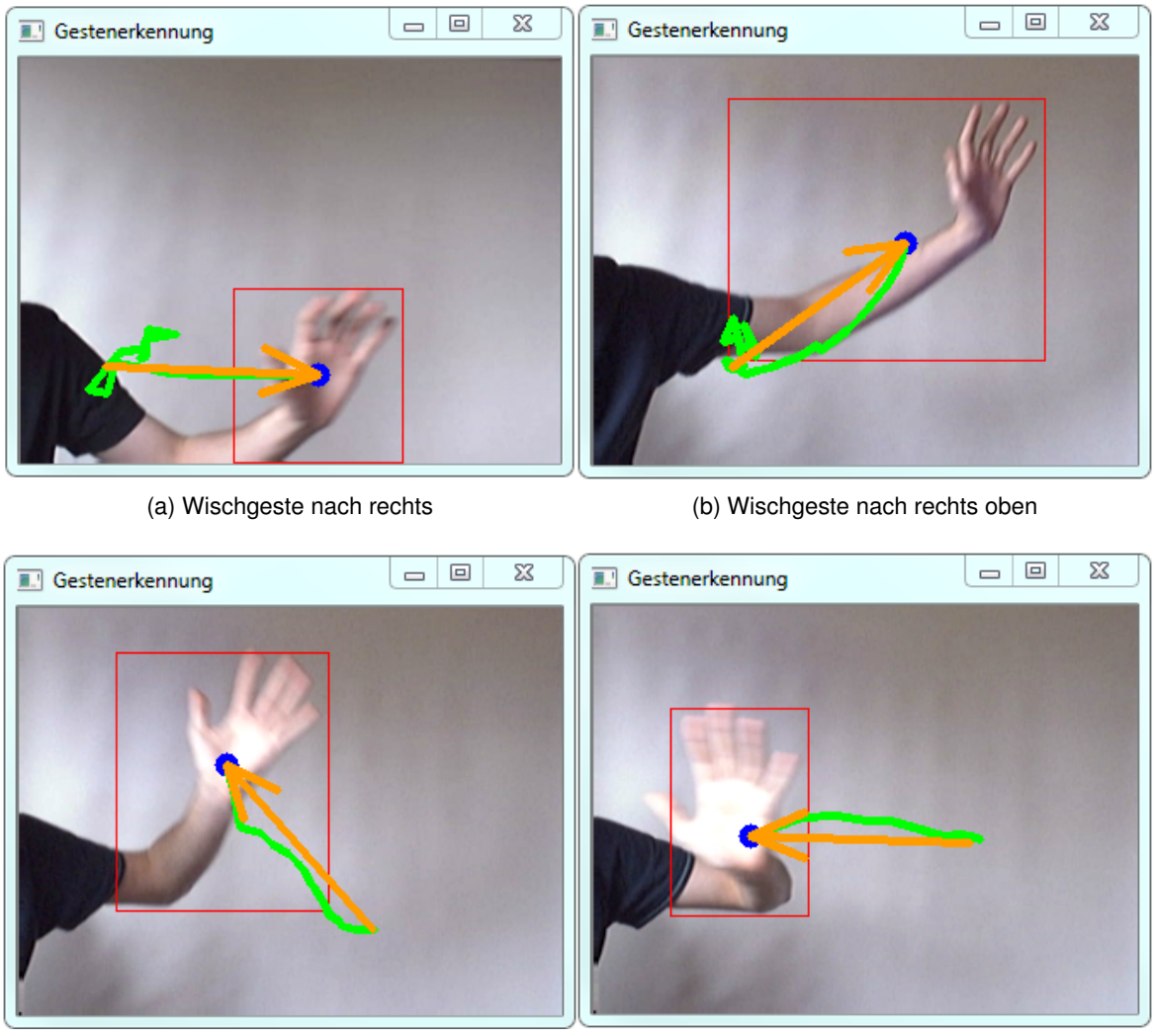
Da es im Rahmen dieser Bachelorarbeit hauptsächlich darum geht relativ einfache Gesten zu erkennen, spielen die Probleme der Start-/Stopp-Erkennung von Gesten keine so große Rolle. Im Grunde wird das Problem durch die Wahl der zu erkennenden Gesten gelöst.

In den vorangegangenen Schritten wurden Hände und Kopf im Bild lokalisiert und der jeweilige Schwerpunkt wurde in je einer Liste gespeichert. Damit existieren drei Listen die die jeweils letzten n Punkte beinhalten. In Tests hat sich ergeben, dass 30 ein guter Wert für n ist, da dabei genug Punkte gespeichert werden, um die Bewegung nachvollziehen zu können.

In erster Linie soll aus der Sequenz von Punkten im Folgenden herausgefunden werden, ob eine einfache Wischgeste gemacht wurde. Diese zeichnet sich dadurch aus, dass die Hand einer Geraden folgt. Vereinfacht gesagt genügt es also zu schauen, ob der erste und der letzte Punkt in einer Liste eine bestimmte Distanz voneinander entfernt sind. Sind sie weit genug auseinander so kann darauf geschlossen werden, dass die Hand sich in der Zwischenzeit auf mehr oder weniger direktem Weg zwischen diesen Punkten bewegt hat. Dabei zeigen Tests, dass diese Annahme auch zu brauchbaren Ergebnissen führt.

Die Richtung der Geste wird nun bestimmt, indem über alle Punkte in der Liste eine Ausgleichsgerade gebildet wird. Dabei steht OpenCV mit mehreren implementierten Algorithmen helfend zur Seite. Auf Grund der Anforderung an die Performance, fällt die Wahl hierbei auf die „Methode der kleinsten Quadrate“. Dieser Algorithmus ist relativ einfach und effizient bei der Berechnung einer Ausgleichsgeraden über die Punkte in der Sequenz.

Um die Ergebnisse noch zu verbessern, hat es sich als hilfreich herausgestellt, nur eine Teilmenge der gespeicherten Sequenz zu betrachten und nicht die komplette Sequenz vom Ersten bis zum Letzten. Am einfachsten war dabei der iterative Ansatz, bei dem untersucht wird, ob der erste und der zweite Punkt in der Liste weit genug auseinander liegen, um eine Wischgeste darzustellen. Wenn ihre Distanz zu gering ist, dann wird der erste mit dem



(a) Wischgeste nach rechts

(b) Wischgeste nach rechts oben

(c) Wischgeste nach links oben

(d) Wischgeste nach links

Abbildung 4.8: Wischgesten

dritten Punkt verglichen, und so weiter. Dieser Mehraufwand an Berechnungen verbessert die Erkennungsrate noch einmal deutlich.

Die resultierende Erkennung ist in Abbildung 4.8 zu sehen. Die Bewegung der Hand wird dort jeweils wieder mit einer grünen Linie markiert. Die gefundenen Wischgesten erhalten einen orangefarbenen Pfeil um die Richtung anzuzeigen. Dabei ist deutlich zu erkennen, dass die Richtung der Bewegung recht zuverlässig erkannt wird.

Wie in Kapitel 4.3 bereits erwähnt, wird nicht direkt nach Händen im Bild gesucht, sondern nur nach sich bewegenden hautfarbenen Flächen. Um die Bewegung dieser Flächen nachvollziehen zu können, wird der jeweilige Schwerpunkt berechnet. Dieser Punkt stimmt allerdings nicht mit dem tatsächlichen Mittelpunkt der Hand überein. Dies zeigt sich besonders deutlich in Abbildung 4.8b. Die Tatsache ändert allerdings nichts daran, dass die Bewegungsrichtung trotzdem recht genau erkannt wird.

In der Praxis trat zudem häufig das Problem auf, dass eine Wischgeste zwar erkannt wurde, dass aber die Hand direkt danach wieder eine Geste in die entgegengesetzte Richtung auslöste. Um dem zuvor zu kommen, wird nach einer Geste eine kurze Zeit gewartet in der keine neuen Gesten ausgelöst werden können. Diese Zeit wird definiert durch die Dauer, die benötigt wird um alle Punkte der Sequenz durch neue zu ersetzen. Wenn also eine Sequenz aus 30 Punkten besteht und in dieser eine Geste gefunden wird, dann müssen erst 30 neue Punkte gesammelt werden, bevor wieder eine Geste erkannt werden kann. Dadurch ist ausreichend Zeit vorhanden um die Hand nach einer Geste wieder in ihre Ausgangsposition zu bringen.

4.4.3 Kreisgesten

Neben den Wischgesten wurde noch eine weitere Gestenart implementiert. Dabei handelt es sich um Kreisgesten. Wie der Name bereits sagt, führt die Hand hierbei eine Kreisbewegung aus. In Abbildung 4.9 sind zwei Kreisgesten zu erkennen. Dabei wird besonders aus Abbildung 4.9a ersichtlich, dass die Hand keinen perfekten Kreis ausführen muss, um eine Geste auszulösen. Es genügt, wenn die Bewegung kreisähnlich ist. Um das herauszufinden wird überprüft, ob die Bewegung hauptsächlich einer Links- oder einer Rechtskurve folgt.

Wird eine Kreisgeste gefunden, dann werden der Mittelpunkt und der Radius berechnet und beides wird dem Benutzer bereit gestellt. Zur Verdeutlichung wurden in die beiden Abbildungen jeweils ein orangefarbener Kreis mit den berechneten Daten eingezeichnet.

(a) (b)

Abbildung 4.9: Die Hand führt eine Kreisbewegung aus und dies wird als Geste erkannt

4.5 Kommunikation mit Processing

Die bisherige Vorgehensweise basierte hauptsächlich auf OpenCV und damit auf der Programmiersprache C++. Nun lautet der Titel dieser Arbeit aber „Processing-Erweiterung“ und von daher muss nun noch die Verbindung zwischen Processing, bzw. Java, und C++ hergestellt werden.

Der ganze Ablauf beginnt im Processing-Programm, in dem das Bild der Kamera ausgelesen wird. Processing bringt für diesen Zweck bereits eigene Bibliotheken mit, so dass diese Aufgabe in einigen wenigen Zeilen Code zu erledigen ist. Das eingelesene Kamerabild liegt nun als Integer-Array vor, wobei jeder Pixel durch einen Integer repräsentiert wird. Dieses Array wird nun über das Java Native Interface (JNI) an den C++-Code übergeben. Dieser wandelt das Integer-Array in ein Byte-Array um, wobei für den Rot-, den Grün- und den Blau-Anteil eines Pixels jeweils ein Byte reserviert wird (sprich drei Byte pro Pixel). Einziges Hindernis hierbei war, dass OpenCV die Farbanteile nicht in der Reihenfolge RGB speichert, sondern dies in umgekehrter Reihenfolge tut.

Nachdem das Bild nun in die richtige Form gebracht wurde, kann es von den OpenCV-Funktionen verarbeitet werden. Dabei werden nun alle Algorithmen durchgeführt, die in diesem Kapitel bisher beschrieben wurden. Über Differenzbild und Hautfarbenerkennung werden Hände und Kopf im Bild lokalisiert, dann deren Schwerpunkte berechnet und diese in Listen gespeichert. Anhand der Punkte in den Listen wird dann versucht herauszufinden,

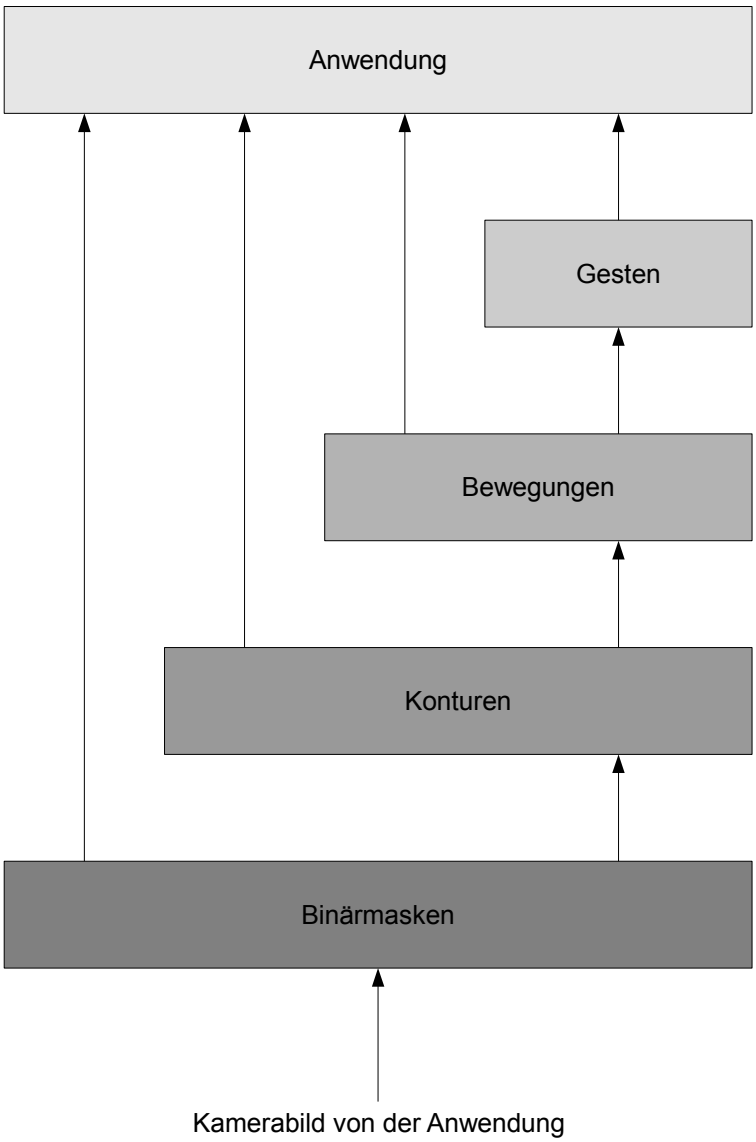


Abbildung 4.10: Das Diagramm zeigt, welche Schritte während der Gestenerkennung durchlaufen werden und welche Daten dem Benutzer zur Verfügung gestellt werden

ob eine Geste stattgefunden hat oder nicht. Dem Benutzer wird zurückgegeben wie viele Gesten erkannt wurden.

Nun hat der Benutzer die Wahl, ob er sich auf die erkannten Gesten verlässt, oder ob er selber versucht aus den Bewegungen etwas zu interpretieren.
Für den Fall, dass er den Deutungen der Bewegung traut, kann er sich einfach mit einem Methodenaufruf die Gesten geben lassen und mit diesen dann in Processing weiter arbeiten. Wenn er selber in den Bewegungen nach Gesten suchen möchte, dann hat er die Möglichkeit, sich die Listen mit den Punkten geben zu lassen. In den Listen kann er dann sogar nach Gesten suchen, die von der Bibliothek noch gar nicht vorgesehen sind und somit hat der Anwender selber die Möglichkeiten, die Bibliothek nach seinen Ansprüchen zu erweitern. In einigen Anwendungsfällen mag es noch hilfreich sein, die aktuellen Koordinaten der Hände und des Kopfes zu bekommen. Auch diese Informationen kann der Benutzer mit einem Methodenaufruf bekommen.

Im Diagramm 4.10 ist dies noch einmal schematisch dargestellt. Von unten nach oben ist dort der Ablauf abgebildet, den das Programm durchläuft. Vom Entgegennehmen des Kamerabildes, über die Binärmasken bis hin zu den erkannten Gesten. Die Pfeile nach ganz oben zur Anwendung verdeutlichen, dass der Benutzer diese Daten bekommen kann, wenn er möchte.

Durch diese Vielzahl an Möglichkeiten wird der Benutzer in seiner Programmierung nicht eingeschränkt und er hat jeder Zeit vollen Zugriff auf alle anfallenden Informationen.

4.6 Hypothesen

Eigentlich handelt es sich bei der Gestenerkennung hauptsächlich um eine Bewegungsinterpretation. Bei dieser Deutung der Bewegungen können aber auch Fehler passieren. Da können Gesten erkannt werden, obwohl der Anwender keine gemacht hat, oder es können die falschen Gesten erkannt werden. Aus diesem Grund ist es denkbar, die Gestenerkennung eher auf Hypothesen zu stützen. Der Benutzer bekäme dann keine klare Antwort zurück, die ihm sagt, dass dort Geste XY war, sondern er würde gesagt bekommen, dass da mit einer Wahrscheinlichkeit von X Prozent eine Geste XY vorlag. Im Zweifelsfalle würde er so die Information bekommen, dass eine Hand zwei verschiedene Gesten mit unterschiedlichen Wahrscheinlichkeiten durchgeführt hat und dann liegt es am Benutzer zu entscheiden, welche er davon ignoriert und mit welcher er weiter arbeitet.

Da so ein System allerdings nicht trivial in der Umsetzung ist, wurde im Rahmen dieser Arbeit darauf verzichtet.

4.7 Fazit

Wie beschrieben sind diverse Verfahren und Algorithmen notwendig um die, für den Menschen so einfache, Aufgabe der Handerkennung zu meistern. Er müssen Differenzbilder erstellt werden und das Kamerabild muss nach Hautfarben durchsucht werden. Dann muss in einer Binärmaske markiert werden, an welchen Stellen Bewegung und Hautfarbe zusammen anzutreffen sind. In dieser Maske muss dann nach zusammenhängenden Bereichen gesucht und von den größeren Bereichen muss der Schwerpunkt berechnet werden. Die Schwerpunkte werden dann in Listen zusammen gefasst, in denen die Bewegungen der Hand über einen gewissen Zeitraum abgebildet werden. Anhand dieser Bewegungen muss nun versucht werden herauszufinden, ob eine Geste stattgefunden hat oder nicht.

Trotz dieses relativ komplizierten Verfahrens zeigen die Ergebnisse, dass es möglich ist, mit einer einfachen Kamera und geeigneter Software einfache Gesten zu erkennen. Bei komplizierteren Gesten tritt diese Art der Herangehensweise allerdings an ihre Grenzen, weil die Erkennung der Hände nicht genau genug ist.

Es hat sich aber auch gezeigt, dass OpenCV bei der Umsetzung des Vorhabens sehr hilfreich war. Zwar wäre es auch durchaus möglich gewesen, alle Funktionen selber zu implementieren, allerdings hätte das ein Vielfaches an Zeit in Anspruch genommen.

5 Umsetzung

5.1 Benutzung der Bibliothek

In Kapitel 3.2.2 wurde bereits erwähnt, dass bei der Bibliothek besonderen Wert auf die einfache Benutzbarkeit gelegt werden muss. Infolgedessen wurde bei der Planung berücksichtigt, dass die Bedienung mit wenigen Methodenaufrufen möglich ist.

```
void setup() {
    size(320, 240);
    video = new Capture(this, width, height);
    Gestenerkennung.setup(video.width, video.height);
}
```

Listing 5.1: Beispiel der Initialisierung

Wie im Codebeispiel 5.1 zu sehen ist, erfolgt die Initialisierung der Gestenerkennung mit einem einzigen Befehl. Dabei muss übergeben werden welche Ausmaße das Kamerabild hat, damit der nötige Speicher reserviert werden kann.

```
void draw() {
    if (!video.available()) return;
    video.read();
    video.loadPixels();

    int anzahlNeuerGesten = Gestenerkennung.sucheGesten(video.pixels);
    if (anzNeuerGesten > 0) {
        Geste[] neueFunde = Gestenerkennung.getGesten();
        for (Geste neueGeste : neueFunde) {
            // auf Gesten reagieren
        }
    }
}
```

Listing 5.2: Beispiel der Gestenerkennung

In Listing 5.2 ist zu erkennen, dass auch im weiteren Verlauf des Programms nur wenige Befehle notwendig sind. Zuerst wird wie gewohnt das Bild von der Kamera abgefragt. Dieses wird dann an die Methode `sucheGesten()` übergeben, welche als Antwort die Anzahl der gefundenen Gesten zurück gibt. Wurden Gesten gefunden, kann mit der Methode `getGesten()` ein Array dieser Gesten angefordert werden. Wie im Listing zu sehen, ist dieses Array vom Typ `Geste`. In einem Objekt dieses Typs ist gespeichert, was für eine Geste es war und an welcher Position sie stattgefunden hat. Bei Kreisgesten ist zusätzlich noch der Radius des Kreises angegeben.

Wie in 4.5 erwähnt können jetzt zusätzlich weitere Daten abgefragt werden. Dazu gehören die Schwerpunkte und die umschließenden Rechtecke der gefundenen Hände, welche mit `getKontur()` ausgelesen werden können. Der Rückgabewert ist ein Array des Typs `Kontur`. In diesem ist gespeichert, wo sich das umschließende Rechteck befindet, wie groß es ist und wo der berechnete Schwerpunkt liegt.

Die verschiedenen Listen mit den Schwerpunkten, die die Bewegungen darstellen, bekommt der Benutzer mit `getBewegung()`. Dabei bekommt er für jede der gefundenen Bewegungen ein Array des Typs `Punkt`. Des Weiteren hat der Benutzer noch Zugriff auf die verschiedenen Binärmasken. Er kann sie mit den Methoden `getMaskeHaut()`, `getMaskeBewegung()` und `getMaskeErgebnis()` abrufen.

5.2 Gestenerkennung am Beispiel eines Bildbetrachters

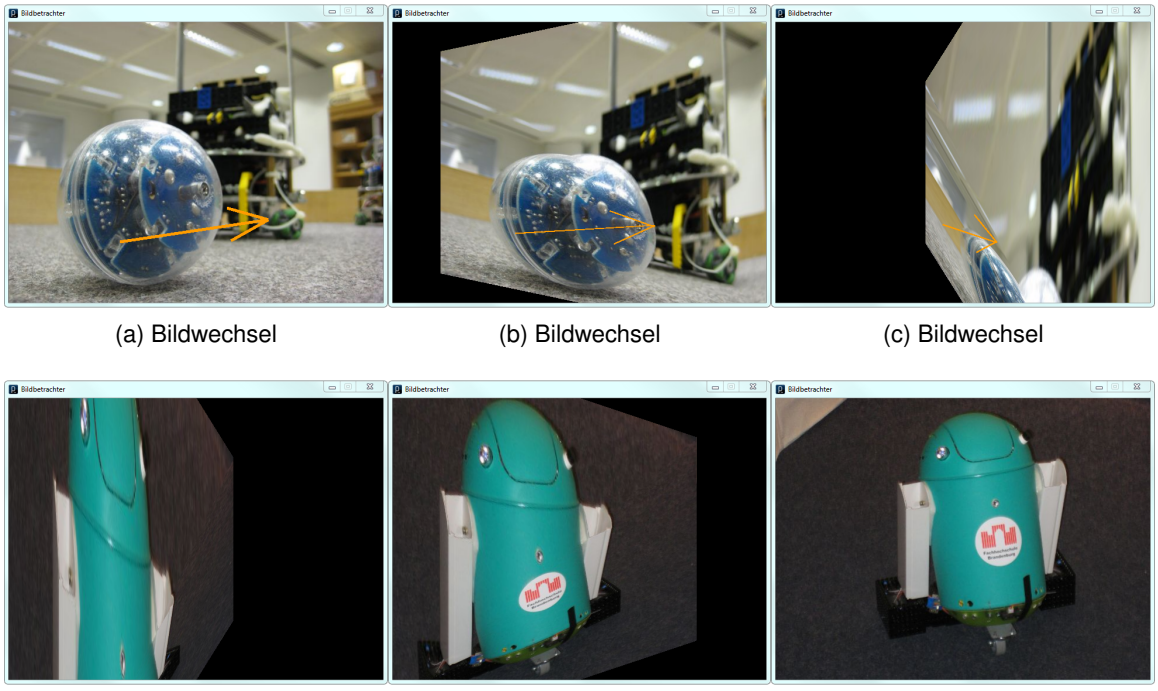
5.2.1 Anwendungsbeschreibung

Eine Bibliothek stellt dem Anwender Funktionen bereit, damit dieser sie nicht selber zu implementieren braucht. Um eine Bibliothek zu testen bedarf es einer Anwendung, die diese Funktionen benutzt. Zur Veranschaulichung der Gestenerkennung entstand deshalb im Rahmen dieser Arbeit ein Programm zum Betrachten von Bildern. Dieses Anwendungsszenario bietet die Möglichkeit die verschiedenen Gesten einzusetzen. Der Hauptzweck der Anwendung ist jedoch die Demonstration der Gestenerkennung, weswegen es sich eher um einen Prototypen als um ein vollwertiges Programm handelt. So kann der Benutzer z. B. ein Menü aufrufen und dort auch Punkte auswählen, aber daraufhin geschieht nichts, da die Funktionalität nicht relevant ist und deshalb auch nicht implementiert wurde.

5.2.2 Gesten

Wischgesten

Wischgesten wurden bereits in Kapitel 4.4.2 beschrieben und in Abbildung 4.8 gezeigt. Dabei handelt es sich um eine gradlinige Bewegung der Hand in eine bestimmte Richtung. Im Beispielprogramm wird diese Art der Geste zum Umblättern der Bilder benutzt. Macht der Benutzer eine Wischgesten nach links, so wechselt er zum vorherigen Bild. Nach rechts bringt ihm zum nächsten Bild. Dabei wird die Y-Richtung der Geste ignoriert. Es ist also egal, ob eine Geste eher nach Rechts-Unten oder nach Rechts-Oben zeigt. Die Hautpsache ist, dass eine klare Bewegung nach Rechts erkennbar ist.



(a) Bildwechsel (b) Bildwechsel (c) Bildwechsel

(d) Bildwechsel (e) Bildwechsel (f) Bildwechsel

Abbildung 5.1: Die erkannte Wischgeste wird als orangefarbener Pfeil eingezeichnet. Die Bildfolge zeigt die Animation, die beim Wechsel der Bilder ausgeführt wird.

Die Abbildungen 5.1a bis 5.1f zeigen den Vorgang des Bildwechsel. Im Bild ist ein orangefarbener Pfeil von links nach rechts eingezeichnet. Dieser symbolisiert die Richtung der erkannten Wischgeste. Nachdem die Geste erkannt wurde, beginnt der Bildwechsel mit einer Animation, bei der die Bilder sich um die Y-Achse drehen.

Beim ersten Bild handelt es sich um einen Fußballroboter, der im Rahmen des Projektmoduls im Informatik-Bachelor an der Hochschule für Angewandte Wissenschaften Hamburg entworfen und gebaut wurde. Nachdem die Animation abgeschlossen ist, zeigt das Programm ein Bild eines Roboters, der von der Fachhochschule Brandenburg entwickelt wurde und der auf Hindernisse, Farben und Geräusche reagiert.

Kreisgeste

Wie in Kapitel 4.4.3 bereits erwähnt, führt die Hand für diese Art der Geste eine Kreisbewegung aus. Im Programm bewirkt der Benutzer dadurch den Aufruf des Menüs. Wie in Abbildung 5.2 zu erkennen, besteht das Menü aus einem Kreis, der in vier Menüpunkte unterteilt ist. In der Mitte befindet sich ein grüner Kreis, der im Laufe der Zeit rot wird. Für diesen Vorgang benötigt er zehn Sekunden und er zeigt damit an, wieviel Zeit noch für die Wahl verbleibt.

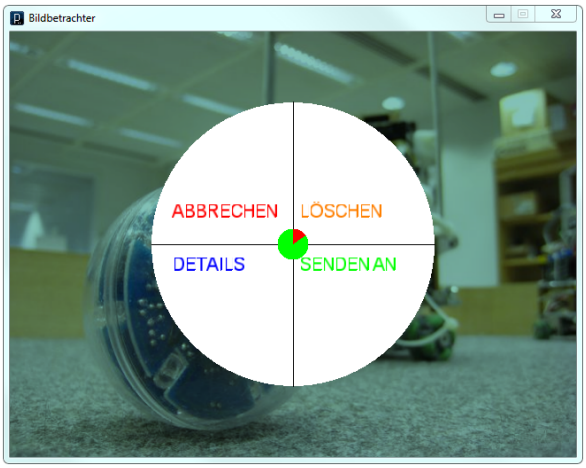
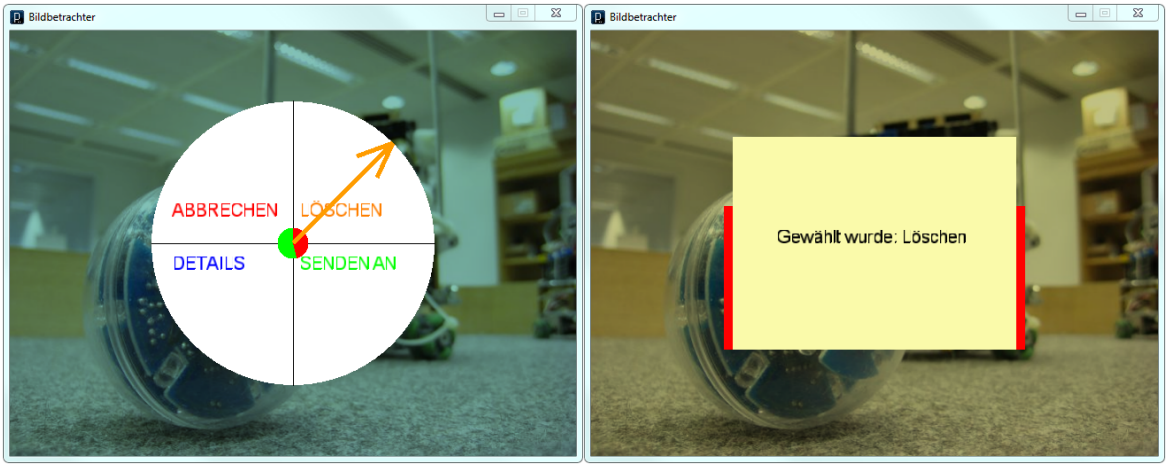


Abbildung 5.2: Das Menü, welches der Benutzer mit einer Kreisgeste aufrufen kann

Zur Auswahl eines Menüpunktes bedient sich der Benutzer einer Wischgeste. Dabei muss die Geste in die Richtung des gewünschten Menüpunktes zeigen. Also wenn der Benutzer den Punkt „Löschen“ auswählen möchte, dann muss er eine Geste von links unten nach rechts oben machen. Nachdem er diese Geste ausgeführt hat, erscheint ein Pfeil, der die Wahl bestätigt (siehe Abb. 5.3a). Die Anzeige für die Zeit beginnt erneut bei Null. Dadurch hat der Benutzer genug Zeit um seine Wahl noch zu überdenken oder sich umzuentscheiden.

Nachdem die Zeit abgelaufen ist, wird dem Benutzer über eine Anzeige (siehe Abb. 5.3b) mitgeteilt, was er gewählt hat. Da es sich bei dem Bildbetrachter nur um eine Demonstration



(a) (b)

Abbildung 5.3: links: der Pfeil markiert die gewünschte Option des Benutzers
rechts: nach Ablauf der Zeit wird angezeigt, welcher Punkt gewählt wurde

der Gestenerkennung handelt, sind die Funktionen jedoch nicht implementiert. Das Bild wird also nicht gelöscht, sondern kann im weiteren Verlauf noch betrachtet werden. Die Anzeige des Ergebnisses verbleibt für fünf Sekunden auf dem Bildschirm und die verbleibende Zeit wird durch zwei rote „Ladebalken“ angezeigt.

5.3 Ist-Analyse die Zweite

Stand der Dinge ist, dass eine Bibliothek entstanden ist, mit der es ermöglicht wird Gesten in einem Live-Video zu erkennen. Die Bibliothek ist in Processing nutzbar und erweitert damit die Funktionalität um die Gestenerkennung.

Allerdings zeigt sich bei der Beispielanwendung auch, dass es gerade im Bereich der Erkennungsrate noch etwas zu tun gibt. Gesten werden zwar erkannt, aber es kommt auch hin und wieder vor, dass eine Geste nicht erkannt wird, obwohl die Bewegung der Hand genau richtig war. Darüber hinaus werden manchmal die unbewussten Bewegungen als Geste interpretiert. Für einfache Anwendungen ist die Erkennungsrate aber durchaus akzeptabel und am Beispiel des Bildbetrachters ist zu erkennen, dass sich damit durchaus benutzbare Programme erstellen lassen.

5.4 Beschränkungen

In Kapitel 3.3 wurden Punkte angesprochen, die eingehalten werden müssen, damit die Gestenerkennung einwandfrei funktioniert. Im Laufe der Entwicklung hat sich gezeigt, dass einige der Einschränkungen berechtigt sind, andere aber durchaus etwas gelockert werden können.

Die Einschränkung, dass möglichst auf gelbe und orangefarbene Kleidung und Hintergründe zu verzichten ist, hat sich als berechtigt herausgestellt. Tests mit entsprechender Kleidung zeigten, dass die Hauterkennung auch die Kleidung erkannte, was zu einem deutlichen Anstieg der Fehlerquote führte.

Ein weiterer berechtiger Punkt ist, dass die Lichtverhältnisse möglichst konstant gehalten werden müssen. Es hat sich mehrfach gezeigt, dass die Hauterkennung unter natürlichem Licht und unter Kunstlicht sehr verschieden reagiert.
Ein Punkt, der dem noch hinzugefügt werden muss, ist die ausreichende Beleuchtung. Bei einer schlechten Ausleuchtung der Umgebung regeln die meisten Kameras (bzw. die Treiber) die Lichtempfindlichkeit hoch. Dies wird erreicht, indem die eintreffenden Signale vom Bildsensor verstärkt werden. Dadurch wird allerdings auch das natürliche Rauschen der Kamera verstärkt. Da die einzelnen Pixel durch dieses Rauschen aber häufig ihren Farb- und Helligkeitswert ändern, kann es dazu kommen, dass Bewegungen erkannt werden, wo eigentlich gar keine sind. Es ist also darauf zu achten, dass die Umgebung immer ausreichend ausgeleuchtet ist.

Einige der Einschränkungen haben sich aber auch als entbehrlich herausgestellt. So ist es nicht notwendig, dass der Benutzer vor einer Geste eine Pause macht, um den Beginn zu kennzeichnen. Da, wie in Kapitel 4.4.2 beschrieben, nur der letzte Teil der Bewegung untersucht wird, spielt es keine Rolle, was vor einer Geste geschehen ist. Wurde erkannt, dass es in den letzten n Bildern eine Wischgeste gab, dann werden die restlichen Bilder der Sequenz nicht mehr untersucht. Von daher ist es irrelevant, ob der Benutzer vorher eine Pause eingelegt hat oder nicht.

Eine weitere Beschränkung, die wieder gelockert werden kann, ist die Begrenzung auf eine Person. Für die Erkennung der Bewegungen nicht nach Personen, sondern nach Händen gesucht. Dabei ist es unerheblich, ob die Hände zu einer oder zu zwei Personen gehören. Anzumerken ist hier aber, dass es bei mehreren Personen im Kamerabild zu Problemen kommen kann, weil es „zu eng“ wird. Wenn die Hände von verschiedenen Personen zu dicht aneinander geraten, dann kann es sein, dass die Software sie als eine einzige Fläche betrachtet. Das kann dazu führen, dass Gesten nicht, oder nicht richtig, erkannt werden.

5.5 Verbesserungspotenzial

Die Gestenerkennung funktioninert zwar prinzipiell, aber es gibt natürlich noch Verbesserungsbedarf. So ist das Angebot der Gesten noch recht begrenzt. Mit nur zwei verschiedenen Gesten lassen sich allenfalls einfache Anwendungen steuern. Wünschenswert wäre es noch weitere Gesten bereitstellen zu können. Denkbar wären zusammengesetzte Gesten, bei denen mehrere Wischgesten z. B. ein Viereck bilden, oder eine Art Winken, bei der sich die Hand mehrfach hin und her bewegt.

Allerdings ist die Erkennung der Hände für Gesten dieser Art vermutlich zu ungenau. Deshalb wäre es denkbar, die Handerkennung auf Verfahren wie SIFT und SURF (siehe Kapitel 4.2.4) umzustellen. Sie versprechen eine höhere Genauigkeit bei der Lokalisierung der Hände im Bild. Dies führt wiederum zu einer genaueren Bewegungserkennung und daraus resultiert eine bessere Gestenerkennung.

5.6 Fazit

Das Hauptziel dieser Arbeit ist das Erstellen einer Erweiterung für Processing um kamerabasiert Gesten erkennen zu können. Dieses Ziel wurde erreicht. Es existiert eine Bibliothek, die einfach in Processing eingebunden werden kann und die es ermöglicht, mit wenigen Befehlen im Video der Kamera nach Gesten zu suchen.

Allerdings gibt es auch Schattenseiten. So ist die Rate der falsch oder gar nicht erkannten Gesten immer noch größer Null. Zwar ist sie nicht so hoch, dass die Erweiterung unbenutzbar wird, aber auf jeden Fall hoch genug, dass sie dem Benutzer auffällt.

Des Weiteren werden bisher nur recht wenige Gesten erkannt. Da der Benutzer aber die Möglichkeit hat auf alle anfallenden Daten der Erkennung zuzugreifen, kann er selber neue Gesten definieren und nach ihnen suchen.

Dessen ungeachtet erfüllt die Erweiterung die selbst gesteckten Ziele, besonders hinsichtlich der Einfachheit. Sie ermöglicht es anderen Programmierern die Gestenerkennung in ihre Processing-Programme in integrieren, ohne dass sie sich selber mit der Bildverarbeitung beschäftigen müssen. Für fortgeschrittene Bedürfnisse bietet sie aber auch den Zugriff auf alle Zwischenschritte der Erkennung, so dass der Benutzer selber die Möglichkeit hat, neue Gesten hinzuzufügen. Auch wenn es noch leichte Defizite bei der Erkennungsrate gibt,

ist die Erweiterung durchaus für einfache Anwendungen nutzbar. Für deutlich kompliziertere Anwendungen bedarf es allerdings einer genaueren Handerkennung, wie sie z. B. durch SIFT oder SUFT erreicht werden kann.

6 Schluss

Die Anfangsproblematik dieser Arbeit war, dass die kamerabasierte Gestenerkennung bisher keine besonders große Verbreitung vorweisen kann. Als Grund wurde genannt, dass die Implementation der Gestenerkennung recht kompliziert ist. Als Lösung sollte dann eine Erweiterung entstehen, die es auch Anwendern mit geringen Programmierkenntnissen ermöglicht, die Gestenerkennung in ihren Programmen zu nutzen. Dieses Ziel wurde auch erfüllt.

Auch wenn es noch ein wenig Verbesserungsbedarf gibt, so ist die Erweiterung doch einsatzfähig. Um die entsprechenden Szenen aus den anfangs genannten Hollywood-Filmen nachzustellen, reicht die Erweiterung hinsichtlich Präzision und Gestenumfang allerdings nicht aus.

Mit der Erweiterung für kamerabasierte Gestenerkennung wird es jetzt möglich von der reinen Bewegungserkennung überzugehen zur Bewegungsinterpretation. Dafür könnte man sich noch einmal das in Kapitel 3.1.2 geschilderte Projekt von Digital Urban Living vorstellen. Bisher wurde einzig und allein auf die Anwesenheit der Besucher reagiert. Mit Gesten wäre es jetzt möglich, dass die Besucher aktiv in das Geschehen auf der Projektionsfläche Einfluss nehmen. Sie könnten mit einfachen Gesten die kleinen Kreaturen herbei winken oder sie irgendwo hin dirigieren. Mit diesem Plus an Interaktionsmöglichkeiten wäre die Installation ungleich interessanter für die Besucher.

Darüber hinaus bleibt es abzuwarten, ob die Erweiterung für Gestenerkennung auch tatsächlich von Künstlern benutzt wird. Die Arbeit der Informatiker ist zunächst einmal abgeschlossen und jetzt ist es an den Künstlern diese Arbeit zu nutzen und fortzuführen.

Literaturverzeichnis

[Bay u. a. 2006] BAY, Herbert ; TUYTELAARS, Tinne ; GOOL, Luc V.: SURF: Speeded Up Robust Features. In: *Proceedings of the 9th European Conference on Computer Vision*, 2006

[Boetzer u. a. 2008] BOETZER, J. ; RAHIMI, M. ; VOGT, M. ; WENDT, P. ; LUCK, K. v.: Gestenbasierte Interaktion mit Hilfe von Multitouch und Motiontracking. (2008)

[Boetzer 2008] BOETZER, Joachim: *Bewegungs- und gestenbasierte Applikationssteuerung auf Basis eines Motion Trackers*, Hochschule für Angewandte Wissenschaften (HAW) Hamburg, Bachelorarbeit, 2008

[Bradski und Kaehler 2008] BRADSKI, Gary ; KAEHLER, Adrian: *Learning OpenCV : [computer vision with the OpenCV library]*. 1. OReilly, 2008. – ISBN 978-0-596-51613-0

[DUL 2010a] DUL: *Center for Digital Urban Living*. http://www.digitalurbanliving.dk/. Letzter Aufruf am 14. September 2010

[DUL 2010b] DUL: *Digital Urban Living - Aarhus by Light*. http://www.aarhusbylight.dk/index-english.html. Letzter Aufruf am 14. September 2010

[Han 2010] HAN, Jeff: *NYU Courant Institute of Mathematical Sciences*. http://cs.nyu.edu/~jhan/. Letzter Aufruf am 14. September 2010

[Hsu u. a. 2002] HSU, Rein-Lien ; ABDEL-MOTTALEB, Mohamed ; JAIN, Anil K.: *Face Detection in Color Images*. 2002

[Istance u. a. 2008] ISTANCE, Howell ; BATES, Richard ; HYRSKYKARI, Aulikki ; VICKERS, Stephen: Snap clutch, a moded approach to solving the Midas touch problem. In: *ETRA '08: Proceedings of the 2008 symposium on Eye tracking research & applications*. New York, NY, USA : ACM, 2008, S. 221–228. – ISBN 978-1-59593-982-1

[Kovac u. a. 2003] KOVAC, Jure ; PEER, Peter ; SOLINA, Franc: Human Skin Colour Clustering for Face Detection. (2003)

[Lowe 2004] LOWE, David G.: *Distinctive Image Features from Scale-Invariant Keypoints*. 2004

[Oliveira und Conci 2009] OLIVEIRA, V. A. ; CONCI, A.: Skin Detection using HSV color space / Universidade Federal Fluminense, Niterói, Brazil. 2009. – Forschungsbericht

[Pavlovic u. a. 1997] PAVLOVIC, Vladimir I. ; SHARMA, Rajeev ; HUANG, Thomas S.: Visual Interpretation of Hand Gestures for Human-Computer Interaction: A Review. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19 (1997), S. 677–695

[Processing 2010] PROCESSING: *FAQ - I know Java, is this Java?* http://wiki.processing.org/w/FAQ. Letzter Aufruf am 14. September 2010

[Rahimi und Vogt 2008] RAHIMI, Mohammadali ; VOGT, Matthias: *Gestenbasierte Computerinteraktion auf Basis von Multitouch-Technologie*, Hochschule für Angewandte Wissenschaften (HAW) Hamburg, Bachelorarbeit, 2008

[Rödiger 2010] RÖDIGER, Marcus: *Interaktive Steuerung von Computersystemen mittels Erkennung von Körpergesten*, Hochschule für Angewandte Wissenschaften (HAW) Hamburg, Masterarbeit, 2010

[Snibbe 2010a] SNIBBE, Scott: *Compliant: bodies distort a soft projected rectangle of light*. http://www.snibbe.com/projects/interactive/compliant. Letzter Aufruf am 14. September 2010

[Snibbe 2010b] SNIBBE, Scott: *Scott Sona Snibbe :: PROJECTS*. http://www.snibbe.com/projects/. Letzter Aufruf am 14. September 2010

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 15. September 2010	
Ort, Datum	Unterschrift