Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Diplomarbeit

Martin Senkbeil

Entwicklung eines Systems zur Programmsteuerung mit Hilfe von Interpretation visueller Gesten

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Martin Senkbeil

Entwicklung eines Systems zur Programmsteuerung mit Hilfe von Interpretation visueller Gesten

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Informatik
Studienrichtung Softwaretechnik
am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Kai von Luck
Zweitgutachter : Prof. Dr. Wolfgang Renz

Abgegeben am 30. August 2005

Martin Senkbeil

Thema der Diplomarbeit

Entwicklung eines Systems zur Programmsteuerung mit Hilfe von Interpretation visueller Gesten

Stichworte

Disappearing Computer, Gestenerkennung, Java, Mensch-Maschine-Kommunikation

Kurzzusammenfassung

Diese Diplomarbeit beschäftigt sich mit der Verbesserung der Kommunikation zwischen Mensch und Maschine, durch den Einsatz intuitiverer Bedienungsformen an Stelle von Tastatur und Maus.

Dazu wird ein System erstellt, welches in der Lage ist, ein farbiges Objekt mit Hilfe einer USB-Kamera zu erfassen. Es verfolgt dessen Bewegungen und wertet sie auf durchgeführte Gesten aus. Beim Erkennen von Gesten führt es hinterlegte Kommandos aus, mit denen andere Anwendungen gesteuert werden können. Zur Steuerung bedient es sich dabei einer Skriptsprache für Microsoft Windows.

Martin Senkbeil

Title of the paper

Development of a system for instrumentation of programs by means of interpretation of visual gestures

Keywords

disappearing computer, gesture recognition, human-computer interaction, Java

Abstract

This thesis (diploma) concerns itself with the improvement of human-computer interaction, by the employment of more intuitive operating forms as a substitute for keyboard and mouse.

To achieve this goal, a system will be developed which is capable of tracking a colored object by using a USB camera. It tracks its movements and evaluates them for accomplished gestures. On the recognition of gestures it executes commands with which other applications can be instrumented. For the instrumentation it makes use of a scripting language for Microsoft Windows.

Danksagung

Vielen Dank meiner Familie und meinen Freunden, ohne die ich heute nicht das wäre, was ich bin.

Danken möchte weiterhin meinem Professor Prof. Dr. Kai von Luck, ohne den diese Arbeit nie zustande gekommen wäre.

Mein Dank gilt auch dem angehenden LaTeX-Guru Björn Jensen, der mich ausgiebig mit LaTeX-Material versorgte und somit die Grundlage für dieses schöne Dokument geschaffen hat.

Abschließend noch ein großes Dankeschön an alle, die mich in sonst einer Form unterstützt haben.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Wir leben heute in einer Zeit, aus der Computer nicht mehr wegzudenken sind. In immer mehr Bereichen umgeben wir uns mit ihnen. Wir haben Navigationssysteme im Auto, Infoterminals an Bahnhöfen und Flughäfen, Überweisungsautomaten in der Bank, ein Multimediacenter in der Wohnung, natürlich den klassischen PC und viele, viele mehr.

Dank der zunehmenden Miniaturisierung tragen wir sogar schon immer öfter welche mit uns herum. Zum Beispiel in Form eines MP3-Players, eines Notebooks oder eines PDAs.

Wir sind uns ihres Nutzens bewußt und genießen es, dass sie uns mit Informationen versorgen und uns das Leben vermeintlich erleichtern. Gleichzeitig geben wir uns jedoch damit zufrieden, dass wir uns ihnen anpassen müssen. So lassen wir uns von ihnen eine künstliche Art der Kommunikation und Interaktion aufdrängen, die nicht nur unergonomisch sondern auch untypisch für den Menschen ist.

Das Benutzen eines durchschnittlichen Computers sieht doch weitestgehend so aus, dass der Benutzer an einem Bildschirm die jeweils möglichen Aktionen abliest, um sie dann durch umständliche Eingaben per Tastatur oder Maus auszuführen. Diese Art der Interaktion führt jedoch oft dazu, dass das Erledigen von Aufgaben weitaus mehr Zeit und Aufmerksamkeit des Benutzers beansprucht, als eigentlich notwendig wäre.[1] Es kommt schnell der Wunsch nach einer Verbesserung auf. Einer Innovation, weg von dem bisherigen Paradigma WIMP[2]-basierter Systeme, hin zu intuitiveren Bedienungsformen.

Ein Wunsch, den [Streitz und Nixon (2005)] zum Beispiel sehr gut beschreiben:

„Computers became primary objects of our attention resulting in an area called „human computer interaction“[3]. Today, however, we must ask: Are we actually interested in interacting with computers? Isn’t our goal rather to interact with information, to communicate and to collaborate with people? Shouldn’t the computer move into the background and disappear?"

[1] Ein Umstand den jeder kennt, der er schon einmal ein umfangreicheres Schriftstück (wie zum Beispiel eine Diplomarbeit) am PC erstellt hat.

[2] WIMP - Kurzform für „Windows, Icons, Menus, Pointing device“. Eine Beschreibung für grafische Benutzeroberflächen, wie sie heutige Betriebssysteme bereitstellen.

[3] Human-Computer-Interaction (HCI) - Damit ist die Art und Weise, in der Mensch und Maschine miteinander interagieren gemeint.

Der Umsetzung dieses Wunsches haben sich mittlerweile viele Forscher und Entwickler verschrieben.

Ihr Bestreben führte dazu, dass in letzter Zeit auch vermehrt von dem Schlagwort „Disappearing Computer“ die Rede ist. Es steht dabei für das Ziel, dass Computer immer mehr in den Hintergrund treten. Sie sollen den Menschen bei seinen Tätigkeiten nur unterstützen statt vorrangig dessen Aufmerksamkeit an sich zu binden. Dadurch werden dann die Benutzer wieder in die Lage versetzt, sich auf das konzentrieren zu können, was sie eigentlich tun wollen.

Maßgeblich wird das Umfeld des „Disappearing Computer“ durch Mark Weisers [Weiser (1991)] Vision des „Ubiquitous Computing“ geprägt. Seine Vision sah es vor, dass wir in der Zukunft von vielen kleinen und spezialisierten Rechnern umgeben sein werden. Diese werden nahtlos in unsere Umgebung integriert sein und über, für uns transparente Interfaces verfügen. Wir werden sie kaum noch bewußt wahrnehmen und so natürlich und intuitiv benutzen, wie andere Alltagsgegenstände auch.

Um jedoch solch ein transparentes Interface zu erhalten, muss der Computer praktisch in den natürlichen Kommunikationsapparat des Menschen integriert werden. Folglich ist es notwendig, dass zukünftig Sprache, Mimik und Gesten als Eingabemedien verwendet werden können, da sie die natürlichsten Kommunikationsformen des Menschen darstellen.

Dies wird nicht zuletzt auch durch die zunehmende Miniaturisierung, die zu immer kleineren und spezialisierteren Geräten führt, erforderlich. Diese Geräte bedürfen ebenso alternativer Formen der Interaktion.

Stellenweise hat sich ja auch schon einiges getan. So gibt es heute durchaus schon einige Alternativen, welche die Bedienung von Computern verbessern:

- Touchscreens, die es erlauben intuitiv mit den Fingern auf Knöpfe und Menüpunkte zu drücken anstatt umständlich einen Zeiger mit der Maus bewegen zu müssen.
- PDAs, welche die Eingabe von Text in handschriftlicher Form ermöglichen und somit das umständliche Bedienen einer Tastatur ersetzen.
- Graphic-Tablets, mit denen man am Computer auf natürliche Weise mit Stift und Lineal zeichnen kann.
- Programme zur Spracheingabe, mit denen zum Beispiel Briefe diktiert werden können.[4]

[4] Leider erfordern diese Programme jedoch einen sehr hohen Aufwand zum Einarbeiten. Sie müssen erst langwierig und sorgfältig mit dem Sprachmuster des Benutzers trainiert werden, damit sie Wörter und ganze Sätze zuverlässig verstehen.

Die heutigen Alternativen bieten zwar schon einen gewissen Grad an Freiheit, jedoch bleibt für eine vollständige Integration in den natürlichen Kommunikationsapparat noch einiges zu tun. Das liegt nicht zuletzt daran, dass die notwendigen theoretischen Grundlagen zur Erkennung und Interpretation von visuellen oder akustischen Informationen noch nicht zuverlässig genug arbeiten, um unter allen Bedingungen akzeptable Ergebnisse erzielen zu können. Die Forschungen zum Verstehen der komplizierten Vorgänge des menschlichen Hörens bzw. Sehens sind noch in vollem Gange.

Für weiterführende Anregungen zur Gestaltung intuitiver Interfaces möchte ich an dieser Stelle noch auf [Nielsen (1993)] hinweisen.

Ebenso möchte ich für Informationen, Beispiele und Entwicklungen zum Thema „Disappearing Computer“, sowie als Ausgangspunkt für weitere Recherchen dazu, auf [Disappearing Computer Initiative; Russell u. a. (2005)] hinweisen.

1.1. Motivation

Wie der vorige Abschnitt gezeigt hat, ist das Bestreben hin zu intuitiveren und bequemeren Eingabemöglichkeiten an sich nicht neu. Es finden schon seit geraumer Zeit Forschungen in diesem Bereich statt.

Dennoch gibt es noch viel zu tun und daher besteht ein Teil meiner Motivation darin, auch einen bescheidenen Beitrag dazu zu leisten. Das Thema dieser Arbeit entstand auch nicht zuletzt aufgrund der Art, wie die Gestenerkennung, meiner Auffassung nach, für gewöhnlich im Bereich des PCs verwendet wird. Ich berufe mich dabei auf meine persönlichen Eindrücke, die ich im Rahmen der Recherche gewonnen habe. Die folgende Einteilung stellt daher keinen Anspruch auf Vollständigkeit und / oder Korrektheit.

Im Bereich des PCs kommt die Gestenerkennung in drei verschiedenen Arten zur Anwendung. Die am weitesten verbreitete Art bedient sich der physikalischen Maus um Gesten zu erfassen. Die nächste wertet Videosignale auf Gesten aus und setzt diese zur Emulation der Maus um. Die dritte arbeitet auch auf Videosignalen, setzt die ausgewerteten Gesten jedoch zur Steuerung von proprietären Anwendungen um. Beispiele zu allen drei Arten sind weiter unten aufgeführt.

Wenn man diese Arten genauer betrachtet, so fällt auf, dass die dritte das meiste Potential besitzt um eine intuitive Steuerung eines PCs zu ermöglichen. Die ersten beiden sind dazu einfach zu mauslastig. Es bietet sich daher an, ein System zu entwerfen, das die dritte Art der Gestenerkennung dahingehend erweitert, dass die Steuerung beliebiger Anwendungen möglich wird.

Für den interessierten Leser sind nachfolgend noch einige Beispiele zu den eben erwähnten Arten, sowie deren Arbeitsweise, aufgeführt.

Gesten mit Hilfe der Maus

Es gibt heute schon freie und kommerzielle Anwendungen, die sich der Gestenerkennung in Form von sogenannten Mausgesten bedienen. Solche Gesten werden dabei für gewöhnlich dadurch ausgeführt, dass der Benutzer eine Taste gedrückt halten muss um die Gestenerkennung zu aktivieren (z. B. die rechte Maustaste). Anschließend führt er die Bewegungen der Geste aus. Abgeschlossen wird sie, indem der Benutzer die anfangs gedrückte Taste wieder loslässt.

Sie lassen sich dabei in zwei unterschiedliche Arten einteilen:

1. Die erste erlaubt nur das Steuern der Anwendung selber, indem der Benutzer eine Geste mit der Maus innerhalb des Anwendungsfensters vollführt. Dazu zeichnet er mit der Maus bestimmte Formen oder drückt deren Tasten in einer bestimmten Reihenfolge um eine Aktion auszulösen. Beispiele für diese Art der Steuerung stellen der Browser Opera [Opera] und das Mozilla-AddOn OptiMoz [OptiMoz] dar. In ihnen dienen die Gesten zum Auslösen von Aktionen, wie dem Neuladen einer Seite oder dem Schreiten durch die Historie.
2. Die zweite Art erlaubt das Steuern von beliebigen Anwendungen auf der Ebene des Betriebssystems. Der Benutzer zeichnet dazu die Geste auf dem Bildschirm und startet auf diese Weise ein Programm oder fährt den Rechner herunter. Ein Vertreter hierfür ist StrokeIt [StrokeIt].

Bei der Art der unterstützen Gesten beschränkt sich der Großteil der Anwendungen in diesem Bereich auf einen fest implementierten Satz, der von Haus aus bereitgestellt wird. Eine Ausnahme davon bildet zum Beispiel StokeIt, da es den Benutzer selber beliebige Gesten definieren lässt, indem er sie einfach auf dem Bildschirm zeichnet.

Emulation der Maus durch visuelle Gesten

In diesem Bereich werden die visuellen Gesten in Aktionen der Maus umgesetzt. Auf diese Weise wird versucht, die Bedienung des gesamten Computers (Betriebsystem und Anwendungen) zu erleichtern, indem der Benutzer zum Beispiel durch Handbewegungen mit dem Desktop interagiert. Nachfolgend sind einige Vertreter dieser Art aufgeführt:

- Der Klassiker hierfür wurde bereits 1980 entwickelt und erlaubte es, mittels Sprache und „Fingerzeig“ Objekte auf dem Desktop zu verschieben. Er hörte auf den Namen „Put-That-There“ und wurde von R. A. Bolt [Bolt (1980)] entwickelt. Seine Bedienung sah so aus, dass der Benutzer mit dem Finger auf ein Objekt deutete und „Put that“ sagte um es auszuwählen. Anschließend deutete er auf einen anderen Bereich des

Desktops und sagte „there“ um es dort wieder abzulegen. Dieses System machte sich sogar schon die Spracherkennung zu nutze.

- Ein Beispiel für die Emulation der Maus durch das Verfolgen der Hand stellt „Mouse WebCam“ [Garcia und Morentin] dar.
- Als eine etwas ausgefalle Idee erweist sich auch die Anwendung Nouse [Gorodnichy und Roth]. Sie setzt die Emulation der Maus um, indem sie die Bewegungen der Nase ihres Benutzers verfolgt. Der Hintergedanke dabei ist, dass auf diese Weise körperlich behinderte Menschen auch einen Computer bedienen können.
- Zuguterletzt sei hier auch noch auf [Smith u. a.; Piekarski und Thomas] hingewiesen. Dort wird die Verfolgung der Hände dazu genutzt, eine Maus innerhalb einer Virtuellen Realität umzusetzen.

Visuelle Steuerung

Ein sehr gutes Beispiel für die Steuerung einer Anwendung wird in [Lenman u. a. (2002)] vorgestellt. Dort wurde versucht, eine menübasierte Steuerung von Heimgeräten, wie dem Fernseher und dem CD-Player zu ermöglichen.

1.2. Zielsetzung

Im Rahmen dieser Diplomarbeit soll ein System erstellt werden, welches die Steuerung von anderen Anwendungen auf einem PC, mit Hilfe von visuellen Gesten, ermöglicht. Es soll dies unter Einsatz von relativ einfachen Mitteln bewerkstelligen und dabei für bereits vorhandene Anwendungen eingesetzt werden können, ohne das selbige dafür extra angepasst oder überarbeitet werden müssen.

Als Beweis für seine Funktionsfähigkeit soll das System am Ende in der Lage sein, die visuelle Steuerung durch eine Präsentation mit Microsoft PowerPoint zu ermöglichen. Der Moderator dieser Präsentation soll dabei in die Lage versetzt werden, die einzelnen Folien durch das Ausführen von Gesten zu wechseln.

1.3. Gliederung

In dem nachfolgenden Kapitel 2 werden Rahmenbedingugen festgelegt, welche zum Einen sicherstellen sollen, dass das zu erstellende System einige grundlegende Anforderungen erfüllt und zum Anderen, um die Komplexität des Projektes zu kontrollieren.

In Kapitel 3 werden benötigte Grundlagen aufgeführt, auf denen diese Arbeit beruht.

Kapitel 4 widmet sich der Analyse und dem Entwurf des Systems.

In Kapitel 5 erfolgt die Realisierung in Form eines Prototypen.

Zum Abschluß wird in Kapitel 6 das Ergebnis der Arbeit noch einmal zusammengefasst und es wird ein Ausblick darauf gegeben, wie sich das entworfene System noch verbessern und erweitern ließe.

1.4. Eingetragene Warenzeichen

2. Rahmenbedingungen

In diesem Kapitel werden Rahmenbedingugen festgelegt, welche zum Einen sicherstellen sollen, dass das zu erstellende System einige grundlegende Anforderungen erfüllt und zum Anderen, um die Komplexität des Projektes zu kontrollieren.

- Das System ist unter Verwendung günstiger Komponenten zu erstellen.
- Das System soll für den Einsatz auf einem handelsüblichen PC ausgelegt werden.
- Als Videoquelle ist eine durchschnittliche und handelsübliche Webcam zu verwenden.
- Das Erkennen von Gesten soll auf das Verfolgen einer ausgezeichneten Farbe in den Videobildern beschränkt werden. Dazu wird angeraten, für die Durchführung der Gesten einen entsprechend farbigen Gegenstand (zum Beispiel einen Handschuh) zu verwenden.
- Es soll für das Verarbeiten von Bildern im RGB-Format, mit einer Farbtiefe von 24 Bit, ausgelegt werden.
- Das System ist vorerst so auszulegen, dass es Bewegungen nur in einem eingeschränkten Sichtbereich wahrnimmt. Es kann daher augenblicklich mit der Gestenerkennung beginnen, sobald der Marker sichtbar ist. Siehe dazu auch Abbildung 2.1 für eine beispielhafte Konfiguration.

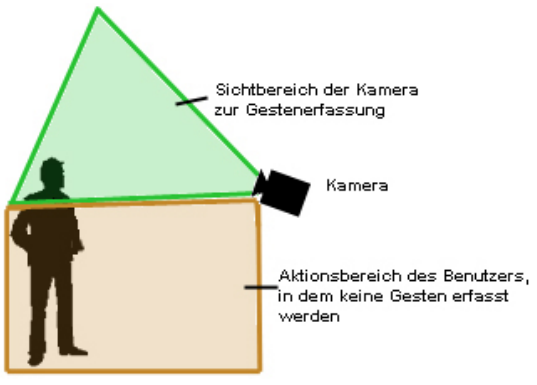


Abbildung 2.1.: Erfassungsbereich der Gesten

- Primär soll das System unter dem Betriebssystem Windows XP von Microsoft arbeiten können. Es ist jedoch wünschenswert, dass es so weit als möglich plattformunabhängig gehalten wird.
- Das System ist so auszulegen, dass bereits bestehende Anwendungen gesteuert werden können. Dabei soll es zunächst einmal primär in der Lage sein, eine Microsoft PowerPoint Präsentation zu bedienen.

Um die Anforderung der Plattformunabhängigkeit zu erfüllen, habe ich mich entschlossen für diese Arbeit Java als Programmiersprache einzusetzen. Es würde sich zwar auch das „.Net“-Framework[1] von Microsoft anbieten, jedoch habe ich mit damit bisher noch keine Erfahrungen gesammelt und somit würde ein zu hoher Aufwand zur Einarbeitung anfallen.

[1] Microsoft hat die Spezifikation der .Net-Laufzeitumgebung offengelegt, wodurch sie auch auf andere Plattformen portiert werden kann. Sie wird daher in naher Zukunft nicht nur auf den bekannten Betriebssystemen aus dem Hause Microsoft zur Verfügung stehen. Das Mono Projekt [Mono] widmet sich zum Beispiel der Portierung nach Linux.

3. Grundlagen

Im Rahmen dieses Kapitels werden Grundlagen aufgeführt, auf denen diese Arbeit beruht.

3.1. Was ist eine Geste?

Da ein Gegenstand dieser Arbeit das Erkennen von Gesten ist, soll erst einmal geklärt werden, was unter diesem Begriff überhaupt zu verstehen ist.

Bei dem Begriff der Geste handelt es sich um ein relativ abstraktes Gebilde. Es beschreibt im Grunde genommen Handlungen, die man dazu verwendet, um seiner Umwelt etwas mitzuteilen. Dabei können die Handlungen verschiedenster Natur sein. So kann eine Geste zum Beispiel darin bestehen, dass man sich vor jemandem verbeugt um ihn zu begrüßen. Eine weitere Möglichkeit besteht darin, dass man die Hand erhebt und hin und her bewegt um sich von jemanden zu verabschieden.

Eine Geste muss jedoch nicht immer von Bewegungen begleitet werden. So kann schon die Haltung der Hand an sich, als Geste aufgefasst werden. Dies wird zum Beispiel von allen Anwendungen verwendet, die die Maus mit Hilfe von Gesten emulieren. Bestimmte Haltungen der Hände bzw. Finger repräsentieren dann das Drücken und Loslassen der Maustasten. Siehe dazu auch Abbildung 3.1.

Abbildung 3.1.: Haltungen der Hand als Gesten, Quelle: [Garcia und Morentin]

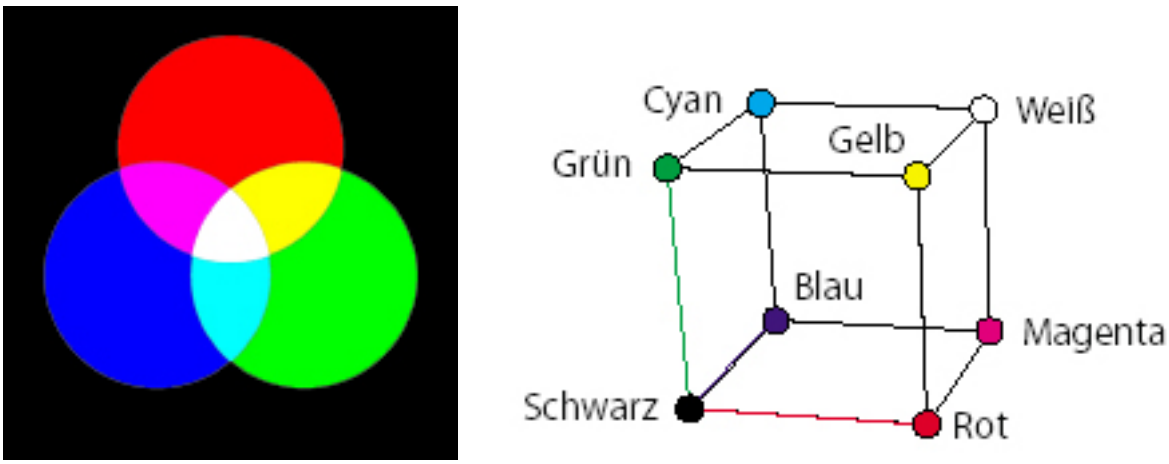
Ein weiteres Merkmal von Gesten ist ihre fehlende Eindeutigkeit. Es verhält sich mit ihnen, wie mit der Sprache. So gibt es Mehrdeutigkeiten in der Form, dass Menschen unterschiedliche Gesten machen, um ein und dieselbe Sache zum Ausdruck zu bringen. Die gewünschte Bedeutung hängt dabei unter anderem stark von dem Umfeld ab, aus dem die jeweilige Person kommt. Es ist daher auch immer individuell vom Einzelnen abhängig, wie und als was eine Geste aufgefasst wird.

Die menschliche Wahrnehmung von Gesten erfolgt für gewöhnlich dadurch, dass man sie mit den Augen sieht. Genauer gesagt, verfolgt man dabei die Haltungen und Bewegungen eines Objekts und wertet sie darauf hin aus, ob es sich bei ihnen um eine Geste handelt oder nicht. Das jeweilige Objekt lokalisiert man dabei anhand seiner Eigenschaften, wie zum Beispiel seiner Farbe. Womit wir zu einer weiteren Grundlage für diese Arbeit kommen - den Farbmodellen.

3.2. Farbmodelle

Die Farbmodelle dienen dazu, Farben anhand von verschiedenen Grundfarben zu beschreiben. In diesem Abschnitt werden die wichtigsten Modelle für diese Arbeit kurz vorgestellt.

3.2.1. Red-Green-Blue (RGB)



(a) Additive Farbmischung

(b) Der RGB Farbraum

Abbildung 3.2.: Das RGB Farbmodell, Quelle: [Gierling (2001)]

Das RGB-Farbraum (Abbildung 3.2) basiert auf der sogenannten additiven Farbmischung. Dabei wird davon ausgegangen, dass man bei der Farbe Schwarz startet und durch das Hinzugeben der drei Farben Rot, Grün und Blau die Farbe Weiß erhält. Daher auch der Name „additive“ Farbmischung. Sie findet immer dann Anwendung, wenn Lichtquellen eine Farbe erzeugen.

So verwendet man sie zum Beispiel bei Monitoren und digitalen Kameras. Bei Monitoren beschießen dabei drei Elektronenstrahlen drei verschiedene Phosphorschichten, welche dann entweder rot, grün oder blau aufleuchten. Bei digitalen Kameras sind lichtempfindliche Sensoren mit einer roten, grünen oder blauen Filterschicht bedampft, so dass nur das von der jeweiligen Filterschicht durchgelassene farbige Licht auf diese Sensoren fallen kann.

Die Werte, mit denen man die Farben Rot, Grün und Blau angibt reichen von 0 (Farbe kommt nicht vor) bis 255 (Farbe ist mit maximaler Intensität enthalten).

Die aus dem RGB-Farbmodell resultierende Beschreibung von Farbe ist für das menschliche Vorstellungsvermögen allerdings zu abstrakt. Beschreibt man zum Beispiel eine Farbe mit den Werten Rot: 135, Grün: 146 und Blau: 108, so kann nicht sofort angegeben werden, um welche Farbe es sich handelt (in diesem Fall ist es ein Olivgrün).

Man versucht deshalb, Farben intuitiver zu beschreiben, und bedient sich dazu des HSB-Modells. Es ist unter Punkt 3.2.3 aufgeführt und kann mit Hilfe der folgenden Formel aus dem RGB-Format abgeleitet werden:

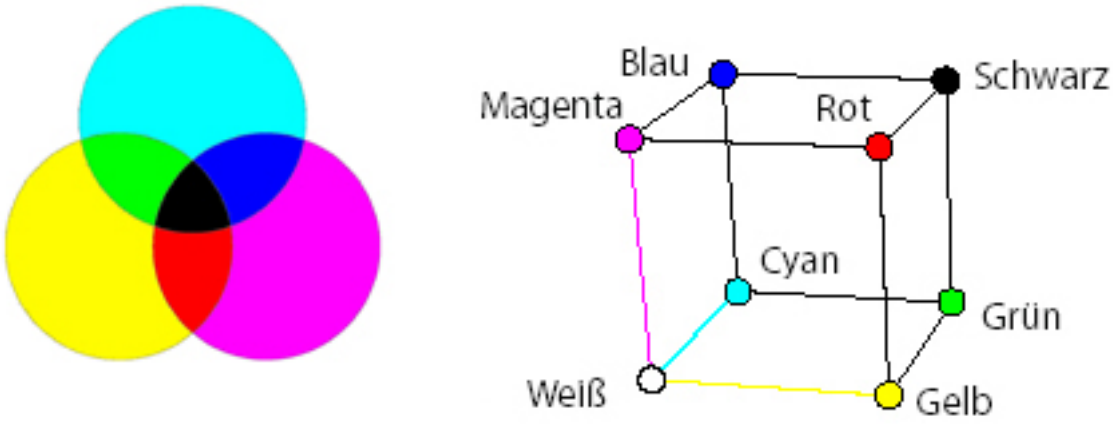
f(rgb) = hsv

$$
\begin{aligned}
&Max = max(R, G, B)\\
&Min = min(R, G, B)\\
&H = \begin{cases} (0 + \frac{G-B}{Max-Min}) \times 60, if(R = Max)\\ (2 + \frac{B-R}{Max-Min}) \times 60, if(G = Max)\\ (4 + \frac{R-G}{Max-Min}) \times 60, if(B = Max) \end{cases}\\
&if(H < 0) H = H + 360\\
&S = \frac{Max-Min}{Max}\\
&V = Max
\end{aligned}
$$

Zunächst soll jedoch der Vollständigkeit halber noch das CMY-Modell erläutert werden. Die Überlegung auf der es beruht stellt nämlich das genaue Gegenstück zu diesem Modell dar. Es lässt sich anhand der folgenden Formel sehr leicht aus dem RGB-Modell ableiten:

$$
\begin{aligned}
C &= 255 - R\\
M &= 255 - G\\
Y &= 255 - B
\end{aligned}
$$

3.2.2. Cyan-Magenta-Yellow (CMY)



(a) Subtraktive Farbmischung (b) Der CMY Farbraum

Abbildung 3.3.: Das CMY Farbmodell, Quelle: [Gierling (2001)]

Beim subtraktiven Farbmodell CMY (Abbildung 3.3) geht man von weißem Licht und einem weißen Blatt Papier aus. Trifft das weiße Licht auf das Papier, so wird es unverändert reflektiert. Trägt man jedoch eine Farbe auf dieses Blatt Papier auf, so lässt sie nur einen farbigen Teil des Lichtes passieren, filtert jedoch einen anderen farbigen Teil heraus. Verwendet werden hierbei die Grundfarben Cyan, Magenta und Yellow (Gelb).

Kombiniert man zwei Grundfarben miteinander, so lassen sich die Mischfarben Rot, Grün und Blau erzeugen. Sind alle drei Grundfarben vorhanden, so werden beim Betrachten alle drei Teilbereiche des weißen Lichts herausgefiltert und es ergibt sich Schwarz. Da man aus dem weißen Licht farbiges Licht entfernt, wird dieses Farbmodell subtraktives Farbmodell genannt.

Dieses Farbsystem bildet die Grundlage für den Farbdruck. Es lässt sich mit Hilfe der nachfolgenden Formel in das RGB-Modell übertragen:

$$R = 255 - C$$
$$G = 255 - M$$
$$B = 255 - Y$$

3.2.3. Hue-Saturation-Brightness (HSB)

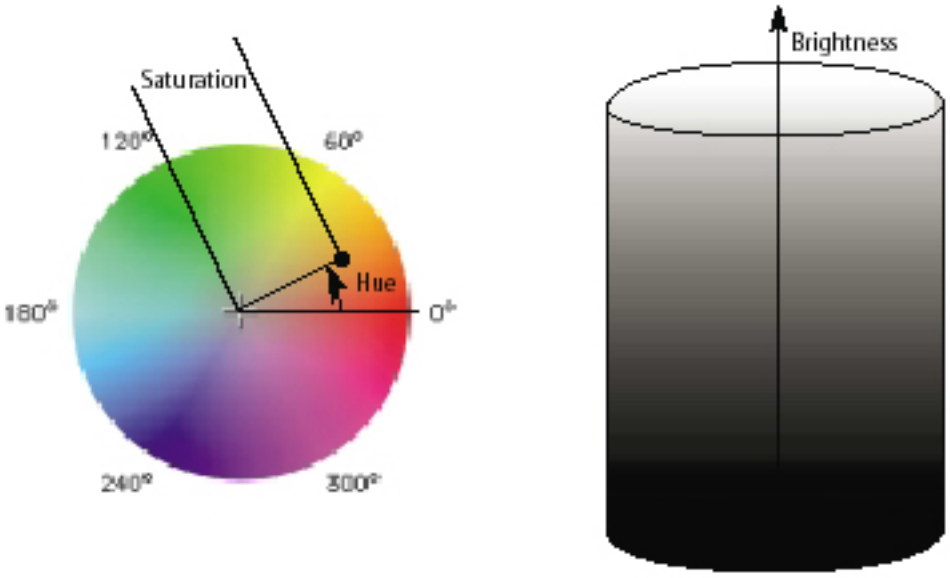


Abbildung 3.4.: Der HSB Farbraum, Quelle: [Gierling (2001)]

Das HSB-Farbmodell leitet sich aus dem RGB-Farbmodell ab und beschreibt Farben für den Menschen intuitiver. Es wird auch oft als HSV-Modell (Hue-Saturation-Value) bezeichnet.

Betrachtet man die Tonne in Abbildung 3.4 von der Seite, so entspricht eine Bewegung von unten nach oben einer Erhöhung der Helligkeit. Im Englischen nennt man die Helligkeit Brightness (B). Ganz unten liegt Schwarz, ganz oben liegt Weiß. Der Wertebereich der Helligkeit geht von 0 (Schwarz) bis 100 (Weiß).

Schneidet man die Tonne bei einem beliebigen Helligkeitswert auf und betrachtet den Ausschnitt von oben, so ergibt sich ein Kreis, in dem die Farbe Rot als Bezugsfarbe festgelegt wurde. Beschreibt man jetzt eine Farbe mit einem Winkel zur Bezugsfarbe Rot, so nennt man diesen Winkel Farbwinkel oder Hue (H). Er gibt die Art einer Farbe an.

Immer wenn Begriffe wie Rot, Orange oder Grün verwenden werden, beschreiben man damit die Art der Farbe. Der Hue-Winkel erfasst einen Wertebereich von 0 - 359 Grad.

Die Sättigung oder Saturation (S) gibt den Abstand einer Farbe zum Kreismittelpunkt an, wo die unbunten Farben angeordnet sind. Das bedeutet, dass die Farben von innen nach außen immer bunter werden. Einem Wert von 0 entspricht eine graue Farbe, einem Wert von 100 entspricht eine maximal gesättigte Farbe. Immer wenn bei der Beschreibung einer Farbe Wörter wie blass oder neon verwenden werden, beschreibt man ihre Sättigung.

Daher ist eine Farbe auch nur dann vollständig beschrieben, wenn man zum Beispiel von einem hellen Blassrosa spricht.

Mit Hilfe der nachfolgenden Formel kann das HSB-Format in das RGB-Format umgewandelt werden:

f(hsv) = rgb

$$
\begin{aligned}
&if(S = 0)R = V; G = V; B = V\\
&Hi = \lfloor \tfrac{H}{60} \rfloor\\
&f = \tfrac{H}{60} - Hi\\
&p = V \times (1 - S)\\
&q = V \times (1 - (S \times f))\\
&t = V \times (1 - (S \times (1 - f)))\\
&if(Hi = 0)R = v; G = t; B = p\\
&if(Hi = 1)R = q; G = v; B = p\\
&if(Hi = 2)R = p; G = v; B = t\\
&if(Hi = 3)R = p; G = q; B = v\\
&if(Hi = 4)R = p; G = t; B = v\\
&if(Hi = 5)R = v; G = p; B = q
\end{aligned}
$$

3.3. Beleuchtung

Die Beleuchtung der Umgebung wirkt sich durch Schatten und Reflexionen auf die grundlegende Fähigkeit zur Erkennung von Farben aus. Starke Reflexionen von Licht auf spiegelnden Oberflächen sorgen zum Beispiel dafür, dass die Farben der näheren Umgebung durch den auftretenden Blendeffekt stark verfälscht und sogar komplett überdeckt werden. Im Gegenzug sorgen Schatten dafür, dass die Farbtöne immer dunkler und dunkler erscheinen, bis sie schließlich in Abstufungen von Grau und Schwarz übergehen. Somit gehen gerade im Bereich der Farbtöne mitunter sehr viele Informationen verloren. Abbildung 3.5 stellt eine solche Situation dar.

(a) Normal ausgeleuchtet (b) Punktuelle Beleuchtung

Abbildung 3.5.: Auswirkung von Beleuchtung

3.4. Instrumentierung von Anwendungen

Unter dem Instrumentieren von Anwendungen versteht man im Allgemeinen das Fernsteuern bzw. Bedienen einer Anwendung durch andere Programme. Es kann in verschiedenen Formen stattfinden.

Eine Anwendung kann zum Beispiel ein spezielles API bereitstellen, über das sie gesteuert werden kann. Ein Beispiel dafür stellen die Office-Programme von Microsoft dar. Jedes davon exportiert Methoden zu seiner Fernsteuerung.

Eine weitere (und allgemeinere) Möglichkeit zur Steuerung einer Anwendung stellt auch das Bedienen ihrer grafischen Oberfläche dar. Um dies zu erreichen bedient man sich im allgemeinen der Methoden, die das API des Betriebssystems bzw. das von seiner Oberfläche zur Verfügung stellt. Als Beispiel wird nachfolgend wieder einmal ein Produkt aus dem Hause Microsoft erwähnt.

Die grafische Oberfläche von Windows basiert auf dem Prinzip, dass alle Ereignisse, welche Anwendungen oder deren grafischen Oberfläche betreffen, in Form von Nachrichten übertragen werden. Geht man nun den klassischen Weg der Programmierung unter Windows mit Hilfe des Win32API, so ist man in der Lage, Ereignisse zu simulieren. Dazu erstellt man einfach die Nachricht, die normalerweise bei dem gewünschten Ereignis versendet wird, und verschickt sie anschließend manuell an eine andere Anwendung.

Da die Programmierung des Betriebssystem-API allerdings oft in C/C++ erfolgt und dadurch etwas umständlich wird, sind für viele Plattformen auch schon Skriptsprachen entwickelt worden. Diese abstrahieren die umständlichen Anweisungen und bieten sie normalerweise mit einer vereinfachten Syntax an. [AutoIt] stellt beispielsweise Methoden zur Steuerung von Windows in einer Sprache bereit, die an Visual Basic angelehnt ist.

4. Analyse und Design

Im Rahmen dieses Kapitels werden zum Einen die Anforderungen betrachtet, die sich für das System ergeben, und zum Anderen wird das Design festgelegt.

4.1. Anforderungen

4.1.1. Hardware

Für eine akzeptable Erkennung von Gesten muss das System in der Lage sein, mit bis zu 30 Bildern pro Sekunde, bei einer Auflösung von mindestens 320x240 Pixeln und 24 Bit Farbtiefe zu arbeiten. Daraus resultiert ein aufkommendes Datenvolumen von:

$$320 * 240 * 3 * 30 = 6912000 Byte/s \approx 6,59 MB/s.$$

In Anbetracht der maximalen Übertragungsrate der üblichen Schnittstellen (siehe Abbildung 4.1) ist es daher erforderlich, dass die eingesetzte Kamera entweder IEEE1394 („FireWire“) oder USB 2.0 unterstützt. Selbiges gilt natürlich auch für den Rechner, auf dem das System eingesetzt wird.

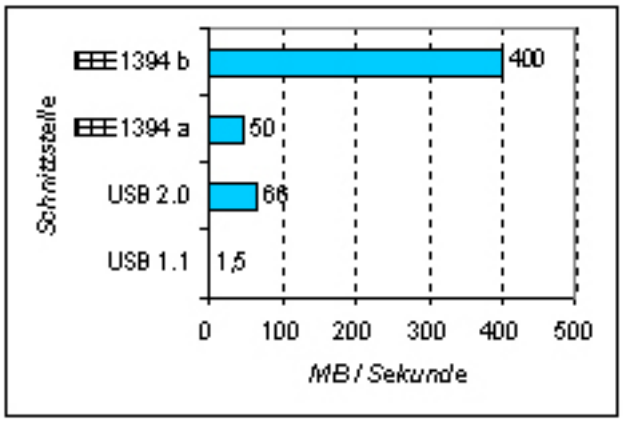


Abbildung 4.1.: Übertragungsraten von USB und IEEE1394 („FireWire“)

Durch die Nutzung dieser Schnittstellen würde auch die Verwendung einer Auflösung von 640*480 Pixeln möglich werden. Bei ihr würde eine Datenvolumen von 23,37 MB/s anfallen.

4.1.2. Software

Da das System zur Steuerung von anderen Anwendungen eingesetzt werden soll, darf es den Rechner nicht zu stark auslasten. Die benötigte Rechenleistung ist daher zu minimieren.

4.2. Systembetrachtung

Grundsätzlich lässt sich die erforderliche Arbeitsweise des Systems folgendermaßen beschreiben:

1. Das System ruft kontinuierlich Bilder von der Kamera ab und wertet sie auf das Vorhandensein eines Objektes aus. Es tut dies anhand einer ausgezeichneten Farbe.
2. Sofern das Objekt erkannt wurde, muss es dessen Bewegung verfolgen und sie auf mögliche Gesten hin auswerten.
3. Bei Erkennen einer Geste prüft es, ob ihr ein Kommando zugeordnet ist. Wenn ja, dann führt es das Kommando aus und instrumentiert dadurch eine externe Anwendung.
4. Nach erfolgter Ausführung des Kommandos geht es wieder bei Punkt 1 weiter.

Bei genauerer Betrachtung dieser Arbeitsweise wird deutlich, dass sich das System in unterschiedliche Aufgabenbereiche einteilen lässt. Dabei handelt es sich um die Bildverarbeitung, die Gestenerkennung, die Kommandos, die Zuordnung von Gesten und Kommandos, die Instrumentierung und etwas, das diese miteinander verbindet.

Darüber hinaus lassen sich diese Bereiche klar voneinander trennen. Sie können daher als eigenständige Komponenten aufgefasst werden. Bei einer Erweiterung dieser Komponenten um eine Benutzeroberfläche ergibt sich dann die nachfolgende Architektur des Systems:

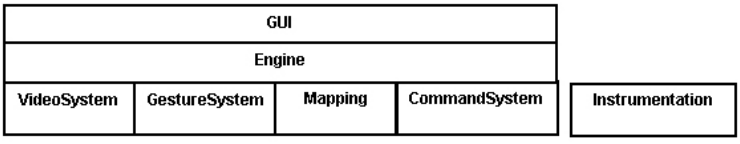


Abbildung 4.2.: Architektur des Systems

- „GUI“ - Die Benutzeroberfläche. Über sie erfolgt die Bedienung und Konfiguration durch den Benutzer.
- „Engine“ - Sie enthält die generelle Programmlogik, die die einzelnen Komponenten miteinander verbindet. Ferner stellt sie eine Fassade dar und verbirgt die interne Arbeitsweise vor der Benutzeroberfläche.
- „VideoSystem“ - Es ist für das Erfassen und Auswerten der Bilder zuständig.
- „GestureSystem“ - Es umfasst die Erkennung und Verwaltung der Gesten.
- „Mapping“ - Die Zuordnung von Gesten und Kommandos.
- „CommandSystem“ - Es umfasst die Verwaltung und Ausführung der Kommandos.
- „Instrumentation“ - Eine externe Instrumentierungskomponente, die das eigentliche Ansteuern anderer Anwendungen durchführt. Der Grund dafür, warum sie extern ist wird unter Punkt 4.7 genauer erläutert.

Abbildung 4.3 stellt die grundlegende Kommunikation zwischen den Komponenten dar, welche zur Erfüllung der Aufgabe des Systems erforderlich ist.

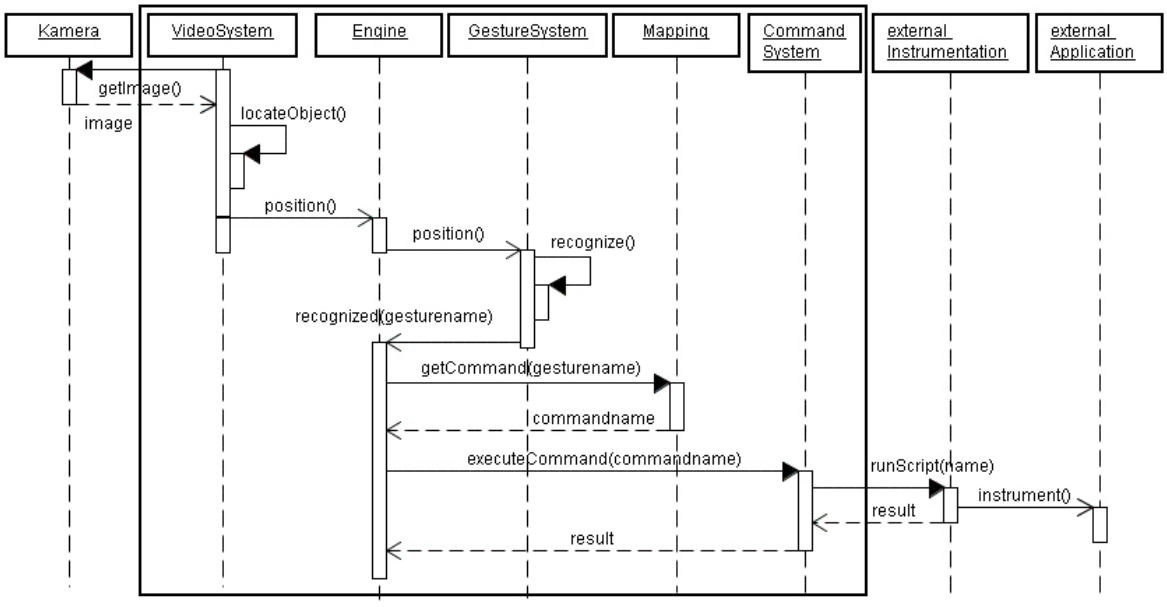


Abbildung 4.3.: Grundlegender Ablauf von Erfassung bis zur Ausführung

4.3. Bildverarbeitung

Die Bildverarbeitung hat zur Aufgabe, dass zu verfolgende Objekt zu erfassen und dessen Position zu ermitteln. Es verwendet dazu den nachfolgend dargestellten Algorithmus. Im Anschluss daran ist unter Punkt 4.3.2 der prinzipielle Aufbau des Videosystems zu sehen.

4.3.1. Algorithmus zur Objektlokation

Das System soll Gesten erkennen, indem es die Bewegung eines einfarbigen Objektes verfolgt. Um dies zu ermöglichen, müssen die Bilder der Kamera daraufhin ausgewertet werden, ob das Objekt vorhanden ist, und wenn ja, an welcher Position es sich in dem jeweiligen Bild befindet.

Das Ermitteln der Position findet dabei unter den folgenden Annahmen statt:

1. Im Kontext eines Bildes stellt das gesuchte Objekt eine zweidimensionale Fläche dar, welche durch eine einzige Farbe beschrieben wird. Dabei wird davon ausgegangen, dass diese Farbe ausschließlich von dem Objekt verwendet wird und ansonsten nicht vorkommt.
2. Die Position des Objektes wird durch den Schwerpunkt der Fläche angegeben.

Zusätzlich ist noch zu beachten, dass selbst einfarbige Objekte, auf Grund ihrer Beschaffenheit und der aktuellen Beleuchtung, nicht über nur einen einzigen Farbton verfügen. Sie besitzen vielmehr mehrere Nuancen ihrer Farbe. Daher wird, unter Verwendung einer anzugebenden Toleranz, ein Farbraum für die Erkennung aufgespannt. Die Angabe der Toleranz erfolgt dabei im HSB-Farbmodell, da es eine intuitivere Definition von Farben ermöglicht und es erlaubt, ein Farbbereich unabhängig von Sättigung und Helligkeit zu definieren. Das RGB-Farbmodell ist dafür zu unflexibel.

Grundlegender Algorithmus zum Ermitteln des Schwerpunktes

Der Schwerpunkt der gesuchten farbigen Fläche lässt sich relativ leicht in zwei Schritten ermitteln.

Im ersten Schritt gilt es, die Koordinaten aller Pixel mit der richtigen Farbe zu ermitteln. Dazu legt man zuerst eine Toleranz für das Erkennen der gesuchten Farbe fest. Anschließend iteriert man über alle Pixel hinweg, wandelt sie jeweils in das HSB-Format um[1], und prüft dann, ob deren Farbe in den Bereich der gesuchten Farbe plus/minus der festgelegten Toleranz fällt. Wenn ja, dann merkt man sich die Koordinaten des jeweiligen Pixels.

[1] Diese Umwandlung ist notwendig, da die Bilder der Kamera im RGB-Format vorliegen.

Im zweiten Schritt wird aus den ermittelten Koordinaten der Schwerpunkt gebildet. Dazu kann man zum Beispiel deren Mittelwert bilden, indem man die Werte jeweils einer Achse aufaddiert und durch die Anzahl der Punkte teilt.

Die Verwendung des Mittelwertes hat jedoch den Nachteil, dass er sehr Anfällig gegenüber Schwankungen im Bereich der Koordinaten ist. Das äußert sich darin, dass selbst wenige Pixel das Ergebnis stark beeinflussen, wenn ihre Koordinaten über hohe Werte verfügen. Daher empfiehlt sich eher der Einsatz des Medians. Dieser reagiert nicht so empfindlich auf Schwankungen.

Der oben beschriebene Algorithmus reicht im Prinzip aus um den Schwerpunkt der gesuchten Fläche zu ermitteln. Es lassen sich jedoch noch einige Optimierungen vornehmen, die es nicht mehr erforderlich machen, dass jedes Pixel betrachtet werden muss. Sie werden im nachfolgenden Punkt beschrieben.

Optimierung des Algorithmus

Bei dem grundlegenden Algorithmus ist es erforderlich, dass jedes Pixel betrachtet wird. Das führt bei der geforderten Mindestauflösung von 320x240 zu der Betrachtung von 76800 Pixeln. Da aber davon ausgegangen werden kann, dass die gesuchte Fläche größer als nur ein Pixel ist, besteht nicht die Notwendigkeit, dass alle betrachtet werden.

In diesem Design wird daher die Nachfolgende Optimierung verwendet:

Um die Anzahl der betrachteten Pixel zu verringern findet eine virtuelle Skalierung des Bildes statt. Dazu wird es zunächst in ein Raster von gleich großen Blöcken unterteilt. Die Blöcke haben dabei eine konfigurierbare Größe, welche standardmäßig 8x8 Pixel beträgt. Sie wurde gewählt, weil bei ihrer Verwendung alle gängigen Bildgrößen von Webcams in ein komplettes Raster unterteilt werden können, ohne dass ein Rand übrig bleibt.

Für den weiteren Verlauf wird das Raster als eigenständiges Bild betrachtet, bei dem die einzelnen Blöcke dessen Pixel repräsentieren. Es wird als Grundlage für das Ermitteln des Schwerpunktes verwendet. Dies bedeutet, dass von nun an der Schwerpunkt anhand der Koordinaten der Blöcke gebildet wird. Welche Blöcke dazu zu verwenden sind, wird anhand der nachfolgenden Vorgehensweise entschieden.

Jeder Block des Rasters ist dann zu verwenden, wenn er eine bestimmte Anzahl von Pixeln mit der gesuchten Farbe enthält. Die Anzahl der benötigten Pixel ist dabei konfigurierbar gehalten, jedoch werden maximal neun Stück betrachtet. Siehe dazu auch Abbildung 4.4. Dort sind sie beispielhaft dargestellt. Tests haben ergeben, dass es normalerweise reicht wenn mindestens vier der betrachteten Pixel die richtige Farbe haben um einen Block als gültig gelten zu lassen.

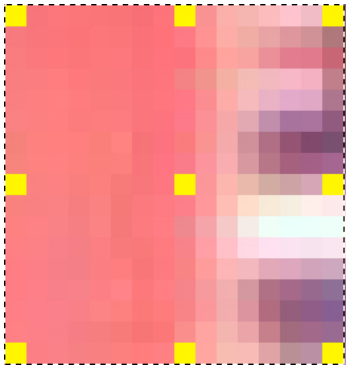


Abbildung 4.4.: Betrachtete Pixel in einem Block

Dank dieser Optimierungen werden nun bei einer Auflösung von 320x240, im schlimmsten Fall (die Farbe wird nicht gefunden) nur noch 10800[2] Pixel betrachtet und im günstigsten sogar nur 4800. Darüber hinaus werden selbst bei einer Auflösung von 640x480 maximal 43200 Pixel betrachtet.

Insgesamt wurde die Anzahl der zu betrachtenden Pixel um den Faktor sieben reduziert.

4.3.2. Videosystem

Das Videosystem ist in Form eines Threads auszulegen, da es eigenständig Bilder von der Kamera abrufen muss. Es besteht hauptsächlich aus einem sogenannten „FrameGrabber“ und Filtern.

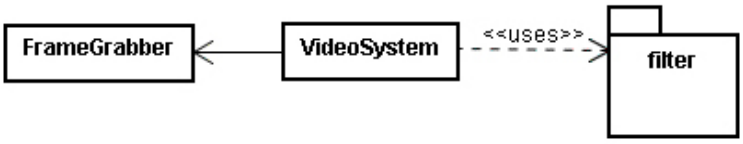


Abbildung 4.5.: Videosystem

Bei dem FrameGrabber handelt es sich um eine Klasse, die den Zugriff auf die Kamera kapselt. Die Filter dienen zum Bearbeiten und Auswerten der erfassten Bilder. Mit ihrer Hilfe wird der Algorithmus zur Objektlokation aus Punkt 4.3.1 umgesetzt.

[2] 320x240 Pixel => 40x30 Blöcke => 1200 Blöcke x 9 Pixel = 10800 Pixel

Die Arbeitsweise des Videosystems sieht so aus, dass es zuerst ein Bild von dem Frame-Grabber anfordert. Anschließend leitet es das Bild durch die Filter und erhält dabei die Position des verfolgten Objektes von dem Lokationsfilter. Diese gibt es dann an die Engine weiter (siehe Abbildung 4.3).

Nachfolgend wird aufgeführt, welche Filter verwendet werden und was deren jeweilige Aufgabe ist. Unter Punkt 4.3.2.1 wird zudem noch eine Erweiterung eingeführt, welche die Performanz des Videosystems erhöht.

Filter

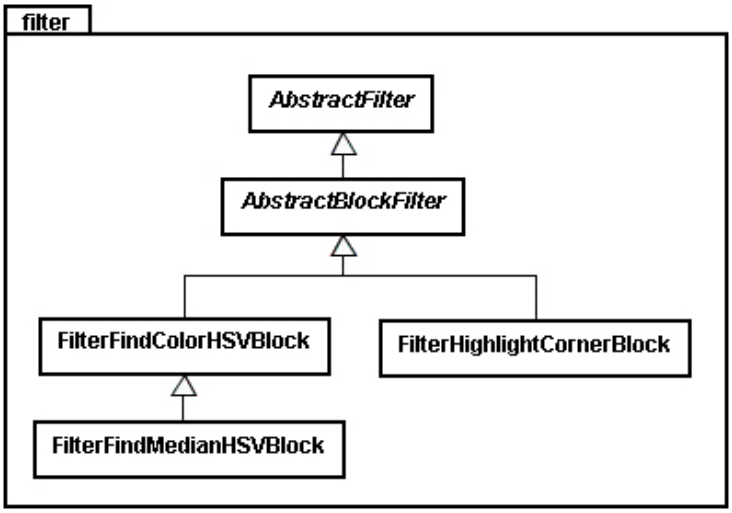


Abbildung 4.6.: Filter

- AbstractFilter - Eine abstrakte Klasse, die grundsätzliche Eigenschaften von Filtern definiert.
- AbstractBlockFilter - Eine abstrakte Klasse, die grundsätzliche Eigenschaften von Filtern definiert, welche auf der Basis von „Blöcken“ (siehe 4.3.1) arbeiten.
- FilterFindColorHSVBlock - Ein Filter der Blöcke mit einer bestimmten Farbe findet und sie farblich hervorhebt.
- FilterFindMedianHSVBlock - Der eigentliche Filter zu Ermittlung des Schwerpunktes einer definierbaren Farbe. Er stellt den Lokationsfilter dar.
- FilterHighlighCornerBlock - Ein Filter zur Visualisierung der Blöcke, in die ein Bild eingeteilt wird.

4.3.2.1. FilterThreads

Durch das hintereinander geschaltete Ausführen der Filter geht sehr viel Zeit verloren, da ein Bild immer erst alle durchlaufen haben muss, bevor ein neues bearbeitet werden kann. Hinzu kommt, dass Wartezeiten beim Abrufen eines neuen Bildes von der Kamera entstehen. Diese Wartezeiten gilt es zu beseitigen.

Daher werden spezielle Threads für die Filter eingeführt werden - die „FilterThreads“.

Ein solcher Thread kapselt jeweils einen Filter und ihm kann einen weiterer „FilterThread“ als Nachfolger zugewiesen werden, wodurch dann eine Pipeline entsteht.

Sie ermöglichen ein gleichzeitiges Bearbeiten der Bilder durch die Filter, während bereits neue von der Kamera abgerufen werden. Auf diese Art kann der erste Filter der Pipeline wieder eine neues Bild erhalten, während sich ein anderes noch an einer anderen Stelle der Pipeline aufhält.

Um den Prozessor nicht vollständig auszulasten, blockieren die Threads wenn kein Bild verfügbar ist. Schließlich sollen auch noch andere Programme laufen können.

Ihre Arbeitsweise ist der nachfolgenden Abbildung (4.7) zu entnehmen.

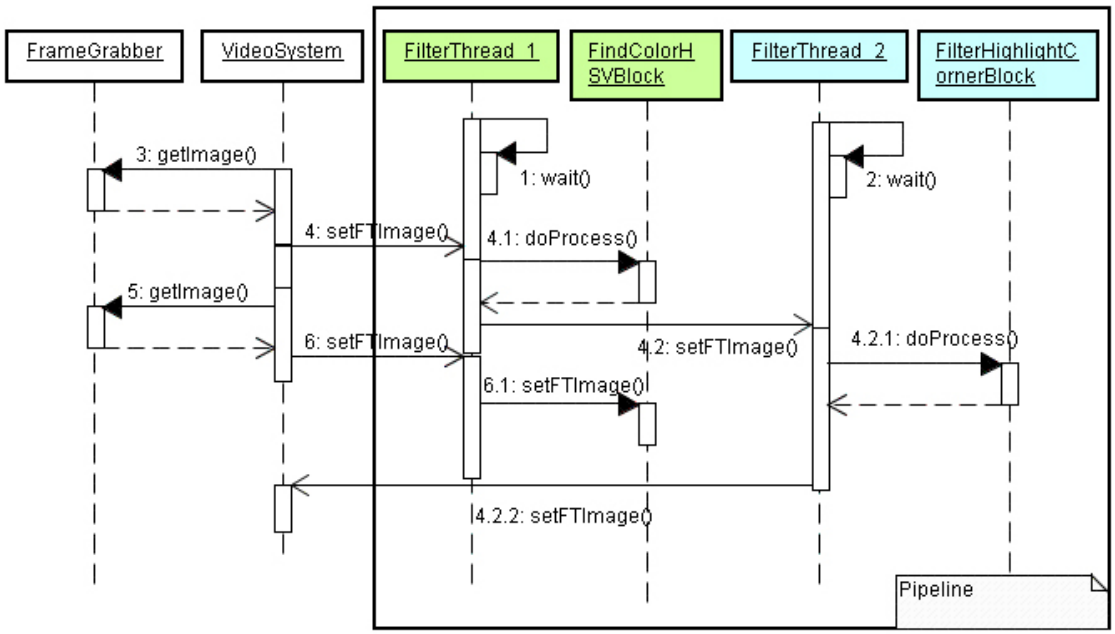


Abbildung 4.7.: FilterThreads

4.4. Gesten und ihre Erkennung

4.4.1. Analyse

- Um einen eventuellen Lernaufwand so gering wie möglich zu halten, müssen die Gesten im Kontext der zu instrumentierenden Anwendungen einen Sinn für den Benutzer ergeben. Hinzu kommt auch, dass Präferenzen und körperliche Möglichkeiten der Benutzer zu berücksichtigen sind. Der eine möchte vielleicht seinen Webbrowser mit einer geraden Bewegung von links nach rechts starten, der andere mit einer Bewegung von oben nach unten.

  Es erscheint daher am sinnvollsten, die Gesten konfigurierbar zu halten und von Haus aus nur einen kleinen Standardsatz anzubieten. Dies bietet Benutzern dann die Möglichkeit, die Gesten ihrem Geschmack entsprechend zu gestalten. Folglich muss es möglich sein, Gesten selber zu erstellen.

- Ohne einen ausgezeichneten Anfang und Ende sind nur einfache Gesten möglich. Sie würden beginnen, sobald das gesuchte Objekt erkannt wird, und enden, sobald es den Sichtbereich der Kamera verlässt. Auf diese Weise würden allerdings nur sehr simple Gesten wie in Abbildung 4.8(a) sinnvoll sein. Besser wäre es aber, wenn auch komplexere Gesten möglich wären, wie sie zum Beispiel in Abbildung 4.8(b) zu sehen sind.

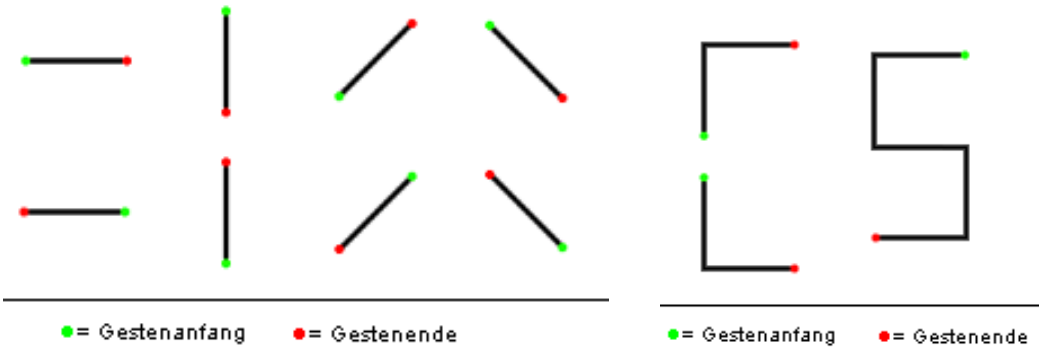
- Ein weitere Überlegung muss auch der Form der Gesten gelten. Welcher Art von Bewegungen sollen unterstützt werden?

  Um die Komplexität vorerst gering zu halten sollte es genügen, wenn eine Geste aus einer Kombination von vier verschiedenen Bewegungen zusammengesetzt werden kann. Hoch, runter, rechts und links. Somit lässt sich zum Beispiel die rechte Geste aus Abbildung 4.8(b) mit dem Bewegungsvektor [left, down, right, down, left] beschreiben.

  Desweitern können mehrere Bewegungen in ein und dieselbe Richtung auf eine gekürzt werden. Dadurch wird aus einem Bewegungsvektor mit den Elementen [left, left, left] der Vektor [left]. Das Ergebnis ist bei beiden gleich, da immer noch eine Bewegung nach links vorliegt. Ihre Länge kann dabei vernachlässigt werden.

4.4.2. Spezifikation

- Eine Geste hat sich aus einem oder mehreren Elementen der Bewegungsmenge [up, down, right, left] zusammenzusetzen. Mehrere aufeinanderfolgende Bewegungen in die gleiche Richtung sind auf eine zu komprimieren.



(a) Einfache (b) Komplexe

Abbildung 4.8.: Gesten

- Um komplexe Gesten zu ermöglichen, sind ein ausgezeichneter Anfang und ein ausgezeichnetes Ende für eine Geste einzuführen. Der Einfachheit halber, und da das System ausschließlich auf der Erkennung einer Farbe basiert, werden Anfang und Ende so definiert, dass das verfolgte Objekt für einen konfigurierbaren Zeitraum stillstehen muss.
- Es muss möglich sein, Gesten zu erstellen und zu löschen. Dazu ist ein entsprechender Dialog auf der Benutzeroberfläche anzubieten. Das Anlegen einer neuen Geste soll dadurch geschehen, dass der Benutzer sie mit der Maus auf diesem Dialog zeichnet.
- Damit eine Geste identifiziert werden kann, muss sie einen Namen erhalten. Dieser ist ihr vom Benutzer zu verleihen.
- Eine Geste setzt sich aus einem Namen und einem Bewegungsvektor zusammen. Beide müssen vorhanden sein, damit eine gültige Geste vorliegt.

4.4.3. Erkennung

Die Arbeitsweise der Erkennung kann grundsätzlich in zwei Bereiche aufgeteilt werden:

1. In die Bewegungserkennung. Sie ermittelt Richtungsangaben anhand der erkannten Positionen des verfolgten Objektes.
2. In die eigentliche Gestenerkennung. Sie ermittelt anhand der Richtungsangaben, ob eine Geste durchgeführt wurde.

Diese beiden Bereiche reichen schon aus, um einfache Gesten zu ermöglichen. Da es aber erforderlich ist, komplexere Gesten zu unterstützen, wird noch ein Zustandsautomat benötigt. Er wird für das Erkennen von Anfang, Durchführung und Ende einer Geste verwendet. Seine Beschreibung ist unter Punkt 4.4.3.3 zu finden.

4.4.3.1. Erkennen der Bewegungen

Für das Erkennen einer Geste ist es notwendig die Bewegung des verfolgten Objektes zu ermitteln. Zu diesem Zweck ermittelt die Bildauswertung bereits laufend seine aktuelle Position und stellt sie in Form eines Punktes in einem zweidimensionalen Koordinatensystem zu Verfügung.

Eine Geste wird anhand von Bewegungsrichtungen beschrieben. Daher gilt es zunächst einmal, anhand der Punkte die Richtung der ausgeführten Bewegung zu ermitteln.

Da die Punkte nacheinander und zeitversetzt eintreffen, ist die Bewegung normalerweise von Punkt zu Punkt zu betrachten. Konkret bedeutet dies, dass jeweils der aktuelle Punkt und dessen Vorgänger zur Ermittlung herangezogen werden. Der Vorgänger dient dabei als Ursprung. Ausgehend von seiner Position erfolgt eine Einteilung in die vier möglichen Richtungen, wie sie in Abbildung 4.9 zu sehen sind.

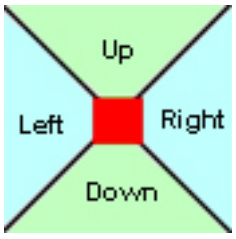


Abbildung 4.9.: Aufteilung in Bewegungsrichtungen

In der Praxis ist es jedoch notwendig, eine gewisse Toleranz zuzulassen, da es aufgrund der natürlichen Bewegung des Menschen und durch qualitative Schwankungen bei der Bildverarbeitung zu Ungenauigkeiten kommt. Diese Äußern sich darin, dass bei Stillstand des

verfolgten Objektes die jeweils aktuelle erkannte Position nicht gleichbleibend ist, sondern vielmehr leicht oszilliert[3].

In diesem Projekt werden zwei Toleranzen verwendet, die jeweils konfigurierbar sind. Die eine kommt zum Tragen wenn das Objekt stillsteht. Die andere wenn es sich bewegt. Letzte ist dabei größer dimensioniert, da es bei der Bewegung zu stärkeren Abweichungen kommen kann. Beide werden jedoch in ein und der selben Art und Weise verwendet.

Es wird nach jeder erfolgreichen Ermittlung einer Bewegung der neueste Punkt als Vorgänger für die nächste Ermittlung gesichert. Anschließend wird jeder nachfolgende Punkt dahingehend geprüft, ob er sich innerhalb der tolerierbaren Entfernung zum gesicherten Punkt befindet. Ist das der Fall, dann wird der Punkt verworfen. Wird der Toleranzbereich jedoch verlassen, dann wird die Ermittlung der Bewegung durchgeführt und der aktuelle Punkt gesichert. In Abbildung 4.10 ist der Toleranzbereich beispielhaft dargestellt. Der gesicherte Punkt ist rot hervorgehoben und der grüne Bereich markiert die Toleranz. Der blaue Punkt in der mittleren Grafik befindet sich im Toleranzbereich und ist zu verwerfen. In der rechten Grafik wurde der Toleranzbereich verlassen und der entsprechende Punkt gesichert, so dass er zukünftig das Zentrum des Toleranzbereiches bildet.

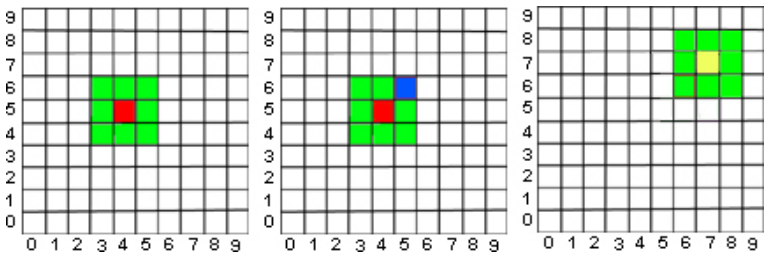


Abbildung 4.10.: Bewegung

Die eigentliche Ermittlung der Richtung erfolgt mit Hilfe des nachfolgenden Pseudocodes, mit dem aktuellen Punkt als „punkt2“ und dem Vorgänger als „ursprung“. Seine Arbeitsweise wird im Anschluss genauer erläutert.

[3] Bildlich gesprochen könnte man sagen, dass der erkannte Schwerpunkt des Objektes um dessen realen „herumflackert“.

```
1  richtung(ursprung, punkt2){
       if( ursprung == punkt2 ) richtung = NOWHERE
3      x = abs( punkt2.x - ursprung.x )
       y = abs( punkt2.y - ursprung.y )
5      if( x < y ){
           if(y > 0) richtung = DOWN
7          else richtung = UP
       }else{
9          if(x > 0) richtung = RIGHT
           else richtung = LEFT
11     }
   }
```

Zum besseren Verständnis sei zunächst einmal erwähnt, dass der Code die Bewegung anhand von Abbildungen im Koordinatensystem erkennt und dabei immer davon ausgeht, dass der vorangegangene Punkt („ursprung“) die logischen Koordinaten (0,0) besitzt. Er arbeitet wie folgt:

Eingangs wird geprüft, ob die Punkte verschieden sind. Wenn nicht, dann liegt keine Bewegung vor. Ansonsten wird in den Zeilen zwei und drei eine Abbildung der Koordinaten vorgenommen, so dass die Betrachtung nun relativ zum mathematischen Ursprung (0,0) vorgenommen werden kann. Durch das Bilden der Absolutwerte werden die Koordinaten gleichzeitig in den I. Quadranten transferiert.

In Zeile fünf erfolgt dann eine Überprüfung, ob der Winkel zwischen dem Punkt (x,y) und der x-Achse, ausgehend vom Ursprung, größer oder kleiner ist als 45°. Der Grund für diese Überprüfung ist die Aufteilung eines Quadranten in zwei verschiedene Bewegungen (siehe Abbildung 4.9). In Abhängigkeit des Ergebnisses wird in den Zeilen sechs bis zehn die Richtung ermittelt, indem ein „virtueller Rücktransfer“ des Punktes (x,y) in seinen ursprünglichen Quadranten vorgenommen wird.

4.4.3.2. Erkennen der Gesten

Da eine Geste mit einem ausgezeichneten Ende aufhört, kann davon ausgegangen werden, dass ihr vollständiger Bewegunsvektor für das Erkennen vorliegt. Als einfachste Variante ist daher lediglich ihr Vektor mit den Vektoren der definierten Gesten zu vergleichen. Bei einer Übereinstimmung handelt es sich um eine gültige Geste, ansonsten wird sie verworfen. In beiden Fällen wird das System über den Ausgang des Vergleichs in Kenntnis gesetzt, damit entweder ein eventuell zugeordnetes Kommando ausgeführt, oder dem Benutzer eine entsprechendes Feedback gegeben werden kann.

Das einfache Vergleichen von Vektoren ist jedoch zu aufwendig, da es von deren Länge, sowie von der Anzahl der definierten Gesten abhängt. Daher wird für die Erkennung ein Baum über alle definierten Gesten erstellt. Seine Knoten werden über die einzelnen Bewegungen ihrer Vektoren verbunden. Ferner beinhaltet jeder Knoten, der den Vektor einer definierten Geste abschließt, deren Namen. Abbildung 4.11 zeigt einen entsprechenden Baum. Er enthält die Gesten „Left“-[L], „Down-Right“-[DR], „Down-Left“-[DL] und „My-U“-[DLU].

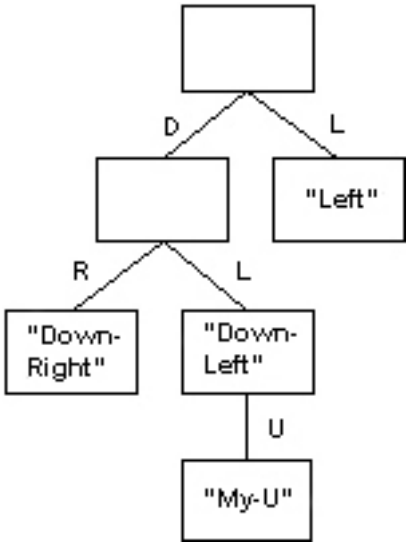


Abbildung 4.11.: Baum der Bewegungsvektoren

Mit Hilfe des Baumes kann nun eine erkannte Geste sehr schnell daraufhin überprüft werden, ob sie einer hinterlegten gleicht. Dazu muss man lediglich mit den Bewegungen der erkannten Geste in dem Baum herabsteigen. Sollte man den Baum dabei verlassen, dann ist die Geste ungültig. Ebenso ist sie ungültig, wenn nach der letzten Bewegung ein Konten erreicht wurde, der keinen Namen enthält. Die gültige Übereinstimmung findet nur dann statt, wenn der zuletzt erreichte Knoten einen Namen enthält. Dieser wird dem System mitgeteilt, damit es dann entsprechend ein Kommando ausführen kann.

Wenn eine Geste als ungültig erkannt wurde, gäbe es allerdings noch die Möglichkeit eine Approximation durchzuführen. Dabei würde man sie als die definierte Geste anerkennen, deren Vektor dem ihrigen am ähnlichsten ist.

In diesem Design wird jedoch bewusst auf diese Möglichkeit verzichtet. Es ist besser eine ungültige Geste als solche zu erkennen, anstatt eine ähnliche auszuwählen. Die Auswahl könnte nämlich dazu führen, dass das anschließend auszuführende Kommando nicht vorhersehbare Folgen nach sich zieht.

4.4.3.3. Erkennungsautomat

Da eine Geste aus mehreren Phasen besteht (siehe 4.4.2), ist es notwendig das die Erkennung unterschiedliche Zustände annehmen kann. Sie bildet daher einen Automaten, wie er in Abbildung 4.12 zu sehen ist. Seine Arbeitsweise wird nachfolgend beschrieben.

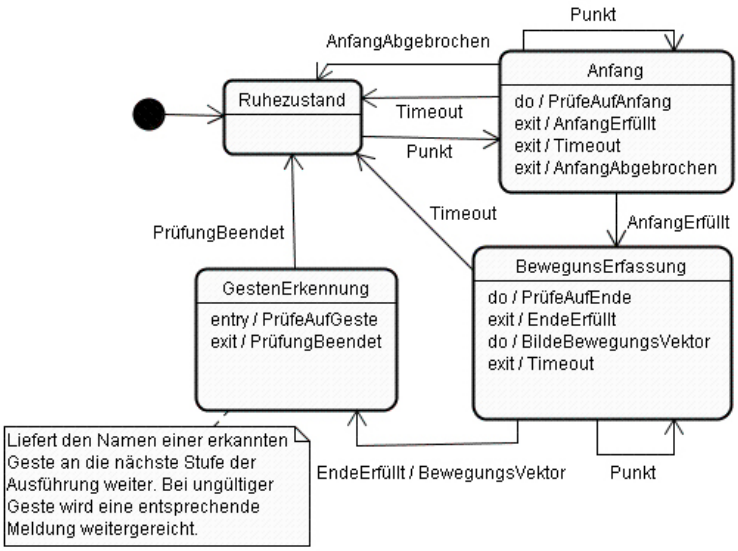


Abbildung 4.12.: Zustände der Gestenerkennung

Anfangs befindet sich die Erkennung im „Ruhezustand“. Sobald nun ein Punkt eintrifft, wechselt sie in den Zustand „Anfang“. Er dient zur Erkennung des Gestenanfangs.

Während sich die Erkennung nun in diesem Zustand befindet, wird bei Eintreffen jeden weiteren Punktes geprüft, ob der Anfang der Geste beendet ist. Da die Bedingung für den Anfang einer Geste, das Stillstehen des verfolgten Objektes für einen bestimmten Zeitraum ist, erfolgt dementsprechend eine Prüfung darauf, ob sich die eintreffenden Punkte innerhalb einer festgelegten Toleranz bewegen. Die Toleranz wird im Rahmen der Bewegungserkennung (Punkt 4.4.3.1) eingehender erläutert. Sollte dies nicht der Fall sein, so wird die Erkennung des Anfangs abgebrochen und es wird wieder in den „Ruhezustand“ gewechselt. Wenn jedoch die Bedingung für den Anfang erfüllt wurde, geht die Erkennung in den Zustand „BewegungsErfassung“ über.

In dem Zustand „BewegungsErfassung“ wird, anhand der eintreffenden Punkte, die Bewegung des verfolgten Objektes ermittelt. Anhand dieser Bewegungen wird ein Bewegungsvektor für die augenblickliche Geste erstellt. Ebenso erfolgt hier die Überprüfung, ob das Ende

der Geste erfüllt wurde. Es geschieht analog zur Erkennung des Anfangs. Sobald das Ende erkannt wurde, wechselt der Zustand zur „Gestenerkennung“. Bei dem Wechsel wird der gebildete Bewegungsvektor übergeben.

Im Zustand „Gestenerkennung“ findet die eigentliche Überprüfung statt, ob die gerade erfasste Geste einer, dem System bekannten, Geste entspricht. Die Durchführung dieser Überprüfung wird unter Punkt 4.4.3.2 beschrieben. Nach der Überprüfung wird dann entweder der Name der erkannten Geste, oder eine Meldung über den Fehlschlag der Erkennung an das System geliefert, damit es entsprechen weitere Schritte einleiten kann. Anschließend geht das System wieder in den „Ruhezustand“ über.

In den Zuständen „Anfang“ und „BewegungsErfassung“ ist zusätzlich noch ein „Timeout“ möglich. Es tritt ein, wenn für einen festgelegten Zeitraum keine Punkte mehr eintreffen. Dies kann immer dann passieren, wenn das zu verfolgenden Objekt nicht länger von der Kamera erkannt wird. In dem Fall wird davon ausgegangen, dass die Geste abgebrochen wurde. Als Folge dessen, wechselt das System wieder in den „Ruhezustand“.

Die nachfolgende Abbildung (4.13) stellt vereinfacht dar, wie der das restliche System über den Status der Erkennung in Kenntnis gesetzt wird.

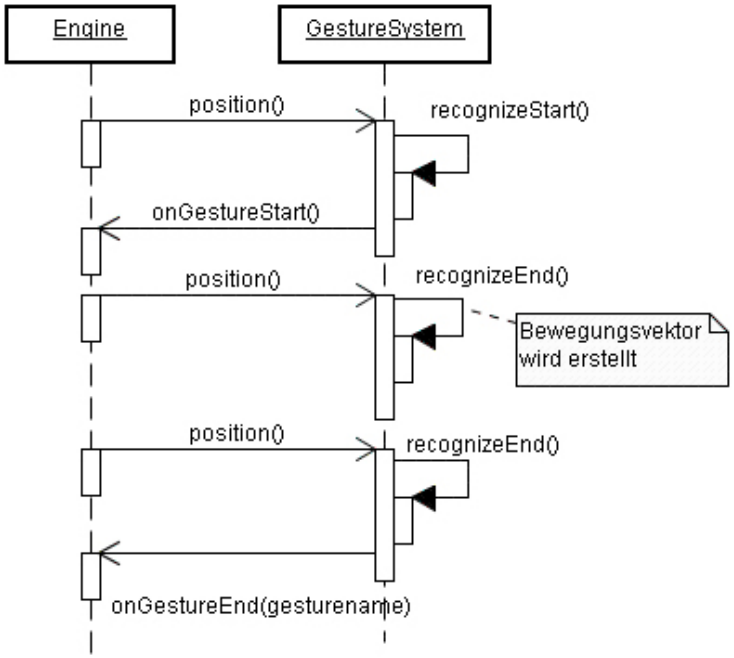


Abbildung 4.13.: Meldungen der Gestenerkennung

Die „Engine“ liefert stetig die neue Position des verfolgten Objektes. Die Erkennung erfasst daraufhin zuerst den Anfang einer Geste und meldet den Erfolg („onGestureStart()“). An-

schließend erstellt sie den Bewegungsvektor, während sie zugleich das Ende zu erkennen versucht. Bei erkanntem Ende der Geste wird das System darüber informiert und erhält den Namen der jeweiligen Geste („onGestureEnd(gesturename)“).

4.5. Kommandos

Die Kommandos sind dazu gedacht, dass bei dem Erkennen einer Geste eine Aktion ausgelöst werden kann. Um welche Aktion es sich dabei handelt, hängt von dem jeweiligen Kommando ab. Sie sollen vom Benutzer erstellt bzw. konfiguriert werden können.

Um für die Ausführung beliebiger Kommandos eine einheitliche Struktur zu erhalten, und um das System in Zukunft leicht um weitere Kommandos ergänzen zu können, wird das „Command-Pattern“ verwendet. Dazu werden die Eigenschaften möglicher Kommandos von der Klasse „AbstractCommand“ (Abbildung 4.14) festgelegt. Alle Kommandos müssen von ihr abgeleitet werden.

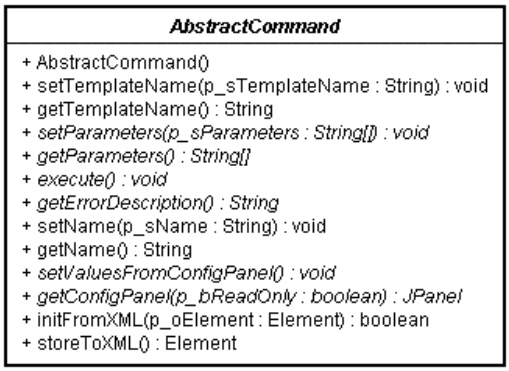


Abbildung 4.14.: AbstractCommand

Damit es möglich wird, dass Kommandos ohne weitere Anpassungen der Benutzeroberfläche hinzugefügt werden können, muss jedes Kommando ein Fenster zur Verfügung stellen, dass zur Konfiguration des Kommandos verwendet wird. Dieses Fenster wird dann bei Bedarf innerhalb der Benutzeroberfläche dargestellt.

Als nächstes werden die Kommandos in zwei logische Gruppen unterteilt:

- „TemplateCommand“ - Diese Bezeichnung steht für die Klasse eines Kommandos. Dies kann jede Klasse sein, die von AbstractCommand abgeleitet wurde. Mit ihnen kann der Benutzer neue Kommandos erstellen.

- „ConcreteCommand“ - Bei diesem Kommando handelt es sich um eine Instanz (ein Objekt) eines „TemplateCommands“. Ein vom Benutzer erstelltes und konfiguriertes Kommando, dass ausgeführt werden kann.

Diese Einteilung wird vorgenommen, um dem Benutzer das Erstellen und Konfigurieren von Kommandos zu ermöglichen. Selbiges soll in der folgenden Form geschehen können:

1. Zuerst wählt der Benutzer ein „TemplateCommand“ aus.
2. Das System erstellt davon eine Instanz und präsentiert dem Benutzer deren Konfigurationsfenster.
3. Der Benutzer führt die Konfiguration durch, indem er auf Fenster entsprechende Einstellungen vornimmt. Anschließend bestätigt er den Vorgang.
4. Das System ordnet das nunmehr konfigurierte Kommando in die Menge der bereits vorhandenen ein und stellt es für die Zuordnung zu Gesten zur Verfügung. Von diesem Zeitpunkt an wird das Kommando als „ConcreteCommand“ bezeichnet, da es sich um ein konkretes und ausführbares Kommando handelt.

Da die Steuerung der externen Anwendungen unter Verwendung einer ebenfalls externen Instrumentierungskomponente vorgenommen wird (siehe Punkt 4.7), ist zur Zeit nur ein Kommando vorgesehen. Es soll ein externes Programm starten, und ihm dabei Parameter übergeben können. Die folgende Abbildung zeigt seinen Aufbau.

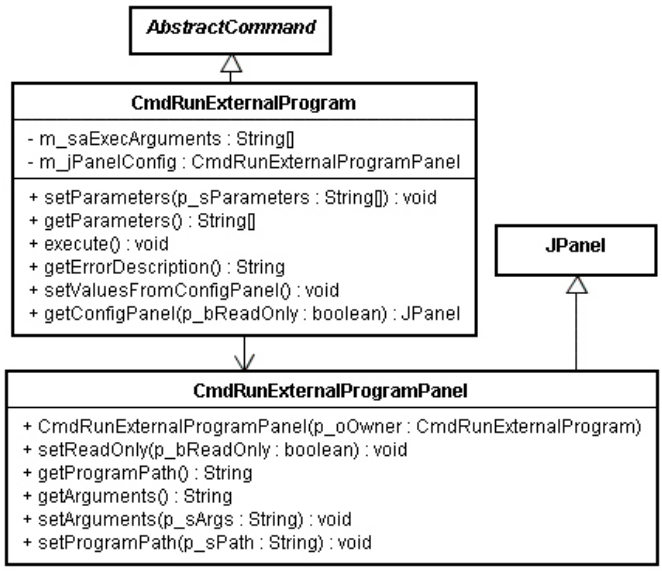


Abbildung 4.15.: Das Kommando - „Run a Program“

4.6. Zuordnung von Kommandos zu Gesten

Damit bei dem Erkennen einer Geste ein Kommando ausgeführt werden kann, ist eine entsprechende Zuordnung vorzusehen. Sie soll dynamisch erfolgen können, damit der Benutzer in der Lage ist sie zur Laufzeit vornehmen bzw. ändern zu können.

Es sind dabei die folgenden Zuordnungen vorzusehen:

- Einer Geste soll kein, oder maximal ein Kommando zugeordnet werden können.
- Ein Kommando soll keiner, oder mehreren Gesten zugeordnet werden können.

Die Zuordnungen können in Form einer einfachen Map realisiert werden. Die Geste wird dabei als Schlüssel, und das zugeordnete Kommando als Wert verwendet. Möglich ist dies aufgrund der Tatsache, dass eine Geste immer nur einem Kommando zugeordnet werden kann.

Um die Zuordnungen vornehmen zu können, ist ein entsprechender Dialog zu erstellen.

4.7. Instrumentierung von Anwendungen

Das Instrumentieren von Anwendungen ist stark mit dem zugrundeliegenden Betriebssystem verbunden. Da eine entsprechende Einbindung des Windows API mittels des Java Native Interfaces jedoch nicht im Rahmen dieser Arbeit erfolgen kann, wird für die Instrumentierung die Skriptsprache AutoIt verwendet.

Zur Verwirklichung des gesetzten Zieles, in Form der Steuerung einer Präsentation mit PowerPoint, reicht es aus, wenn lediglich zwei verschiedene Tastendrücke emuliert und an PowerPoint gesendet werden. Dabei handelt es sich um „Bild rauf“ zum Anzeigen der nächsten Folie und „Bild runter“ zum Anzeigen der vorherigen.

Es sind daher entsprechende Skripte in AutoIt zu erstellen.

Die Skripte sind mit Hilfe des „CmdRunExternalProgram“- Kommandos (siehe 4.5) auszuführen.

4.8. Feedback

Einen wichtigen Punkt stellt das Feedback dar. Der Benutzer muss zum Beispiel darüber informiert werden, ob das System eine Geste erkannt hat, oder ob ein Kommando nicht ausgeführt werden konnte. Dabei muß berücksichtigt werden, dass das Anwendungsfenster des zu erstellenden Systems nicht immer sichtbar sein kann. Bei einer Präsentation ist es sogar zwingend erforderlich, dass es nicht im Vordergrund bleibt.

Daher ist vorgesehen, dass das System über ein Logfenster verfügen muss. In diesem Fenster werden alle Statusmeldungen des Systems ausgegeben. Sofern der Benutzer nicht gerade direkt mit der Benutzeroberfläche des Systems interagiert, sind Dialoge zur Ausgabe von Meldungen nur dann zu verwenden, wenn ein schwerer Programmfehler vorliegen sollte, der ein weiteres sinnvolles Verwenden des Systems verhindert.

Darüber hinaus sind der Start und das Ende der Erfassung durch Ausgabe von Tönen zu verdeutlichen. Die Ausgabe der Töne soll deaktiviert werden können.

4.9. Persistenz der Konfigurationsdaten

Damit nicht bei jedem Start des Systems alle Einstellungen erneut vorgenommen werden müssen, sind sämtliche konfigurierbaren Werte in eine Datei zu schreiben. Die Speicherung hat dabei im XML-Format zu erfolgen.

5. Realisierung

Dieses Kapitel widmet sich der Implementierung des Prototypen „***CamCon***“, welcher im Rahmen dieser Arbeit erstellt wurde. Seine Realisierung erfolgte gemäß dem Design.

Punkt 5.1 beschreibt die verwendete Entwicklungsplattform und unter Punkt 5.2 werden fertige Softwarebibliotheken aufgeführt, welche zur Realisierung einiger Aspekte des Prototypen verwendet wurden. Ebenso wird dort erwähnt, für welche sie zum Einsatz kamen.

Das farbige Objekt, das für die Erkennung von Gesten verwendet wurde, ist unter Punkt 5.3 aufgeführt.

Die systemabhängige Instrumentierungskomponente zum Ansteuern von PowerPoint und deren Einbindung in CamCon sind unter Punkt 5.4 aufgeführt.

Unter Punkt 5.5 wird die Oberfläche des erstellten Prototypen und damit auch die typische Arbeitsweise mit ihm vorgestellt.

Abgeschlossen wird dieses Kapitel mit Punkt 5.6, unter dem durchgeführte Performance-Messungen und deren Ergebnisse zu finden sind.

5.1. Entwicklungsplattform

Hardware

Für die Entwicklung kam ein PC mit einem AMD® Athlon™ XP 2200+ Prozessor, 1024 MB DDR-RAM und einer USB 2.0-Schnittstelle zum Einsatz.

Als Videoquelle diente eine USB 2.0-Webcam der Firma Logitech, Model QuickCam Web. Die Kamera verfügt an der Vorderseite über einen Ring zum manuellen Einstellen des Fokus und sie unterstützt fünf verschiedene Auflösungen bei 24 Bit Farbtiefe. Diese sind 160x120, 176x144, 320x240, 352x288 sowie 640x480 Pixel. Sie liefert leicht verrauschte Bilder von mittlerer Qualität, wie sie von einer durchschnittlichen Webcam zu erwarten sind.

Software

Als Betriebssystem wurde Microsoft Windows XP Professional verwendet. Die Programmierung erfolgte in Java unter Verwendung der JBuilder 8 - Entwicklungsumgebung von Borland.

5.2. Verwendete Bibliotheken dritter

Java Media Framework

Um unter Java das Ansprechen einer Kamera zu ermöglichen, wurde das Java Media Framework von [Sun b] verwendet. Es erlaubt eine plattformunabhängige Nutzung von Kameras, deren Treiber die „Video for Windows“-Schnittstelle von Microsoft Windows, respektive die „Video4Linux“-Schnittstelle unter Linux, unterstützen.

Das Framework diente als Grundlage zum Erstellen des FrameGrabbers. Er ist dabei an ein Beispiel von den Entwickler-Seiten von Sun [Sun c] angelehnt. Das dortige Beispiel ist einfacher Natur und zeigt die grundlegende Vorgehensweise um einen Grabber zu erstellen. Es wurde entsprechend modifiziert, um unter anderem das Ändern der Auflösung zur Laufzeit zu ermöglichen.

Eine andere Alternative hätte auch die Verwendung des „Camera Frameworks“ aus der Diplomarbeit von Ilia Revout [Revout (2003)] dargestellt. Es handelt sich dabei um ein Framework, das in C++ erstellt wurde und welches sowohl das Ansprechen der Kamera, als auch das Erstellen, Testen und Verwenden von Filtern ermöglicht. Leider ist es jedoch nur unter Windows einsetzbar, so dass die Plattformunabhängigkeit entsprechend verloren geht. Ferner stellte sich heraus, dass eine Verwendung mittels des Java Native Interfaces nur unter hohem Aufwand möglich gewesen wäre. Dies hätte jedoch den Zeitrahmen dieser Arbeit überschritten.

JDOM

Die Realisierung der Persistenz für die Konfigurationsdaten erfolgte mit Hilfe der XML-Bibliothek JDOM [JDOM-Project]. Ein Beispiel für eine damit erstellte Konfigurationsdatei ist in Anhang A.2 zu finden.

5.3. Das verwendete Objekt für die Erkennung

Im Rahmen dieser Arbeit wurde als Objekt für die Gestenerkennung der Handschuh aus Abbildung 5.1 verwendet. Er ist aus Wolle und verfügt er über eine matte, wenig reflektierende Oberfläche.

Abbildung 5.1.: Das verwendete Objekt für die Erkennung

5.4. Verwendete Instrumentierungskomponente - „AutoIt“

Für das Instrumentieren von Anwendungen unter Windows wurde AutoIt V3 [AutoIt] verwendet. Es handelt sich dabei um eine Skriptspache, die große Ähnlichkeit mit Visual Basic von Microsoft besitzt und das automatisieren der Windows-Oberfläche ermöglicht. Sie erlaubt das Simulieren von Tastendrücken und Maus-Ereignissen, sowie eine direkte Manipulation von Anwendungsfenstern und darauf dargestellten Kontroll-Elementen.

Erstellte Skripte zur Steuerung von PowerPoint

Zur Erzeugung der benötigten Tastendrücke für die Steuerung von PowerPoint, sind zwei verschiedene Skripte erstellt worden. Eins für das Anzeigen der nächsten Folie und eins für die vorherige. Da beide Skripte analog zueinander arbeiten und sich nur in dem emulierten Tastendruck unterscheiden, wird an dieser Stelle lediglich auf das erste der beiden eingegangen. Für das andere sei auf den Anhang A.1 verwiesen. Ausgeführt werden die Skripte unter

Windows, indem das Programm „Autoit3.exe“ mit dem jeweiligen Skriptnamen als Parameter gestartet wird.

„Nächste Folie anzeigen“

```
1  WinActivate("PowerPoint-")
2  If WinActive("PowerPoint-") Then
3      Send("{PGDN}")
4  EndIf
```

Das Skript beginnt damit, dass es in der ersten Zeile versucht ein Fenster zu aktivieren[1], dessen Titel mit der Zeichenfolge „PowerPoint-“ anfängt. In der zweiten Zeile erfolgt eine Überprüfung darauf, ob ein entsprechendes Fenster auch wirklich den Fokus erhalten hat. Sie ist für den Fall gedacht, dass kein solches Fenster vorhanden ist bzw. das gewünschte nicht aktiviert werden konnte. Auf diese Weise wird verhindert, dass ungewollte Aktionen mit unabsehbaren Folgen ausgelöst werden, wenn ein anderes Fenster den Fokus hat. Die Anweisung in Zeile drei sendet nämlich den Tastendruck „Bild runter“ einfach an das aktive Fenster. Es ist ihr dabei egal, um welches es sich dabei handelt.

Erstellte CamCon-Kommandos zum Ausführen der Skripte

Um die erstellten AutoIt-Skripte ausführen zu können wurden zwei entsprechende „Run a program“-Kommandos mit Hilfe von CamCon erstellt. Sie erhielten die Namen „PowerPoint-Prev“ und „"PowerPoint-Next". Als Parameter wurden zum Einen der Pfad zu AutoIt.exe („./autoit-v3/AutoIt3.exe“) und zum Anderen der Pfad zu dem jeweiligen Skript („./autoit_scripts/PowerPoint-PGUP.au3“ bzw. „./autoit_scripts/PowerPoint-PGDN.au3“) als Parameter übergeben.

[1] Ein Fenster zu aktivieren bedeutet, dass es in den Vordergrund gebracht wird und den Eingabefokus erhält.

5.5. Der Prototyp - „CamCon“

In den nachfolgenden Unterpunkten werden die einzelnen Bereiche und Elemente der Anwendung beschrieben. Als Reihenfolge für die Beschreibungen wurde dabei die gleiche gewählt, wie sie normalerweise auch bei der Benutzung der Anwendung auftritt.

5.5.1. Die Registerblätter

- „Camera Setup“ (Abbildung 5.2)

  Dieses Registerblatt dient zur Auswahl der Kamera und der Bildauflösung, die von CamCon verwendet werden sollen.

  1. Mit Hilfe der beiden Dropdown Listen wählt man die Kamera und die Auflösung aus.

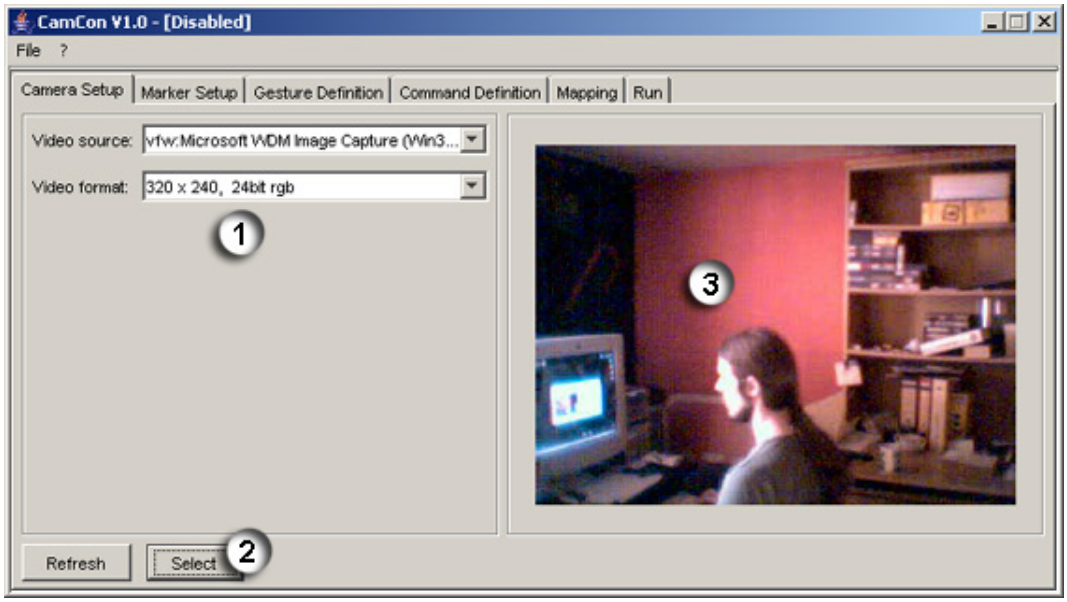


Abbildung 5.2.: Registerblatt - „Camera Setup“

  2. Mit dem Knopf „Select“ wird die Auswahl einer Kamera bestätigt.
     Der Knopf „Refresh“ dient zur Aktualisierung der Listen von Punkt 1, für den Fall, dass eine neue Kamera angeschlossen wurde.
  3. In diesem Bereich wird nach der bestätigten Auswahl von Kamera und Format eine Vorschau auf den Videostrom gegeben.

- „Marker Setup“ (Abbildung 5.3)

  Dieses Registerblatt dient zum Einrichten des zu verfolgenden Objektes, welches der Einfachheit halber als Marker bezeichnet wird. Hier werden seine Farbe und deren Toleranz eingestellt.

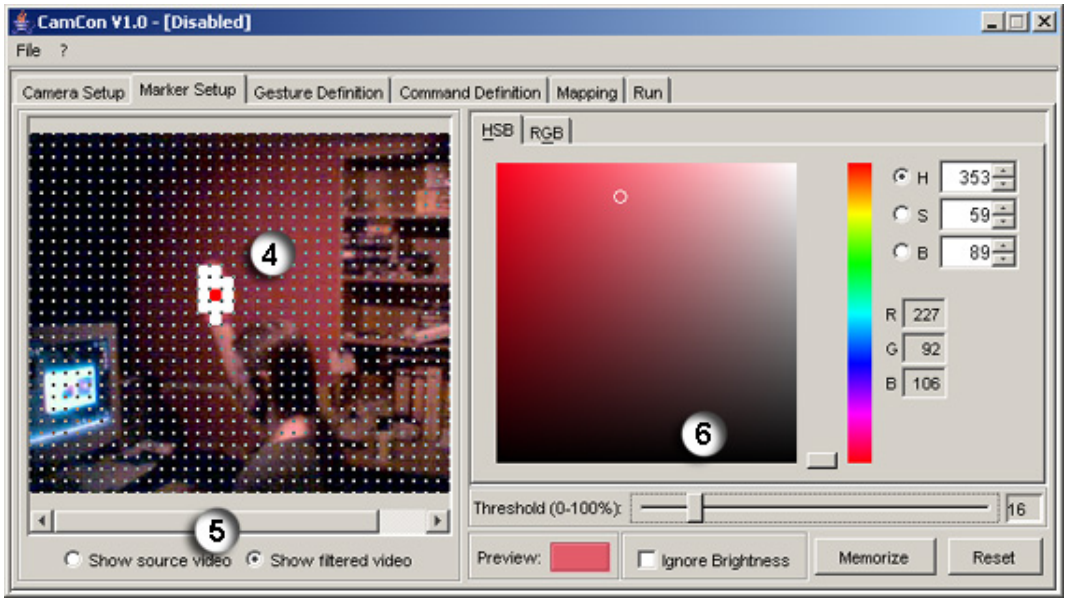


Abbildung 5.3.: Registerblatt - „Marker Setup“

4. In diesem Bereich wird der Videostrom der Kamera entweder ungefiltert oder gefiltert dargestellt. Die Art der Darstellung hängt dabei von der Auswahl ab, welche unter Punkt 5 getroffen wird. Die gefilterte Darstellung dient zur Kontrolle, ob der Marker auch korrekt erkannt wird.

   Ferner kann hier auch mit Hilfe der Maus die Farbe des Markers ausgewählt werden. Dazu wählt man einen Punkt in dem dargestellten Bild aus, indem man ihn mit der linken Maustaste angeklickt. Seine Farbe wird dann automatisch nach Punkt 6 übernommen und dargestellt.

5. Mit Hilfe der Knöpfe in diesem Bereich kann die Art der Videoausgabe umgeschaltet werden, die in 4 dargestellt wird.

6. Dieser Bereich dient zur Feineinstellung der zu verfolgenden Farbe. Es werden dazu mehrere Steuerelemente bereitgestellt.

   Im oberen Bereich befindet sich ein Farb-Auswahldialog, in dem die Farbe durch Auswahl mit der Maus oder durch Eingabe ihres Farbwertes spezifiziert werden kann. Darunter ist ein Schieberegler zum Einstellen der Toleranz angeordnet. Sie

gibt an, um wieviel Prozent die einzelnen Farbwerte von denen der ausgewählten Farbe abweichen dürfen.

Unter dem Schieberegler befinden sich von links nach rechts gehend die nachfolgenden Elemente. Zuerst ist dort ein Feld, dass die ausgewählte Farbe etwas übersichtlicher darstellt. Danach kommt ein Auswahlfeld, welches das Ignorieren des Helligkeitswertes bei der Farberkennung ein- bzw. ausschaltet. Rechts sind dann noch zwei Knöpfe um die aktuell eingestellte Farbe zu speichern oder wiederherzustellen. Sie sind dazu gedacht, um sich bei der Feinabstimmung ein gutes Zwischenergebnis merken zu können, bevor man weitere Änderungen vornimmt und dabei evtl. die Erkennung verschlechtert.

- „Gesture Definition“ (Abbildung 5.4)

  Auf diesem Registerblatt werden die Gesten verwaltet, die CamCon erkennen soll.

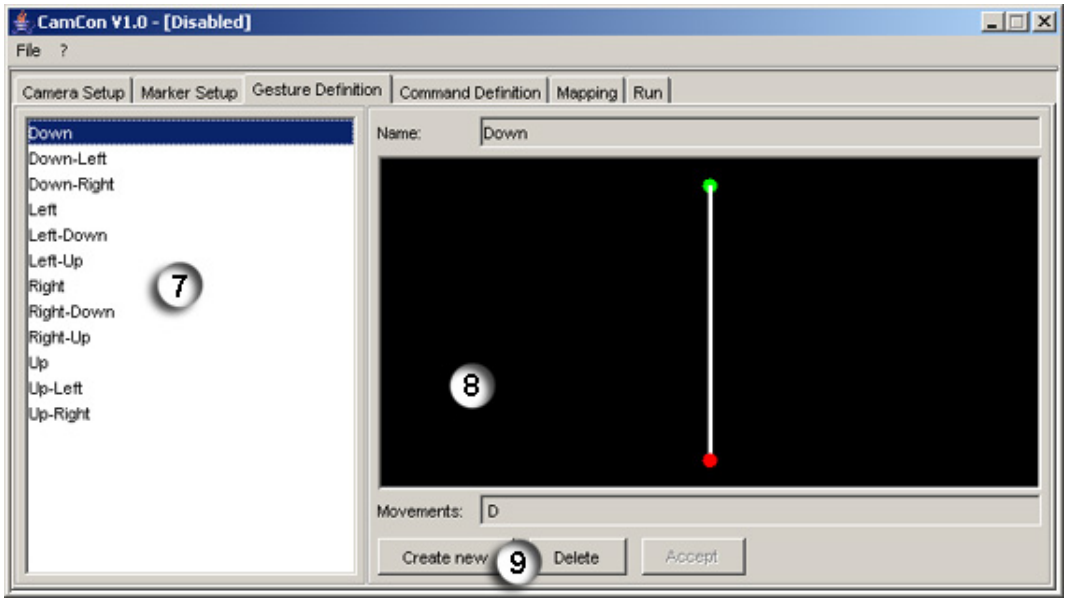


Abbildung 5.4.: Registerblatt - „Gesture Definition“

7. Diese Liste enthält die Namen aller definierten Gesten. Aus ihr können Gesten ausgewählt werden, um sie entweder anzuzeigen oder zu löschen.

8. Im oberen Teil wird der Name der ausgewählten Geste noch einmal aufgeführt und in dem mittleren Bereich wird sie grafisch dargestellt. Der grüne Punkt markiert dabei ihren Anfang und der rote ihr Ende.

   Das Textfeld darunter zeigt ihre einzelnen Bewegungen in Textform an.

9. Die Knöpfe im unteren Bereich dienen zum Anlegen einer neuen Geste bzw. zum Löschen der ausgewählten.

   Das Anlegen einer neuen Geste ist in der nachfolgenden Abbildung 5.5 zu sehen.

– Definieren einer neuen Geste (Abbildung 5.5)

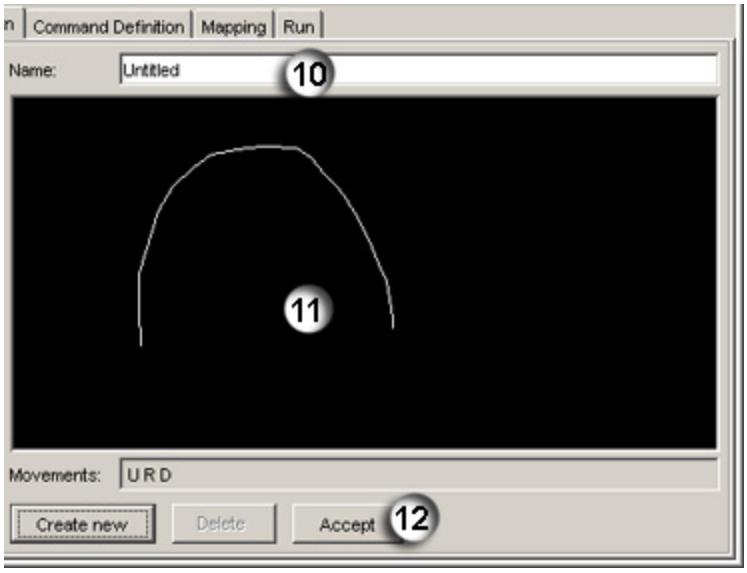


Abbildung 5.5.: Definieren einer neuen Geste

10. In dem oberen Textfeld wird der Name für die neue Geste eingegeben.

11. Das mittlere Feld dient nun der Definition der Gestenbewegungen, indem sie bei gedrückter linker Maustaste mit der Maus gezeichnet wird. Gleichzeitig werden die Bewegungen zur Kontrolle in dem darunter liegenden Textfeld ausgegeben.

12. Mit dem Knopf „Accept“ wird die neue Geste übernommen und anschließend dargestellt (siehe Abbildung 5.6). Sollte bereits eine Geste mit gleichem Namen und / oder mit den gleichen Bewegungen vorhanden sein, so erfolgt vorher noch eine Nachfrage in Form eines Dialoges.

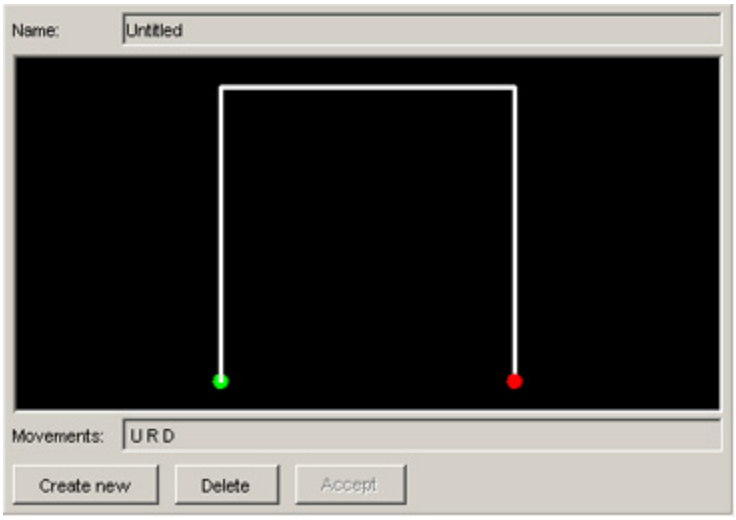


Abbildung 5.6.: Ansicht der neuen Geste

- „Command Definition“ (Abbildung 5.7)

  Auf diesem Registerblatt werden die Kommandos verwaltet.

  13. Auf der linken Seite wird eine Übersicht über die verfügbaren Kommandomuster dargestellt. Auf der Basis dieser Muster kann ein neues Kommando definiert werden. Dazu ist das gewünschte Muster auszuwählen und der Knopf „Create..“ zu drücken. Es wird dann ein Dialog geöffnet, in dem der Name und die Parameter für das neue Kommando eingegeben werden können.

      Der Inhalt des Dialoges ist dabei abhängig von dem jeweiligen Muster. Abbildung 5.8 zeigt ein Beispiel anhand des momentan verfügbaren „Run a program“. Der Dialog wird dort allerdings zum Bearbeiten eines Kommandos benutzt und daher ist das Eingabefeld für den Namen deaktiviert.

  14. Auf der rechten Seite werden die definierten Kommandos aufgelistet. Mit Hilfe der Knöpfe am unteren Rand kann das jeweils ausgewählte bearbeitet oder gelöscht werden.

      Die nachfolgende Abbildung 5.8 zeigt beispielhaft das Bearbeiten eines Kommandos. Der Inhalt des Dialog hängt dabei, wie bereits unter Punkt 13 erwähnt, von der Art des Kommandos ab.

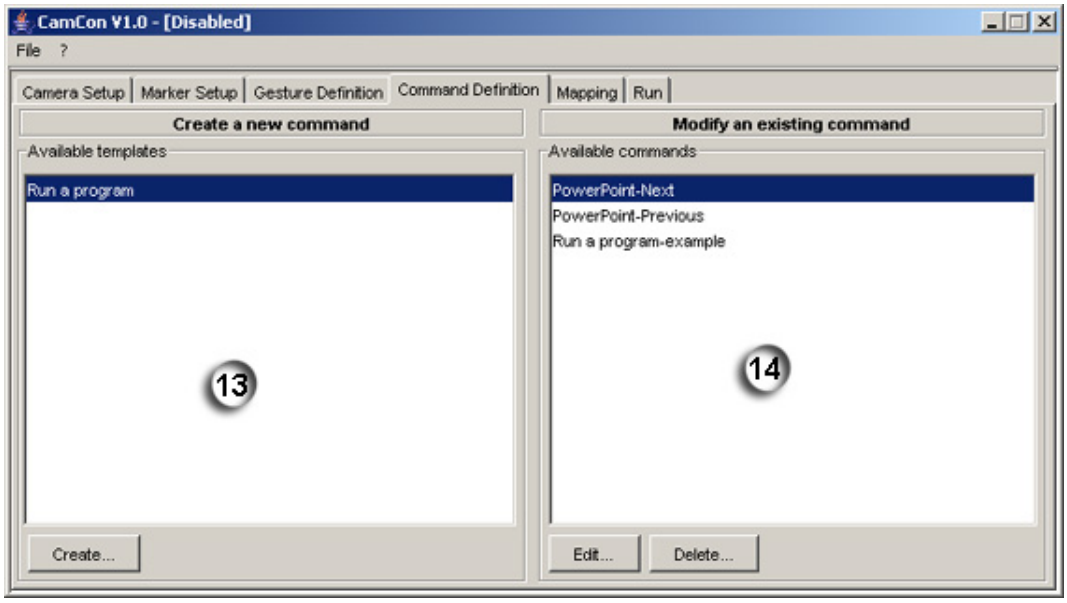


Abbildung 5.7.: Registerblatt - „Command Definition“

– Bearbeiten eines Kommandos (Abbildung 5.8)

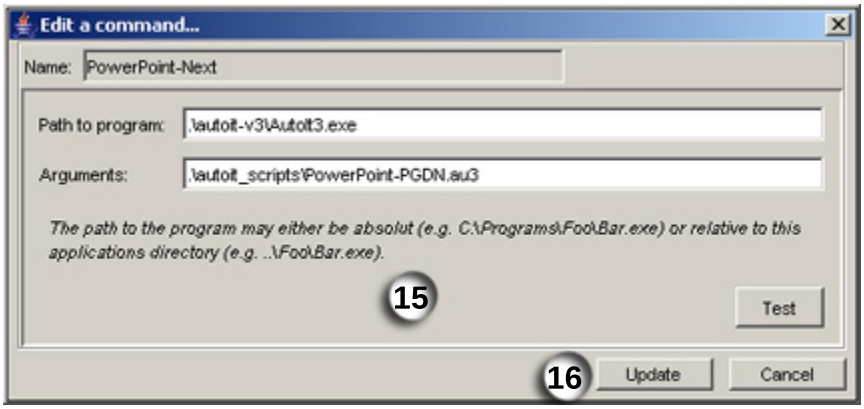


Abbildung 5.8.: Bearbeiten eines Kommandos

15. Innerhalb dieses Bereiches können die Parameter des zu bearbeitenden Kommandos geändert werden. Da dem abgebildeten Kommando das Kommandomuster „Run a program“ zugrunde liegt, können hier der Pfad zum auszuführenden Programm und die, ihm zu übergebenden, Parameter eingegeben werden. Ebenso ist als Besonderheit dieses Kommandos der Knopf „Test“ vorhanden, mit dem das Programm probeweise ausgeführt werden kann.

16. Der untere Bereich des Dialogs enthält je einen Knopf zum Speichern der Änderungen und zum Abbrechen des Vorgangs.

- „Mapping“ (Abbildung 5.9)

  Auf diesem Registerblatt können die Zuordnungen von Kommandos zu Gesten verwaltet werden. Die Zuordnungen werden hierbei in Form einer zweispaltigen Liste über alle vorhandenen Gesten angezeigt.

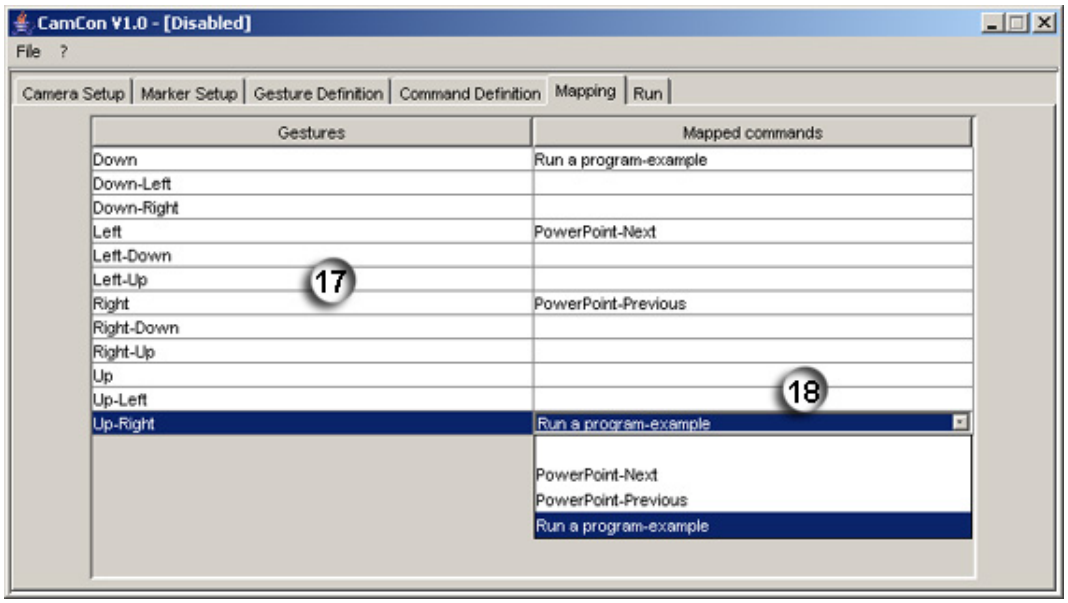


Abbildung 5.9.: Registerblatt - „Mapping“

17. Die linke Spalte einer Zeile enthält den Namen der jeweiligen Geste.

18. In der rechten Spalte sind die aktuell zugeordneten Kommandos aufgeführt. Hier findet auch das eigentliche Zuordnungen statt. Dazu klickt man mit der linken Maustaste in die Zeile, die auch die gewünschte Geste enthält. Es wird dann eine Dropdown Liste mit allen verfügbaren Kommandos angezeigt, aus der man das gewünschte auswählen kann.

    Um eine Zuordnung aufzuheben, muss der erste (leere) Eintrag aus der Dropdown Liste ausgewählt werden.

- „Run“ (Abbildung 5.10)

  Dieses Registerblatt dient zur Aktivierung bzw. Deaktivierung der Gestenerkennung, sowie zur Einstellung weiterer Parameter für die Erkennung von Gesten.

  19. Mit dem obersten Schieberegler kann die Zeit eingestellt werden, wie lang der Marker für das Erkennen von Gestenanfang und Gestenende jeweils stillstehen muss.

      Bei dem mittleren wird die Zeit eingestellt, die CamCon zwischen dem Erkennen von zwei Gesten wartet. Während dieses Zeitraumes werden keine Gesten akzeptiert.

      Mit dem unteren Regler wird der Zeitraum eingestellt, für wie lange der Marker verloren gehen darf, bevor das Erkennen der aktuellen Geste abgebrochen wird.

  20. Die beiden Auswahlfelder erlauben das Ein- und Ausschalten der Tonausgabe bei dem Erkennen von Gestenanfang und Gestenende.

  21. Dieser Knopf aktiviert bzw. deaktiviert die Gestenerkennung.

  22. Dieses Feld stellt eine Historie über die Erkennung von Gesten und das Ausführen von Kommandos dar. Es zeigt entsprechende Meldungen an, ob eine gültige Geste gefunden wurde. Ob ihr ein Kommando zugeordnet ist und ob bei der Ausführung des Kommandos Fehler auftraten.

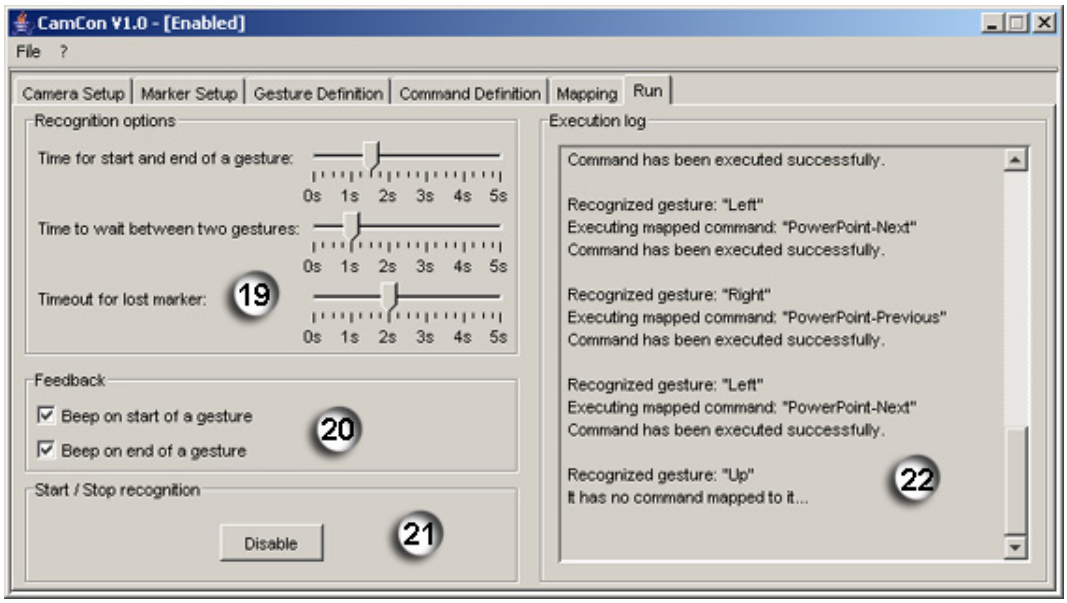


Abbildung 5.10.: Registerblatt - „Run“

5.5.2. Die Menüleiste

In diesem Abschnitt wird die Funktionalität der Menüleiste beschrieben.

- „File“ (Abbildung 5.11)
  - „Load config...“ öffnet einen Dialog, mit dessen Hilfe eine gespeicherte Konfiguration von CamCon geladen werden kann. Nach dem Laden einer Datei ist die Gestenerkennung deaktiviert und es wird das Registerblatt zum Auswählen der Kamera (siehe Abbildung 5.2) angezeigt.
  - „Save config“ speichert die aktuelle Konfiguration von CamCon in der Datei, aus der sie geladen wurde.
  - „Save config as...“ öffnet einen Dialog, mit dessen Hilfe die aktuelle Konfiguration unter einem einzugebenden Pfad gespeichert werden kann.
  - „Exit“ beendet die Anwendung. Nicht gespeicherte Konfigurationsdaten gehen verloren.

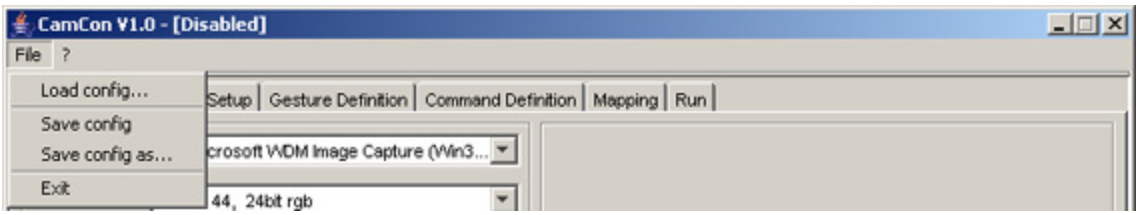


Abbildung 5.11.: Menüleiste - „File“

- „?“ (Abbildung 5.12)
  - „About“ öffnet einen Dialog mit Informationen über CamCon.

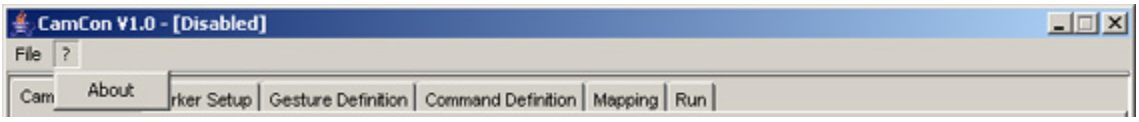


Abbildung 5.12.: Menüleiste - „?“

5.6. Evaluierung

Während der Entwicklung und nach Fertigstellung wurde das System auf der verwendeten Entwicklungsplattform getestet.

Die Tests wurden unter folgenden Bedingungen durchgeführt:

- Keine sonstige Aktivität auf dem Rechner.
- Mittlere Aktivität durch Ausführen mehrerer Programme wie Audiowiedergabe, Office-Anwendungen, Dateimanager.
- Hohe Aktivität, simuliert durch das Packen eines 500 MB großen Archivs mit dem Packprogramm WinRar 3.0 unter Verwendung der höchsten Kompressionsstufe.
- Verwendung verschiedener Fensterstadien des CamCon-Fensters (im Vordergrund, im Hintergrund, normal, minimiert).
- Aufruf des Java-Interpreters mit speziellen Parametern von der Kommandozeile aus. Siehe dazu auch Punkt 5.7.

Bei keiner bzw. mittlerer Aktivität war die Ansprechzeit der Erkennung vollkommen zufriedenstellend, mit Ausnahme einer Verzögerung bei aktiver Darstellung des gefilterten Videostroms auf dem Registerblatt „Marker Setup“. Die Verzögerung muss dabei jedoch als normal bezeichnet werden.[2] Ihre Stärke hängt von der verwendeten Videoauflösung in CamCon und der jeweiligen Auslastung des Rechners ab.

Ohne die Darstellung des gefilterten Videostroms ergaben Messungen, die mit Hilfe des Taskmanagers von Windows durchgeführt wurden, dass selbst bei einer verwendeten Auflösung von 640x480 Pixeln der Speicherbedarf im Durchschnitt „gerade“ mal 45 MB bei einer Prozessorauslastung von 70 % betrug. In Anbetracht des verarbeiteten Datenvolumens und der Natur von Java sind das recht annehmbare Werte. Das Starten und Ausführen von anderen Programmen konnte weiterhin flüssig durchgeführt werden.

Bei hoher Aktivität wurde es jedoch sehr problematisch und CamCon kam fast zum Erliegen. Dieser Umstand liegt allerdings in der Natur der Sache begründet, da in solch einem Fall nicht genügend Rechenleistung für das Verarbeiten der Bilder zur Verfügung steht. Umgehen lässt sich dies nur dadurch, dass man dem Prozess von CamCon eine höhere Priorität zuweist.

[2] Es stellte sich leider heraus, dass Java mit dem farbigen Hervorheben des Rasters und des Erkennungsergebnisses schon ein wenig überfordert ist. Dank der Einführung der Filter-Threads im Design tritt jedoch kein Ruckeln durch verlorengegangene Bilder auf, sondern die Ausgabe erfolgt nur insgesamt träger.

5.7. Notwendige Optionen der Java-Umgebung

Bereits in frühen Tests stellte sich heraus, dass der Garbage-Collector von Java zu großen Problemen bei der Ausführung von CamCon führte. Er ist sehr oft aktiv geworden und sorgte für ein starkes Ruckeln im Videostrom. Dieses Verhalten lässt sich jedoch über die Kommandozeile abstellen, indem der Java-Interpreter mit dem Parameter „-Xincgc“ aufgerufen wird. Er sorgt dafür, dass Java an Stelle des normalen Garbage-Collectors einen inkrementellen verwendet. Im Gegensatz zum Normalen läuft dieser ständig mit und verhindert dadurch, dass es zu Pausen durch den Start des Collectors bzw. durch langwieriges Aufräumen des Speichers kommt.

Darüber hinaus konnte die Performance durch das Verwenden des Schalters „-Xbatch“ noch weiter gesteigert werden. Er sorgt dafür, dass der Java-Binärcode niemals interpretiert wird, sondern jede Zeile vor der Ausführung vollständig in plattformspezifische Anweisungen kompiliert wird. Ohne diesen Schalter würde das Kompilieren im Hintergrund stattfinden, während im Vordergrund der Binärcode solange interpretiert wird, bis das Kompilieren abgeschlossen ist.

Der resultierende Aufruf von CamCon unter Verwendung der oben aufgeführten Schalter sieht wie folgt aus:

```
java -Xbatch -Xincgc -Xnoclassgc camcon.CamConApp
```

6. Zusammenfassung und Ausblick

Abschließend werden in diesem Kapitel die Diplomarbeit zusammengefasst, das Ziel bewertet und ein Ausblick auf mögliche Weiterentwicklungen gegeben.

6.1. Zusammenfassung

Das Ziel dieser Arbeit war es, ein System zu entwickeln, welches die Steuerung von Programmen mit Hilfe visueller Gesten ermöglicht. Dieses Ziel wurde mit der vorliegenden Arbeit erreicht.

Dazu wurde eingangs eine Szenario definiert, welches die Steuerung einer Bildschirmpräsentation mit Hilfe von visuellen Gesten vorsah. Ferner wurden Rahmenbedingungen festgelegt, welche sicherstellen sollten, dass die zu erstellende Software grundlegende Anforderungen erfüllt, und um die Komplexität des Projektes zu kontrollieren.

Anschließend wurden auf Basis des Szenarios und der Rahmenbedingungen eine Analyse und das Design erstellt.

Das Design wurde realisiert und es wurde der Prototyp „***CamCon***“ erstellt.

6.2. Fazit

Positive Aspekte

Mit dieser Arbeit konnte gezeigt werden, dass es mit relativ einfachen und kostengünstigen Mitteln möglich ist, Programme durch visuelle Gesten zu bedienen.

Das erstellte System erlaubt durch seine Plattformunabhängigkeit, dass es unter verschiedenen Betriebssystemen zum Einsatz kommen kann. Es müssen nur ein entsprechender Java-Interpreter und eine plattformabhängige Instrumentierungskomponente (z. B. eine Skriptsprache) vorhanden sein.

Seine offene Struktur erlaubt die Erweiterung um eigene Kommandos, indem sie als entsprechende Unterklassen von „AbstractCommand“ in Java implementiert werden. Dabei sind die Kommandos nicht auf die Nutzung von Java allein beschränkt. Sie können vielmehr mittels des Java Native Interfaces (JNI) beliebigen systemspezifischen Programmcode in C/C++ ausführen.

Negative Aspekte

Wenn auf dem Rechner sehr rechenintensive Prozesse ausgeführt werden kommt die Gestenerkennung fast zum Erliegen. Dieser Umstand liegt allerdings in der Natur der Sache begründet, da in solch einem Fall nicht genügend Rechenleistung für das Verarbeiten der Bilder zur Verfügung steht. Umgehen lässt sich dies nur dadurch, dass man dem Prozess von CamCon eine höhere Priorität zuweist.

Da das System ausschließlich auf der Erkennung und Verfolgung von Farben basiert ist es relativ anfällig gegenüber wechselnden Beleuchtungsverhältnissen. Hinzu kommt dabei noch, dass der automatische Weißabgleich der eingesetzten Kamera mitunter sehr heftig reagiert hat und dadurch die Erkennung negativ beeinflusste.

Im Augenblick ist es am besten, wenn der automatische Weißabgleich vor dem Einsatz von CamCon über den systemspezifischen Konfigurationsdialog des Kameratreibers deaktiviert wird. Leider ist dies softwareseitig mit dem verwendeten Java Media Framework nicht möglich, da es sehr abstrakt gehalten ist und keine entsprechende Konfigurationsmöglichkeit vorsieht.

Ein weiterer Punkt ist, dass eine durchschnittliche Webcam zwar den Vorteil hat preisgünstig zu sein, jedoch leider auch den Nachteil der relativ schlechten Bildqualität aufgrund des Rauschens. Letzteres führt dazu, dass selbst ein stillstehendes Objekt nicht immer gleichmäßig erkannt wird, sondern vielmehr zu „wabern“ scheint. Als Folge davon, ist der ermittelte Schwerpunkt besagter Fläche mitunter recht instabil.

Durch den Einsatz des Medians bei der Schwerpunktfindung wird zwar ein Großteil der Instabilität beseitigt, jedoch bleibt ein Restflackern bestehen. Seitens der Anwendung kann dieses momentan nur durch eine großzügigere Farbtoleranz bei der Markererkennung und / oder eine höhere Toleranz bei der Gestenerkennung ausgeglichen werden. Als einfache und pragmatische Lösung bietet sich jedoch auch der Einsatz einer qualitativ hochwertigeren Webcam aus dem „gehobeneren“ Preisumfeld an.

6.3. Ausblick

Das erstellte System funktioniert zwar schon recht gut, es gibt allerdings noch einen relativ breiten Spielraum für Verbesserungen und Erweiterungen. Auf einige dieser Punkte wird im Folgenden eingegangen.

Verbesserungen

- Die Auswertung und Verarbeitung von Bildern ist in Java leider nur bedingt performat umzusetzen. Es wird daher für weiterführende Entwicklungen empfohlen, externe Bibliotheken mittels des Java Native Interfaces einzubinden.

  Eine mögliche Bibliothek wäre zum Beispiel die freie „Open Source Computer Vision Library“ [OpenCV] von Intel. Sie ist für Windows und Linux erhältlich und würde, aufgrund ihrer Verfügbarkeit für mehrere Plattformen, gut in das Konzept von CamCom passen.

  Sie wurde im Rahmen dieser Arbeit noch nicht verwendet, weil ich leider zu spät von ihr erfahren habe und somit keine Zeit mehr für eine entsprechende Einarbeitung zur Verfügung stand.

- Die grafische Darstellung definierter Gesten muss erweitert werden, so dass die jeweilige Geste in Form von einer Animation gezeichnet wird. Erst dadurch wird die Darstellung von Gesten, wie zum Beispiel „Links-Rechts-Links“, wirklich sinnvoll. Bisher erscheint eine solche Geste nur als eine einzige Linie, da ihre Bewegungen deckungsgleich sind.

- Für den Einsatz in Umgebungen mit Tageslicht müsste ein Filter für den Ausgleich des automatischen Weißabgleiches der Kamera erstellt werden um die Störsicherheit zu verbessern.

- Das Ergebnis der Bildverarbeitung hängt aufgrund der reinen Verfolgung einer Farbe wesentlich von den Einsatzbedingungen ab. Es wäre daher wünschenswert, wenn weitere Eigenschaften des verfolgten Objektes zur Erkennung herangezogen werden würden.

- Den wirklich größten Schritt würde der Wechsel von einem künstlichen Objekt hin zu der Hand als natürlichem darstellen. Dazu ist das bisherige optische Erkennungssystem jedoch komplett zu überarbeiten. Das wirklich zuverlässige Erkennen der Hand kann nicht anhand der Farbe allein gelingen, da deren Schwerpunkt durch die Sichtung anderer Körperteile stark abgelenkt wird.

Erweiterungen

- Das Feedback für den Benutzer beschränkt sich bisher hauptsächlich auf die Ausgabe von Tönen bei Gestenanfang und -ende. Es wäre besser die Art des Feedbacks zukünftig konfigurierbar zu halten. Erreicht werden könnte dies durch eine offene Struktur, wie sie bei den Kommandos verwendet wird. Dazu ist das konkrete Feedback in einer eigenen Klasse bzw. in eigenen Klassen zu kapseln und eine entsprechende Auswahlmöglichkeit im User Interface sowie der Konfigurationsdatei vorzusehen.

  Auf diese Weise wäre dann eine Anpassung an die Gegebenheiten des Einsatzgebietes möglich, indem zum Beispiel an Stelle von Tönen vielleicht optische Signale auf einem externen Gerät ausgegeben werden.

- Die Erkennung von Gesten könnte von der bisherigen Richtungserkennung auf eine Erkennung der „visuell gezeichneten“ Form umgestellt werden. Selbige müsste zwar aufwendiger mittels Mustererkennung ausgewertet werden, würde allerdings wesentlich feinere Gesten ermöglichen.

- Das System könnte um eine Komponente erweitert werden, welche die Maus emuliert. Die Bedienbarkeit von Programmen bzw. der grafischen Oberfläche des Betriebssystems würde somit noch umfangreicher und komfortabler möglich. Dazu ist es jedoch notwendig das System in zwei Betriebsmodi aufzuteilen. Einen zum Betreiben als reine Gestenerkennung und einen zur reinen Maus-Emulation.

  Um allerdings eine Maus vernünftig emulieren zu können ist es erforderlich, dass das optische Erkennungssystem überarbeitet wird. Es muss in die Lage versetzt werden mehrere Farben bzw. Marker zu verfolgen, so dass eine Repräsentation der Maustasten möglich wird. Der weitere Aufwand für die notwendigen Änderungen beschränkt sich dann auf das Erstellen des Emulators und einer entsprechenden Anpassung der Klasse „CamConEngine“, sowie der Benutzeroberfläche.

Es gibt also noch viel zu tun, doch leider ist im Augenblick keine Zeit mehr dafür...

Abschließend bleibt mir daher nur noch zu sagen, dass das Erstellen dieser Arbeit eine sehr lehrreiche Erfahrung für mich dargestellt hat und ich hoffe, dass andere auch einen Nutzen aus ihr ziehen können.

Literaturverzeichnis

[AutoIt] AUTOIT: *Freeware BASIC-like scripting language for automating the Windows GUI*. – URL http://www.autoitscript.com/autoit3. – Zugriffsdatum: 23.06.2005

[Bolt 1980] BOLT, R. A.: 'Put-That-There': Voice and gesture at the graphics interface. In: *ACM SIGGRAPH Computer Graphics* 14 (1980), Nr. 3, S. 262–270

[Borland] BORLAND: *Borland Software Corporation*. – URL http://www.borland.com

[Disappearing Computer Initiative] DISAPPEARING COMPUTER INITIATIVE: *The Disappearing Computer Initiative*. – URL http://www.disappearing-computer.net. – Zugriffsdatum: 08.05.2005

[Gamma u. a. 2001] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster. Elemente Wiederverwendbarer objektorientierter Software*. Addison-Wesley, 2001. – ISBN 3-8273-1862-9

[Garcia und Morentin] GARCIA, Enrique ; MORENTIN, Laura: *Mouse WebCam*. – URL http://dftuz.unizar.es/~rivero/alumnos/vmouse.html. – Zugriffsdatum: 18.06.2005

[GhostView] GHOSTVIEW, GhostScript: *An interpreter for the PostScript language and for PDF*. – URL http://www.cs.wisc.edu/~ghost/. – Zugriffsdatum: 24.07.2005

[Gierling 2001] GIERLING, Rolf: *Farbmanagement*. Bonn : mitp-Verlag, 2001. – ISBN 3-8266-0679-5

[Gorodnichy und Roth] GORODNICHY, Dmitry O. ; ROTH, Gerhard: *Affordable yet robust and precise face tracking using USB cameras with application to designing handsfree user interfaces*. – URL http://www.cv.iit.nrc.ca/research/Nouse. – Zugriffsdatum: 18.06.2005

[JDOM-Project] JDOM-PROJECT: *XML-API for Java*. – URL http://www.jdom.org. – Zugriffsdatum: 19.07.2005

[Jensen] JENSEN, Björn: *Eine LaTeX-Vorlage zur Erstellung wissenschaftlicher Arbeiten an der Hochschule für Angewandte Wissenschaften Hamburg* . – URL http://www.mirou.de/downloads/latex_vorlage.zip. – Zugriffsdatum: 13.08.2005

[Lenman u. a. 2002] LENMAN, Sören ; BRETZNER, Lars ; THURESSON, Björn: *Computer Vision Based Hand Gesture Interfaces for Human-Computer Interaction*. Juni 2002. – URL http://cid.nada.kth.se/pdf/CID-172.pdf. – Zugriffsdatum: 17.06.2005

[Logitech] LOGITECH: *Logitech*. – URL http://www.logitech.com

[Microsoft] MICROSOFT: *Microsoft Corporation*. – URL http://www.mirosoft.com

[Mono] MONO PROJEKT: *dotNet auf Linux*. – URL http://www.mono-project.com/Main_Page/

[Nielsen 1993] NIELSEN, Jakob: Noncommand User Interfaces. In: *Communications of the ACM* 36 (1993), April, Nr. 4, S. 83–99. – URL http://www.useit.com/papers/noncommand.html. – Zugriffsdatum: 12.06.2005

[OpenCV] INTEL CORPORATION: *Open Source Computer Vision Library*. – URL http://www.intel.com/research/mrl/research/opencv/. – Zugriffsdatum: 20.07.2005

[Opera] OPERA SOFTWARE: *Web-Browser Opera*. – URL http://www.opera.com/. – Zugriffsdatum: 11.05.2004

[OptiMoz] OPTIMOZ PROJECT: *Mouse gestures for Mozilla*. – URL http://optimoz.mozdev.org/. – Zugriffsdatum: 24.06.2005

[Piekarski und Thomas] PIEKARSKI, Wayne ; THOMAS, Bruce H.: *Thumbs Up: Integrated Command and Pointer Interactions for Mobile Outdoor AR Systems*. – URL http://www.acrc.unisa.edu.au/print/people/bht/piekarski-hcii-2003.pdf. – Zugriffsdatum: 18.06.2005

[Revout 2003] REVOUT, Ilia: *Design und Realisierung eines Frameworks für Bildverarbeitung*, Hochschule für Angewandte Wissenschaften Hamburg, Diplomarbeit, 2003. – URL http://users.informatik.haw-hamburg.de/~kvl/revout/diplomarbeit.pdf. – Zugriffsdatum: 11.06.2005

[Russell u. a. 2005] RUSSELL, Daniel M. ; STREITZ, Norbert A. ; WINOGRAD, Terry: Building Disappering Computers. In: *Communications of the ACM* 48 (2005), März, Nr. 3, S. 42–48

[Smith u. a.] SMITH, Ross ; PIEKARSKI, Wayne ; WIGLEY, Grant: *Hand Tracking For Low Powered Mobile AR User Interfaces*. – URL http://www.acrc.unisa.edu.au/print/people/bht/piekarski-hcii-2003.pdf. – Zugriffsdatum: 18.06.2005

[Streitz und Nixon 2005] STREITZ, Norbert ; NIXON, Paddy: The Disappearing Computer. In: *Communications of the ACM* 48 (2005), März, Nr. 3, S. 32–35

[StrokeIt] JEFF DOOZAN, TCBNETWORKS: *Mouse Gestures for Microsoft Windows*. – URL http://www.tcbmi.com/strokeit/. – Zugriffsdatum: 18.06.2005

[Sun] SUN MICROSYSTEMS INC.: *Sun Microsystems*. – URL http://www.sun.com

[Sun a] SUN MICROSYSTEMS INC.: *Java 2 Platform, Standard Edition (J2SE)*. – URL http://java.sun.com/j2se/. – Zugriffsdatum: 12.06.2005

[Sun b] SUN MICROSYSTEMS INC.: *Java Media Framework*. – URL http://java.sun.com/products/java-media/jmf/. – Zugriffsdatum: 12.06.2005

[Sun c] SUN MICROSYSTEMS INC.: *Image Capture From Webcams using the Java Media Framework API*. – URL http://java.sun.com/developer/technicalArticles/Media/mediaframework/. – Zugriffsdatum: 12.06.2005

[TEXnicCenter] TEXNICCENTER: *open-source LATEX-shell*. – URL http://www.toolscenter.org. – Zugriffsdatum: 24.07.2005

[Weiser 1991] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* (1991), September, S. 94–104

A. Listings

A.1. AutoIt-Skripte zur Steuerung von Microsoft PowerPoint

„Nächste Folie“

```
; This script tries to activate a window whose title
; begins with "PowerPoint-". If such a window could
; be activated it sends the "page up" keystroke to it.

WinActivate("PowerPoint-")
If WinActive("PowerPoint-") Then
    Send("{PGUP}")
EndIf
```

„Vorherige Folie“

```
; This script tries to activate a window whose title
; begins with "PowerPoint-". If such a window could
; be activated it sends the "page down" keystroke to it.

WinActivate("PowerPoint-")
If WinActive("PowerPoint-") Then
    Send("{PGDN}")
EndIf
```

A.2. CamCon-Konfigurationsdatei

```
<?xml version="1.0" encoding="UTF-8"?>
<camcon_config>
  <general>
    <version value="1.0" />
    <beeponrecognition value="true" />
    <beepongesturestart value="true" />
    <timebetweengestures value="1000" />
  </general>

  <recognitionsystem>
    <timeforgesturestartandend value="1500" />
    <lostmarkertimeout value="2000" />
    <maxstandingflickering value="2" />
    <maxmoveingflickering value="3" />
  </recognitionsystem>

  <video>
    <camera name="vfw:Microsoft WDM Image Capture (Win32):0"
    resx="320" resy="240" />
    <markercolor value="d95f72" threshold="0.14"
    ignorebrightness="false" />
    <blocksize width="8" height="8" />
    <minpixelsforvalidblock value="4" />
  </video>

  <commands>
    <commandtemplates>
      <commandtemplate name="Run a program" class=
     "camcon.engine.commandsystem.commands.CmdRunExternalProgram"/>
    </commandtemplates>

    <concretecommands>
      <command name="PowerPoint-Prev" commandtemplate="Run a program">
        <param value=".\autoit-v3\AutoIt3.exe" />
        <param value=".\autoit_scripts\PowerPoint-PGUP.au3" />
      </command>
      <command name="PowerPoint-Next" commandtemplate="Run a program">
        <param value=".\autoit-v3\AutoIt3.exe" />
        <param value=".\autoit_scripts\PowerPoint-PGDN.au3" />
      </command>
    </concretecommands>
  </commands>
```

```
  <gestures>
    <gesture name="Right" movements="R" />
    <gesture name="Left" movements="L" />
  </gestures>

  <mappings>
    <mapping gesture="Right" command="PowerPoint-Prev" />
    <mapping gesture="Left" command="PowerPoint-Next" />
  </mappings>
</camcon_config>
```

B. Inhalt der CD-ROM

Die CD-ROM zu dieser Arbeit enthält folgende Verzeichnisstruktur:

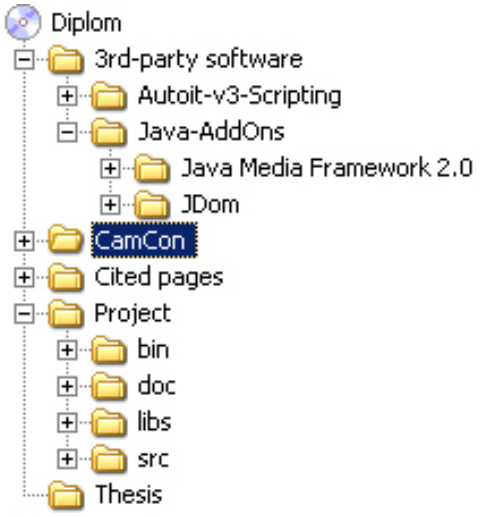


Abbildung B.1.: Inhalt der CD-ROM

- „3-rd-party software“ - enthält fertige Programme und Bibliotheken anderer Autoren, welche vollständig oder teilweise zur Erstellung von CamCon verwendet wurden.
- „CamCon“ - enthält das fertige Produkt dieser Diplomarbeit. Die Anwendung zur Steuerung von Programmen anhand visueller Gesten - ***CamCon***.
- „Cited pages“ - enthält Kopien der verwendeten Internetquellen, soweit selbige angefertigt werden konnten.
- „Project“ - enthält das Borland JBuilder 8-Projekt der CamCon-Anwendung, den Quellcode und die dazugehörige Dokumentation im Javadoc-Format.
- „Thesis“ - enthält diese Diplomarbeit im pdf-Format.

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, 30. August 2005	
Ort, Datum	Unterschrift