UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

AGAS
Arbeitsgruppe Aktives Sehen

Demonstrator für Interaktion durch Augenbewegung I

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von

Sascha Lange

Betreuer:	Dipl.-Inf. Detlev Droege, Institut für Computervisualistik, Fachbereich Informatik
Erstgutachter:	Dipl.-Inf. Detlev Droege, Institut für Computervisualistik, Fachbereich Informatik
Zweitgutachter:	

Koblenz, im Juni 2007

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Richtlinien der Arbeitsgruppe für Studien- und Diplomarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden. ja ☐ nein ☐

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu. ja ☐ nein ☐

Koblenz, den ..... Unterschrift

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Im Jahr 2004 entstand in der Diplomarbeit [Fri05] das System „GoldenGaze“, welches die aktuelle Blickposition auf dem Bildschirm ausgibt. Dieser gaze tracker („Blickrichtungsverfolger“) kann ausschließlich mit Low-Cost-Equipment betrieben werden und stellt damit eine günstige Alternative zu den teuren kommerziellen Systemen dar. Die Diplomarbeit ist Bestandteil des EU-geförderten Programms COGAIN (Communication by Gaze Interaction), welches sich zum Ziel gesetzt hat, Menschen mit körperlicher Behinderung eine Kommunikation mittels Augenbewegung zu ermöglichen. „GoldenGaze“ wurde komplett unter Linux mit PUMA (Programmierumgebung für die Musteranalyse) realisiert.

2007 folgt voraussichtlich ein weiterer gaze tracker, der Bestandteil der Diplomarbeit [Gei07] ist. Dieser wird komplett unter Windows realisiert und befand sich bei Fertigstellung dieser Ausarbeitung noch in Arbeit.

Beide Systeme nutzen eine preisgünstige Kamera, die sich dank eines Exview HAD-Chips durch eine besonders hohe Lichtempfindlichkeit auszeichnet. Infrarot-Dioden sorgen dafür, dass ein Glanzlicht auf der Hornhaut entsteht, welches anschließend im aufgenommenen Bild gefunden wird, um aus der relativen Position des Glanzlichts zum Pupillenmit-

telpunkt die Blickrichtung zu berechnen.

Trotz eines ähnlichen Ansatzes, unterscheiden sich beide gaze tracker in der Bedienung der Systeme deutlich voneinander. Beide Programme sind mit einer eigenen Kalibrierung versehen, die sich aber durch unterschiedliche Funktionen auszeichnen. Das „GoldenGaze“ berechnet in der Kalibrierung die Positionen der fünf virtuellen Tasten zur Steuerung des Texteingabeprogramms „ERIC“. Eine kontinuierliche Ausgabe der aktuellen Blickrichtung auf dem Bildschirm war im Rahmen der Diplomarbeit nicht vorgesehen. Dagegen soll der gaze tracker von Thorsten Geier ein kontinuierliches Feedback geben, um z.B. den Mauszeiger zu bewegen.

Hier soll diese Studienarbeit ansetzen. Damit auf Tagungen oder Fachmessen die Probanden vor einem Test der Systeme nicht erst in die Funktionsweise eingewiesen werden müssen, sondern je nach belieben eines dieser Systeme gestartet und ohne fachmännische Einleitung betrieben werden kann, bedarf es einer einheitlichen Benutzeroberfläche, über die sowohl die Kalibrierung als auch einige Programme zur Demonstration direkt gestartet werden können.

1.2 Ziel der Studienarbeit

Ziel dieser Arbeit, ist die Visualisierung von berechneten Blickpositionen auf dem Bildschirm und eine Interaktion durch Augenbewegung, die die berechneten Koordinaten als Grundlage verwenden.

Da die berechneten Koordinaten zunächst sehr ungenau sind, weil sie entweder unkalibriert sind oder zur Kalibrierung unterschiedliche Methoden angewendet wurden, ist es erforderlich eine einheitliche Kalibrierung vorzunehmen, mit dem Ziel, die Resultate unterschiedlicher Systeme später vergleichen zu können. Neben der Kalibrierung soll ein Programm implementiert werden, welches einen einheitlichen Vergleich unterschiedlicher gaze tracking Systeme ermöglicht. Dieses soll mehrere Level umfassen, um unterschiedliche Qualitätsmerkmale zu erfassen:

- Genauigkeit der Blickposition bei ruhendem Auge

- Genauigkeit der Blickposition bei der Objektverfolgung
- Genauigkeit der dwell time

Da der Umgang mit gaze tracking Systemen auch spielerisch gestaltet werden soll, wurde ein kleines Spiel implementiert, welches ebenfalls komplett mit den Augen bedient werden kann. Auf die Problematik von Spielen, die ohne Tastendruck im herkömmlichen Sinn auskommen müssen, soll in dieser Arbeit separat eingegangen werden.

Das Resultat dieser Studienarbeit soll ein benutzerfreundliches Programm sein, das oben genannte Funktionalitäten erfüllt und ohne großen Aufwand das Einbinden eines beliebigen gaze trackers ermöglicht.

1.3 Aufbau der Arbeit

Im Anschluss dieser Einleitung folgt Kapitel 2, in dem zunächst der Begriff gaze tracking näher erläutert und die Funktionsweise grob umrissen wird, um schließlich auf die Anwendungsgebiete zu verweisen. Außerdem sollen einige Anwendungen, die mit den Augen gesteuert werden können, beschrieben werden. In Kapitel 3 wird dann die Umsetzung der einzelnen Schritte beschrieben, die zur Erstellung des Programms nötig waren. Ein besonderer Schwerpunkt in diesem Kapitel liegt auf dem Thema der Kalibrierung. Hier sollen mehrere Lösungsansätze vorgestellt und anschließend die Implementierung erklärt werden. Die Ergebnisse der Evaluierungsphase werden dann in Kapitel 4 beschrieben und ausgewertet. In Kapitel 5 werden die Ergebnisse dieser Arbeit nochmal zusammengefasst und um einen kurzen Ausblick erweitert.

Kapitel 2

Stand der Wissenschaft

2.1 Gaze tracking allgemein

Gaze tracking bedeutet soviel wie „Blickrichtungsverfolgung“. Systeme, die ein gaze tracking ermöglichen, verfolgen das Ziel, die aktuelle Blickposition einer Person zu berechnen und diese entweder auf dem Bildschirm auszugeben oder bestimmte Aktionen auszulösen.

2.1.1 Funktionsweise

Um die Blickrichtung zu berechnen, ist es zunächst erforderlich, die Augen der betroffenen Person mit einer Kamera aufzunehmen. In den meisten Fällen wird der Bildausschnitt nicht nur auf den Bereich der Augen beschränkt, sondern der ganze Kopf mit etwas Spielraum zu allen Seiten erfasst. Dadurch soll gewährleistet werden, dass das System auch noch bei leichten Kopfbewegungen funktioniert. Außerdem hat die Erfahrung gezeigt, dass es durchaus hilfreich ist, zusätzliche Merkmale im Gesicht, wie z. B. die Nase (oder besser: die Nasenlöcher), mit in die Berechnung einfließen zu lassen. So lässt sich allein durch die Position der Augen und der Nase relativ leicht die Kopfstellung bestimmen. Die Bildqualität der verwendeten Kamera spielt beim gaze tracking eine große Rolle. Da zur Bestimmung der Blickrichtung häufig die Position der Pupille entscheidend ist, muss diese auch im Bild eindeutig lokalisiert werden. Bei einer zu niedrigen Auflösung wäre

die Pupille bei einem durchschnittlichen Abstand von 70 cm nur noch wenige Pixel breit und hoch, was eine exakte Lageposition erschwert.

Die sogenannte Hornhaut-Reflektionsmethode ist dabei nur einer von vielen Ansätzen. [Fri05] nennt in seiner Diplomarbeit folgende Ansätze zur Bestimmung der Blickrichtung:

- **Bestimmung der Blickrichtung mit der Hornhaut-Reflektionsmethode:** Infrarotstrahlen sorgen für eine ausgeleuchtete Szene, wobei das Infrarotlicht für den Menschen nicht sichtbar und somit auch nicht störend ist. Die Kamera, die in den meisten Fällen infrarot-empfindlich ist, kann dann das Bild mit dem Glanzlicht auf der Hornhaut aufnehmen. Zur Bestimmung der Blickrichtung wird nun die relative Position des Glanzlichtes zur Pupillenmitte berechnet. Eine Methode, die auch mit Kameras geringerer Qualität funktioniert, da die Glanzlichter als hellste Punkte im Bild, eine Lokalisation der Augen erleichtern.

- **Bestimmung der Blickrichtung anhand der Purkinje-Bilder:** Bei der Beleuchtung mit Infrarotlicht treten auf der Hornhaut insgesamt vier Reflektionen auf. Aus deren relativen Positionen, lässt sich die Blickrichtung bestimmen. Allerdings sind diese Reflektionen zu schwach, um sie aus größerer Entfernung identifizieren zu können.

- **Bestimmung der Blickrichtung durch aussehensbasierte Methoden:** Man vergleicht das aktuelle Augenbild mit mehreren Referenzbildern und leitet daraus die Blickrichtung ab. Dieses Verfahren ist vermutlich nicht sehr robust gegenüber Positions- und Helligkeitsschwankungen.

- **Bestimmung der Blickrichtung durch neuronale Netze:** Nach Eingabe eines Bildes mit dem aufgenommenen Auge, soll die Ausgabe des neuronalen Netzes die Bildschirmkoordinaten der aktuellen Blickposition liefern. Das Verfahren ist sehr ineffizient und benötigt zu Beginn ca. 2000 Trainingsbilder. Insgesamt ist dieses Verfahren sehr undurchsichtig, was ein typisches Problem bei neuronalen Netzen darstellt.

- **Bestimmung der Blickrichtung als Ausrichtung des Gesichts:** Es wird nicht die Blickrichtung in den Augen bestimmt, sondern die Ausrichtung des Kopfes als Maß

für die aktuelle Blickrichtung herangezogen. Allerdings müsste jede Blickrichtungsänderung mit einer entsprechenden Kopfbewegung vollzogen werden. Problematisch ist ebenso die Wahl des Tastendrucks; wenn die Augen geschlossen sind könnte trotzdem jederzeit eine Aktion ausgelöst werden.

- **Bestimmung der Blickrichtung durch „limbus tracking“:** Limbus bezeichnet die Übergangszone zwischen Cornea (Hornhaut) und Sclera (Lederhaut), die durch eine deutliche Abgrenzung der Iris zur Sclera sichtbar ist. Dadurch kann die Iris leichter lokalisiert und ihre Ausrichtung bestimmt werden. Problematisch bei diesem Verfahren ist die Lokalisation der Augenränder, um daraus auf die relative Position der Iris im Auge zu schießen. Hinzu kommt, dass, für eine erfolgreiche Bestimmung der Blickrichtung, der Kopf stets unbewegt bleiben muss.

- **Bestimmung der Blickrichtung durch „pupil tracking“:** Wie „limbus tracking“, nur dass diesmal die Pupille anstatt die Iris gesucht wird. Ein Vorteil dabei ist, dass die Pupille im Gegensatz zur Iris nicht teilweise durch die Augenlieder verdeckt ist, ein Nachteil allerdings, dass sie sich nur wenig von der Iris unterscheidet.

Kommerzielle Systeme erzielen mittlerweile eine pixelgenaue Bestimmung der Blickrichtung, was eine gezielte Steuerung einer Benutzeroberfläche ermöglicht. Eine möglichst exakte Berechnung ist besonders deswegen erstrebenswert, da das System dann nicht mehr abhängig von spezieller Software ist, die man mit dem gaze tracker verwenden kann (z.B. mit sehr großen virtuellen Tasten), sondern jedes beliebige Programm, welches mit einer grafischen Oberfläche ausgestattet ist, bedient werden kann. Der aus der Diplomarbeit entstandene gaze tracker GoldenGaze konnte diese pixelgenaue Bestimmung der Blickrichtung nicht erzielen, da das System von vornherein mit stark eingeschränkten Hardwarevoraussetzungen auskommen musste. Der Genauigkeitstest aus [Fri05] ergab einen durchschnittlichen Fehler von weniger als 5° und reicht damit völlig aus, um die für „ERIC“ benötigten fünf Tasten zu bedienen.

Neben den im Rahmen von COGAIN entstandenen gaze tracking Systemen, die alle zu einem erschwinglichen Preis nachzubauen sind, gibt es einige kommerzielle Systeme. Einige Beispiele sind „MyTobii“ von Tobii Technologies, EyegazeSystems von LC Technologies INC und „faceLAB“ von Seeing Machines. Der Preis dieser Produkte liegt bei ca.

Bild 2.1: MyTobii P10 (Quelle [TTA]).

20000 Euro.

2.1.2 Anwendungsgebiete

Eine große Rolle spielen gaze tracker für die **Zugänglichkeit** zu alltäglicher Kommunikation von Menschen mit körperlicher Behinderung. Für einige dieser Fälle käme neben dem gaze tracking auch eine Spracherkennung zur Interaktion in Frage. Aber in vielen Fällen ist das Sprachzentrum ebenfalls betroffen, so dass kontrollierte Aktionen lediglich über die Augen möglich sind. Dass die Zugänglichkeit zu diesen Medien für die Betroffenen

Bild 2.2: Der Einsatz von gaze trackern bei der Kommunikation von körperbehinderten Menschen spielt eine große Rolle (Quelle [COG]).

erheblich zur Lebensqualität beitragen kann, liegt auf der Hand. So können sie auf diese Weise beliebige Programme auf einem Computer bedienen und darüber mit ihrer Umwelt in Kontakt treten.

Darüber hinaus gibt es aber noch eine Vielzahl anderer Möglichkeiten, in denen der Einsatz von gaze tracking Systemen sinnvoll sein kann.

Die automatische Erkennung der Blickrichtung ermöglicht eine ganz neue Form von **Benutzertests**. So können Firmen ihre Softwareprodukte und Internetseiten effektiv, schnell und objektiv evaluieren und für ein größeres Maß an Ergonomie sorgen [TTA]. Besonders im Internet spielt das Design und der Aufbau von Webseiten eine große Rolle. Das Bild 2.3 zeigt auf einer Heatmap (Hitzekarte), welche Bereiche der Trefferliste einer Suchmaschine besonders aufmerksam betrachtet werden.

Auch in der **Erforschung** zur Mensch-Maschine Interaktion und im Bereich der Kogniti-

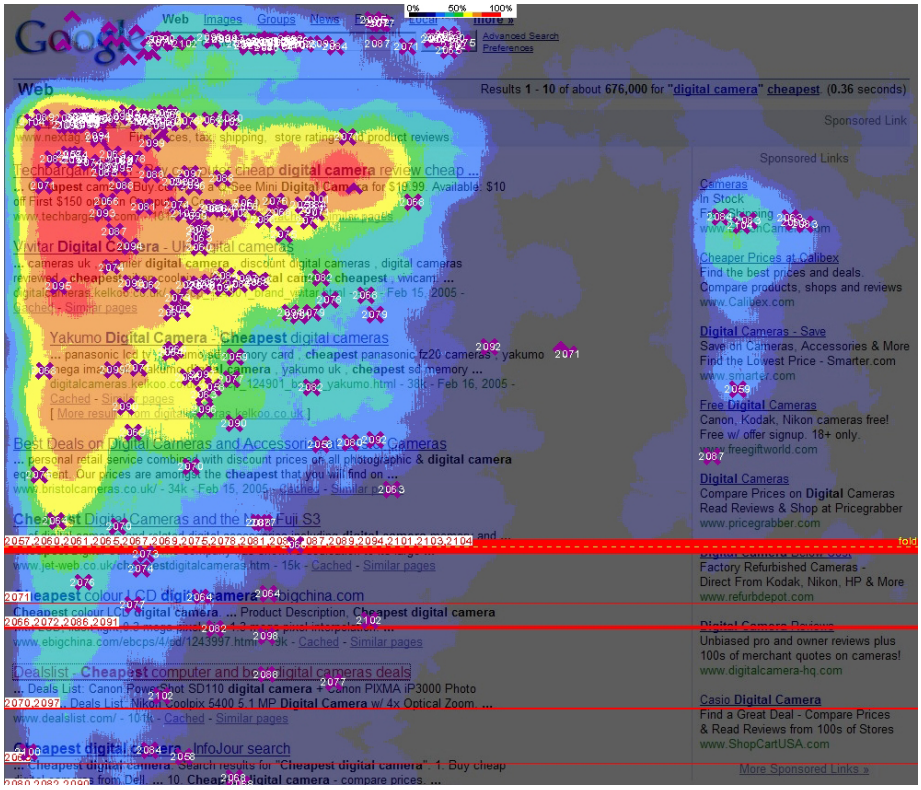

Bild 2.3: Diese Heatmap zeigt, um wieviel größer die Aufmerksamkeit der ersten Treffer im Gegenstz zu den weiter unten aufgelisteten Treffern ist (Quelle [Len]).

onspsychologie und Neurophysiologie finden bereits gaze tracking Systeme Verwendung [TTA].

Darüber hinaus ist es möglich, **intelligente Anwendungen** zu entwickeln [TTA]. Da das System ständig über unsere Blickposition informiert ist, kann das laufende Programm sofort auf den Benutzer reagieren. So kann ein Dokument z.B. selber hoch- und runterscrollen, je nachdem, wohin jemand gerade sieht. Auch könnte das Programm auf unterschiedliche Gemütszustände reagieren. Wenn sich aus den Blickfolgen und der Geschwindigkeit von Blickrichtungsänderungen ableiten lässt, ob der Benutzer abgelenkt oder müde ist, könnte das Programm seine Geschwindigkeit oder Komplexität verändern, um die Aufmerksamkeit zu erhöhen.

Ein anderer Bereich, in dem der Einsatz von gaze tracking erforscht wird, ist die **Diagnose**. In diesem Bereich gäbe es vielfältige Möglichkeiten. So könnte man aufgrund von Blickrichtungsmuster eventuell auf Autismus oder Schizophrenie schließen oder die geistige Entwicklung von Säuglingen bewerten. Auch der Einfluss von Drogen könnte in Echtzeit überprüft werden [TTA].

Außerdem wäre ein Einsatz von gaze tracking auch bei handwerklichen Tätigkeiten z.B. bei der Montage denkbar. Die Augen könnten eine dritte **„helfende Hand“** darstellen, mit der Geräte parallel bedient werden könnten.

2.2 Existierende Anwendungen

Auch wenn kommerzielle Systeme bereits eine pixelgenaue Blickrichtungsberechnung durchführen können, macht es Sinn, sich nicht nur für gaze tracker mit einer geringeren Auflösung, Gedanken über spezielle Software zu machen.

Die Interaktion über die Augen ist doch in vielerlei Hinsicht anders als die gewohnte Steuerung mit Maus und Tastatur. Deshalb sollen in diesem Abschnitt zwei Anwendungen vorgestellt werden, die sich auf unterschiedliche Weise dieser Problematik stellen.

2.2.1 „GazeTalk“

Das Programm „GazeTalk“, welches von der „Eye Gaze Interaction Group“ an der Universität von Kopenhagen entwickelt wurde, integriert verschiedene Funktionen, auf die mit einem gaze tracker schnell zugegriffen werden können [COG]. Das Layout des Programms besteht aus mehreren großen Quadraten, die als virtuelle Tasten fungieren. Es wurde bewusst eine Oberfläche mit wenigen, aber dafür um so größere Tasten gewählt, damit GazeTalk auch von gaze trackern mit einer geringeren Genauigkeit verwendet werden können. Der Tastendruck wird bei der Verwendung eines gaze trackers durch eine dwell time („Verweildauer“) realisiert, deren Dauer beliebig eingestellt werden kann. Sobald eine virtuelle Taste fixiert wird, kann man den Fortschritt der dwell time nachvollziehen, da die schwarze Taste in der Mitte zusammenschrumpft, bis die Taste vollständig grau ist und

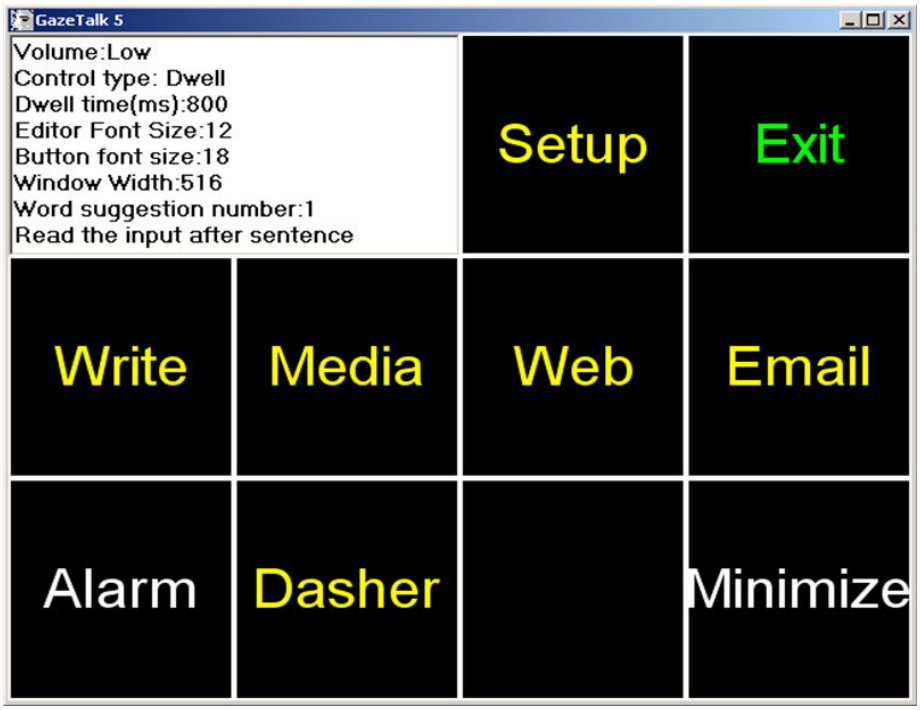

Bild 2.4: Die oberste Ebene des Programms „GazeTalk“. Von hier aus können Multimedia -Anwendungen oder Programme zum Schreiben und Lesen von Texten ausgewählt werden (Quelle [COG]).

die Taste bestätigt wird. „GazeTalk“ lässt sich aber auch ohne gaze tracker bedienen, vorausgesetzt die Person hat die Möglichkeit wenigstens eine Taste mechanisch betätigen zu können. Der dafür entwickelte „scan modus“ markiert alle Tasten nacheinander in schneller Abfolge, so dass der Benutzer lediglich eine Taste benötigt, um die aktuell markierte virtuelle Taste zu betätigen.

„GazeTalk“ ermöglicht dem Benutzer einen schnellen Zugriff auf Multimedia-Funktionen, z. B. Abspielen von Musik oder Video. Auch das Lesen von Dokumenten und Surfen im Internet ist Bestandteil des Programms. Die Möglichkeit zur Kommunikation war aber das eigentliche Ziel der Entwickler. So bietet „GazeTalk“ die Möglickeit, Texte zu verfassen. Dazu wird jeweils eine Auswahl von Buchstaben angezeigt, die durch linguistiche Methoden vorgenommen wird, so dass nur die Buchstaben angezeigt werden, die die größte

Wahrscheinlichkeit haben, nach dem vorangegangenen Buchstaben ausgewählt zu werden. Zusätzlich kann das Programm auch Wortvorschläge machen, um das Schreiben zu beschleunigen. Um eine Kommunikation nicht nur auf dem schriftlichen Weg zu ermöglichen, können alle verfassten Worte über die Soundkarte ausgegeben werde.

Das Schreiben mit Hilfe der dwell time hat aber einen Nachteil: Die Schreibgeschwindigkeit bleibt konstant. So kann nicht, wie beim Tippen mit den Fingern, eine geübte Folge von Tasten schneller eingetippt werden, als Tasten, die man seltener benutzt und etwas mehr Zeit brauchen, ehe sie betätigt werden. Doch andere Methoden zum virtuellen Tastendruck bieten keine bessere Alternative. Es wäre unvorstellbar, wenn man bei jedem Tastendruck seinen Blick vom Bildschirm ruckartig entfernen oder blinzeln müsste. Daher sind für eine angemessene Texteingabe über gaze tracker neue Wege gefordert. Einen ganz neuen Ansatz verfolgt das Programm „Dasher“, welches im anschließenden Abschnitt vorgestellt wird. „Dasher“ ist zudem auch in „GazeTalk“ integriert worden, so dass von „Dasher“ verfasste Texte in „GazeTalk“ verwendet werden können, um sie z.B. per E-Mail zu versenden.

Das Programm „GazeTalk“ steht auf der Homepage von COGAIN (www.cogain.org) kostenlos zum Download zur Verfügung.

2.2.2 „Dasher“

Wie eben schon erwähnt, stellt die Texteingabe über gaze tracker eine besondere Herausforderung dar. Eine virtuelle Adaption einer Tastatur ermöglicht zwar die Eingabe von Text, allerdings nicht besonders schnell.

Einen ganz neuen Ansatz, um auf effiziente Weise Texte zu verfassen, ohne dass die Tastatur nötig ist, verfolgt das Programm „Dasher“, welches von der Inference Group an der Universität Cambridge im Rahmen von COGAIN entwickelt wurde.

Auf der rechten Bildschirmseite erscheinen übereinander geordnete Buchstaben in Kästchen, wobei die Anordnung und Größe eines jeden Buchstabenfeldes nach den Wahrscheinlichkeiten des nächsten Buchstabens geordnet sind. Dabei enthält jedes Kästchen mit einem Zeichen wiederum alle möglichen Zeichen usw. Der Mauszeiger, der ganz nor-

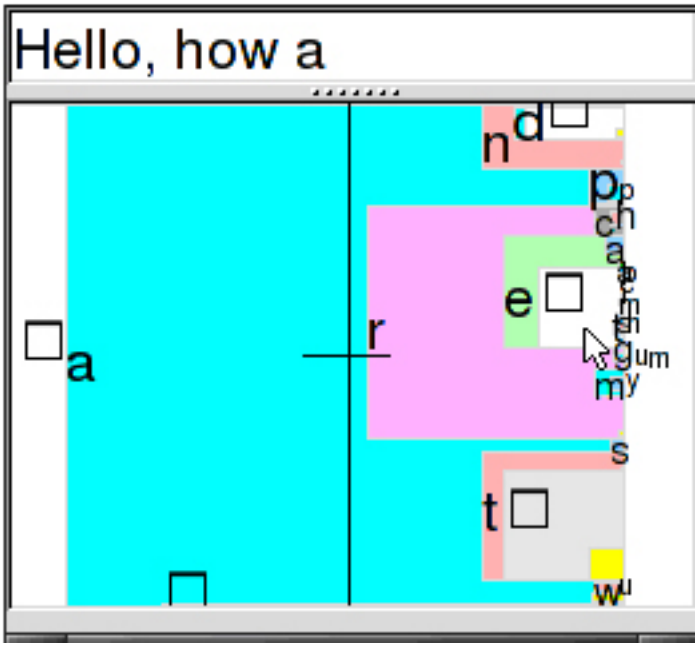

Bild 2.5: Das Programm „Dasher“, das gerade „Hello, how are you?“ tippt. Am oberen Rand sieht man die Buchstabenfolge „n“ „d“ mit anschließendem Leerzeichen, sowie am unteren Rand das „t“ mit anschließendem Leerzeichen, wodurch die im englischen häufigsten Buchstabenkombinationen nach dem Buchstaben „a“ am schnellsten getippt werden können (Quelle [COG]).

mal mit der Hand oder eben auch von einem gaze tracker bewegt werden kann, reicht zur Navigation aus.

In der horizontalen Ebene wird die Schreibgeschwindigkeit festgelegt. Dabei bedeutet die Verlagerung der Maus nach rechts das Eintippen von Buchstaben, während die Positionierung der Maus nach links ein Löschen der gewählten Buchstaben zur Folge hat. Dabei wird die Geschwindigkeit von der Entfernung zum Fenstermittelpunkt beeinflusst. In der Mitte des Fensters geschieht nichts, ermöglicht also eine Pause, um über die nächsten Worte nachzudenken.

In der vertikalen Ebene kann durch die Buchstabenfelder gescrollt werden. Dabei ist der Abstand zum Mittelpunkt wieder entscheidend für die Geschwindigkeit. Sobald das gewünschte Zeichen gefunden wurde, wird dieses bestätigt, indem die Maus in das entspre-

chende Feld bewegt wird. Weiße Felder markieren ein Leerzeichen. Auch andere Zeichen wie Punkt, Komma und Ausrufezeichen sind verfügbar. Diese befinden sich aber, sehr klein und versteckt im unteren oder oberen Bereich des jeweiligen Buchstabenfeldes, da ihr Vorkommen im Text verhältnismäßig gering ist, und das Programm nicht abschätzen kann, wann ein Satz zu ende ist.

Den Angaben der Entwickler zufolge, können geübte Benutzer 29 Wörter pro Minute schreiben, wenn zur Bewegung des Mauszeigers ein gaze tracker verwendet wird, und bis zu 39 Wörter pro Minute eingeben, bei der Verwendung mit der Maus [COG].

Das Programm „Dasher“ steht auf der Homepage von COGAIN (www.cogain.org) kostenlos zum Download zur Verfügung.

2.3 Gaze tracking in Computerspielen

Ein sehr interessanter Ansatz wäre es, gaze tracking zur Steuerung von Computerspielen zu nutzen. Besonders nahe liegend wäre der Einsatz bei den sogenannten First Person Shooter Games (FPGS). Der Spieler erlebt seine Umwelt aus seiner eigenen Perspektive, wobei der Blick mit der Maus in alle Richtungen gelenkt werden kann. Genauso wird auch auf Gegner gezielt und mit einer Maustaste attackiert. Wenn man sich vorstellt, dass mit gaze trackern ermöglicht wird, den Mauszeiger ziemlich genau zu steuern, wäre eine Übertragung denkbar einfach: Der reale Blick würde dann den virtuellen Blick steuern und sobald ein Gegner fixiert wird, könnte dieser nach der dwell time angegriffen werden.

In der Praxis ist das allerdings nicht einfach übertragbar. [Jö05] zitiert in ihrer Magister-Arbeit Shumin Zhai, einen Forscher von IBM, der sich auf Mensch-Maschine Interaktion spezialisiert hat, dass er sehr skeptisch mit der Idee von augengesteuerten Computerspielen umgegangen sei, weil „the eye is sometimes controlled by will and sometimes by extern events“.

Dies kann man sich auch daran verdeutlichen, dass wir für unsere Wahrnehmung ständig die Blickrichtung ändern müssen. Selbst wenn man ganz gezielt versucht einen Punkt zu fixieren, wird der Blick immer wieder davon abweichen und die nähere Umgebung absuchen. Wenn man sich nun vorstellt, dass ein Computerspiel auf jede Blickrichtungsände-

rung reagiert, würde dies zu einer unkontrollierbaren Folge von Aktionen führen. Angenommen das Spiel besteht aus einem Hindernis-Parkour, welcher durch gezielte Blickrichtungen durchfahren werden soll. Dann würde jedes Mal, wenn der Blick die Umgebung nach Hindernissen absucht, eine entsprechende Richtungsänderung des Fahrers erfolgen. Also entweder, man konzentriert sich einfach auf eine Richtung, in die der Fahrer fahren soll und prallt gegen das erste Hindernis, oder man sucht die Umgebung nach Hindernissen ab, um ihnen zu entgehen, löst dabei aber eine Richtungsänderung des Fahrers aus, so dass dieser dadurch gegen dieses Hindernis fährt.

Dennoch würde ein Einsatz von gaze trackern wesentlich zum Gameplay beitragen. [Jö05] betrachtet sogar die Nutzung von Gesten, Gesichtsaudrücken und der Stimme für die Interaktion in zukünftigen Spielen.

Die Frage nach der Genauigkeit von Ego-Shootern, die durch Blickrichtungen gesteuert werden, stellten sich auch [IM] im Rahmen von COGAIN. Allerdings wurde das gaze tracking lediglich auf das Zielen innerhalb des aktuellen sichtbaren Bereichs beschränkt. Sowohl die Navigation durch die Welt, als auch die Änderung der Blickrichtung und das Betätigen der Waffe wurde mit der Tastatur bzw. Maus erledigt. In der Welt, die extra für diese Arbeit konstruiert wurde, ging es nun darum ein rundes Objekt, dessen Position zufällig generiert wurde, zu lokalisieren und abzuschießen. Die Ergebnisse wurden dann mit den Ergebnissen des Spiels bei Verwendung der Tastatur und der Maus und bei Verwendung mit einem XBox360-Controller verglichen. Dabei zeigte sich, dass die Variante von Tastatur und Maus am Besten abgeschnitten hat. Die Variante mit dem gaze tracker und die mit dem XBox360-Controller schnitten dagegen in etwa gleich ab. Daraus kann man folgern, dass es durchaus Verwendungsmöglichkeiten für gaze tracker in Computerspielen geben kann.

2.4 Programmierung von Benutzeroberflächen

Ziel dieser Arbeit ist es, ein Programm mit Benutzeroberfläche (BOF) zu erstellen, welches die Resultate von gaze trackern demonstriert. Die programmiertechnischen Anforderungen dieser Arbeit sehen wie folgt aus:

- Programmiersprache muss C++ sein
- Source-Code muss plattformunabhängig sein

Es gibt zahlreiche Frameworks, die unter diesen Voraussetzungen eine Programmierung von BOF's erleichtern. Eine kleine Auswahl, entnommen aus [Geo], enthält folgende Werkzeuge:

- gtk++
- Qt
- WxWidgets

Zur Programmierung der BOF wurde Qt gewählt, da es eine sehr gute Klassendokumentation hat und sehr intuitiv zu lernen ist. Qt ist eine Klassenbibliothek, die von Trolltech entwickelt wurde. Die aktuelle Version Qt 4.2 gibt es mit einer kommerziellen Lizenz oder einer Open Source Lizenz, die der GNU General Public License (GPL) zugrunde liegt (mehr Informationen unter www.trolltech.com). Da der Demonstrator zeitgleich unter Windows und Linux entwickelt werden soll, musste eine Entwicklungsumgebung gefunden werden, die eine plattformunabhängige Arbeit ermöglicht. Daher wurde die Umgebung QDevelop gewählt. Es handelt sich um eine sehr einfache, auf die Programmierung von Qt-Applikationen zugeschnittene Umgebung, die als Konsequenz sehr übersichtlich gestaltet ist (weitere Informationen unter http://qdevelop.org/).

Kapitel 3

Eigene Umsetzung

3.1 Programm-Aufbau

Das vereinfachte Klassendiagramm in Bild 3.1 veranschaulicht den Aufbau des Demonstrators. Die Programme „Eprogat“, „Heatmap“ und „ButtonUp“ werden bei jedem Aufruf im Hauptprogramm neu erstellt. Dagegen wird „CalibrationComputation“ nur einmal instanziiert, damit sowohl die gewählte Kalibriermethode als auch die zum Kalibrieren nötigen Parameter in jedem Programm abrufbar sind. Das Programm „Calibration“ hingegen wird bei jedem Aufruf neu erstellt.

Die Klassen „Cursor“ und „Heatmap“ sind Bestandteil der parallelen Studienarbeit [Sch07]. In allen Programmen des Demonstrators wurde als Methode zum Tastendruck die Variante der dwell time gewählt. Was für die dwell time spricht und wie sie visualisiert werden kann bzw. welche Alternativen es gäbe wird ebenfalls in [Sch07] beschrieben.

3.2 Kalibrierung

Im Demonstrator wurden drei verschiedene Methoden zur Kalibrierung implementiert. Ziel der Kalibrierung ist es, die vom gaze tracker berechneten x/y-Koordinaten auf den Bildschirm zu mappen. Dazu müssen die Koordinaten eventuell erst verschoben und ska-

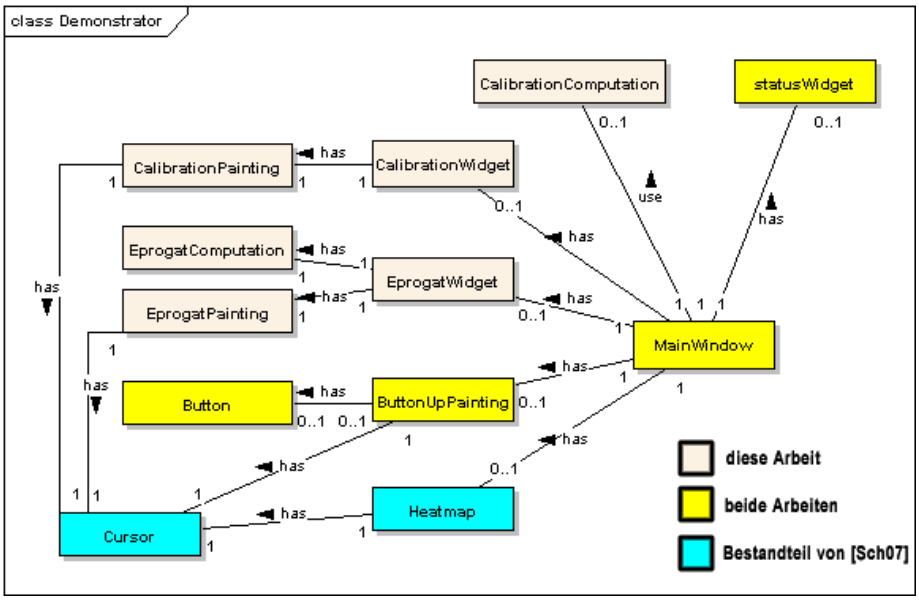

Bild 3.1: Vereinfachtes UML-Klassenmodell des Demonstartors mit farblicher Einteilung der Aufgabenverteilung

liert werden, damit berechnete Punkte mit den fixierten Punkten übereinstimmen.

Die Voraussetzungen der drei Methoden sind dabei gleich und sehen wie folgt aus:

Um Kalibrieren zu können, bedarf es zunächst einiger Referenzpunkte. Die Referenzpunkte werden in Qt als Bilder geladen und an jeder Position für 5 Sekunden eingeblendet. Der Benutzer muss während dieser Zeit den Mittelpunkt fixieren. Insgesamt 40 der vom gaze tracker berechneten Koordinaten werden für jeden Referenzpunkt gemittelt und abgespeichert. Diese gemittelten Koordinaten werden im Folgenden als Beobachtungspunkte bezeichnet (im Gegensatz zu Referenzpunkte). Da es bei einer Verzögerung, von Seiten des gaze trackers, zu verfälschten Beobachtungspunkten kommen kann, da die verzögerten Koordinaten bei einem Blickrichtungswechsel mit in die Mittelwertberechnung eingehen, gibt es die Option „use delay“, mit der eine Verzögerung von maximal fünf Sekunden ausgeglichen werden kann.

Nach Ablauf der Kalibrierung verfügt das Programm über k Referenzpunkte und k dazu korrespondierende Beobachtungspunkte. Die Anzahl k sowie die Positionen der jeweiligen

Referenzpunkte auf dem Bildschirm ist abhängig von der gewählten Methode.

Auf die Berechnung der unterschiedlichen Methoden wird in den folgenden Abschnitten eingegangen.

3.2.1 Kalibrierung mittels lineare Skalierung

Bei der Kalibrierung mittels linearer Skalierung, werden die beobachteten Koordinaten lediglich mit einem Offset in x- und y-Richtung verschoben und für jede Achsenrichtung mit einem Skalierungsfaktor gestreckt bzw. gestaucht.

Für diese Methode sind vier Referenzpunkte nötig, die sich folgendermaßen über den Bildschirm verteilen:

- auf der linken Bildschirmhälfte mittig; im Folgenden Beobachtungspunkt links (kurz Bl)
- in der Bildschirmmitte im oberen Bereich; im Folgenden Beobachtungspunkt oben (kurz Bo)
- auf der rechten Bildschirmhälfte mittig; im Folgenden Beobachtungspunkt rechts (kurz Br)
- in der Bildschirmmitte im unteren Bereich; im Folgenden Beobachtungspunkt unten (kurz Bu)

(Die Kürzel für die korrespondierenden Referenzpunkte ergeben sich analog) Damit liegen die vier Referenzpunkte auf den Koordinatenachsen eines Koordinatensystems mit dem Nullpunkt im Zentrum des darzustellenden Bereichs. Zusätzlich wird im Mittelpunkt noch ein fünfter Referenzpunkt eingeblendet, der nach Ablauf des Kalibriervorgangs zur Überprüfung der Genauigkeit der Kalibrierung nützlich ist.

Wenn die Kalibrierung vollständig ist, können die benötigten Parameter *centerDiff*, *scaleXdirection* und *scaleYdirection* berechnet werden. *centerDiff* ist ein Array, welches die Differenz des beobachteten Mittelpunktes vom tatsächlichen Mittelpunkt in x- und y-Koordinaten enthält.

Die **Verschiebung** in x-Achsen-Richtung errechnet sich durch die Differenz aus Mittelpunkt zwischen den Beobachtungspunkten Bl und Br und dem tatsächlichen Mittelpunkt des Bildschirms. Analog dazu erfolgt die Berechnung für die Verschiebung entlang der y-Achse. Es wird also nicht ein Beobachtungspunkt für die Bildschirmmitte verwendet, sondern jeweils der Mittelpunkt der Strecken $\overline{BlBr}$ und $\overline{BoBu}$.

Der **Skalierungsfaktor** ist ebenso schnell berechnet. Es muss lediglich die Breite bzw die Höhe durch den Abstand zwischen Bl und Br bzw Bo und Bu berechnet werden. Die drei global gespeicherten Parameter *centerDiff*, *scaleXdirection* und *scaleYdirection* werden nun bei Verwendung der Kalibrierung auf jede unkalibrierte Koordinate angewandt. Im Programm sieht das dann folgendermaßen aus:

```
//translate into coordinate system with origin
//at center of screen
x -= width/2;
y -= height/2;

//translate computed center into ideal center
//of the screen
x -= centerDiff[0];
y -= centerDiff[1];

//scale in x and in y direction
x = x * scaleXdirection;
y = y * scaleYdirection;

//translate back into old coordinate system
x += width/2;
y += height/2;

newPoint->setX(x);
newPoint->setY(y);
```

Die kalibrierten Koordinaten werden dann an den Cursor übergeben.

3.2.2 Kalibrierung mittels Achsen-Projektion

Bei der Kalibrierung mittels Achsen-Projektion wird nicht vom Zentrum aus gegangen, sondern von den Ecken. Daher läuft diese Methode im Demonstrator auch unter dem Namen „corner calibration“.

Wie zuvor werden vier Referenzpunkte benötigt, die jeweils eine Ecke markieren (es müssen dabei nicht unbedingt die Ecken des Bildschirms sein, sondern können auf beliebigen Positionen der Diagonalen liegen, solange die Referenzpunkte ein orthogonales Rechteck bilden).

Bei dieser Methode wird nach Ablauf der Kalibrierung, mit Ausnahme der Mittelwertberechnung für die Beobachtungspunkte, nichts weiter berechnet. Statt dessen wird die Koordinatentransformation jedes Mal neu berechnet, wenn ein neuer unkalibrierter Punkt an die Kalibriermethode übergeben wird. Dann wird der Punkt zuerst auf die Achsen des linken oberen Koordinatensystems, welches durch die Beobachtungspunkte links unten, links oben und rechts oben aufgespannt wird, projektiert und anschließend auf die Achsen des rechten unteren Koordinatensystems, welches durch die Beobachtungspunkte links unten, rechts unten und rechts oben aufgespannt wird, ebenfalls projektiert. Anschließend werden beide Koordinaten bilinear interpoliert.

Wenn man von einer othogonalen Anordnung der Beobachtungspunkte ausgeht, wäre eine zweite Projektion nicht nötig. Da der vierte Punkt aber wesentlich vom Referenzpunkt abweichen kann, ist eine zweite Projektion mit genau diesem Punkt als Zentrum hilfreich, da dann zwischen den beiden Ergebnissen interpoliert werden kann und somit der Fehler minimiert wird.

Vermutlich ist diese Kalibriermethode speziell für gaze tracker weniger gut geeignet, da als Referenzpunkte die jeweiligen Ecken herangezogen werden. Diese sind allerdings meistens die ungenauesten Positionen. Daher ist in den meisten Fällen eher die Kalibrierung mittels linearer Skalierung zu empfehlen.

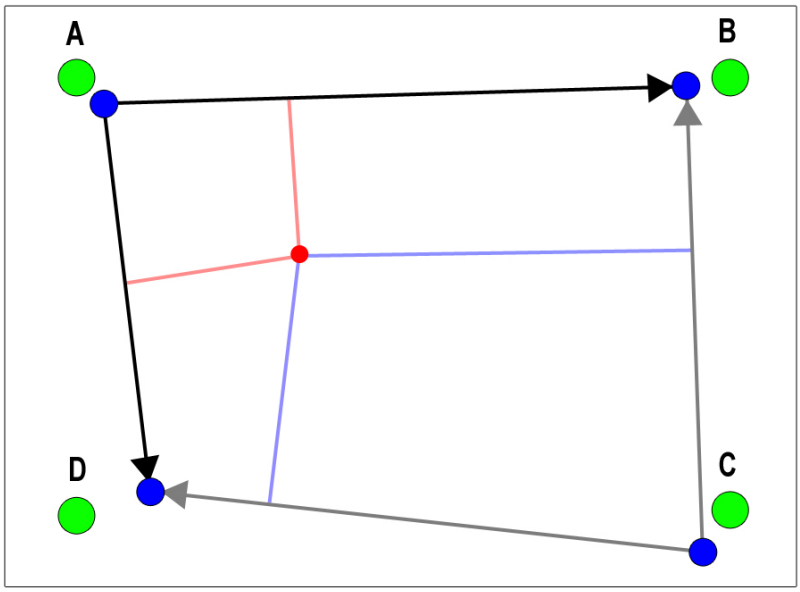

Bild 3.2: Grün = Referenzpunkte, blau = Beobachtungspunkte; der rote unkalibrierte Punkt wird auf beide Koordinatensysteme projektiert. Die beiden Werte werden dann bilinear interpoliert.

3.2.3 Kalibrierung mittels SVD

Ein anderer interessanter Ansatz wäre eine Kalibrierung unter Verwendung der Singulärwertzerlegung (SVD - Singular Value Decomposition). Die SVD zerlägt eine beliebige Matrix in drei Matrizen mit unterschiedlichen Eigenschaften. Sei M eine reelle $(m \times n)$ Matrix, dann lässt sich M durch die SVD als das Produkt von

$U * S * V^T$

darstellen, wobei U eine $(m \times m)$ und V eine $(n \times n)$ Rotationsmatrix und S eine $(m \times n)$ Diagonalmatrix ist, die wie folgt aufgebaut ist:

$$S = \begin{pmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \end{pmatrix}$$

Dabei sind die Werte der Größe nach sortiert. Vorausgesetzt die Matrix M hat nicht den vollen Rang, so gibt es in S mindestens ein σ_i, welches gleich Null ist. Daraus folgt, dass es einen Nullraum gibt. Dies ist entscheidend. Denn dann repräsentiert die zum σ_i multiplizierte Zeile aus V die Parameter, die den Nullraum aufspannen.

Beispiel: svd M = $\begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ u_{2,1} & u_{2,2} & u_{2,3} \\ u_{3,1} & u_{3,2} & u_{3,3} \end{pmatrix} * \begin{pmatrix} 3 & 0 & 0 \\ 0 & 1.4 & 0 \\ 0 & 0 & \mathbf{0} \end{pmatrix} * \begin{pmatrix} v_{1,1} & v_{1,2} & \mathbf{v}_{1,3} \\ v_{2,1} & v_{2,2} & \mathbf{v}_{2,3} \\ v_{3,1} & v_{3,2} & \mathbf{v}_{3,3} \end{pmatrix}$

Im obigen Beispiel bilden die Parameter $v_{1,3}, v_{2,3}$ und $v_{3,3}$ den Nullraum von M, da der dazu gehörende Singulärwert σ_3 gleich Null ist.

Wenn man sich nun vorstellt, dass die Koordinaten, die der gaze tracker liefert, um eine 3. Koordinate mit dem konstanten Wert 1 und eine homogene Koordinate erweitert und alle Referenzpunkte ebenfalls um die homogene Koordinate erweitert werden, so wird aus unserer Kalibrierung eine Überführung von 3D-Raumkoordinaten in 2D-Bildkoordinaten. Diese Überführung lässt sich einfach in ein Nullraum-Problem überführen, welches mit Hilfe der SVD, auch bei nicht exakten Daten, zuverlässig berechnet werden kann.

Folgende Gleichung veranschaulicht die Ausgangssituation noch einmal:

$$\begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & m_{1,4} \\ m_{2,1} & m_{2,2} & m_{2,3} & m_{2,4} \\ m_{3,1} & m_{3,2} & m_{3,3} & m_{3,4} \end{pmatrix} * \begin{pmatrix} w_{i,1} \\ w_{i,2} \\ w_{i,3} \\ 1 \end{pmatrix} = \begin{pmatrix} b_{i,1} \\ b_{i,2} \\ 1 \end{pmatrix}$$

w bezeichnet die Weltkoordinaten (also die erweiterten Beobachtungspunkte) und b die Bildkoordinaten (also die homgogenen Referenzpunkte); $i \in 0, ..., k-1$ ist die Anzahl der vorhandenen Koordinatenpaare. Die Parameter $m_{1,1}$ bis $m_{3,4}$ sind die Unbekannten zur Lösung des Problems. Durch Ausmultiplizieren und Umformen erhält man die zwei homogenen Gleichungen:

$$w_{i,1}m_{1,1} + w_{i,2}m_{1,2} + w_{i,3}m_{1,3} + m_{1,4} - b_1 w_{i,1}m_{3,1} - b_1 w_{i,2}m_{3,2} - b_1 w_{i,3}m_{3,3} - b_1 m_{3,4} = 0$$

$$w_{i,1}m_{2,1} + w_{i,2}m_{2,2} + w_{i,3}m_{2,3} + m_{2,4} - b_2 w_{i,1}m_{3,1} - b_2 w_{i,2}m_{3,2} - b_2 w_{i,3}m_{3,3} - b_2 m_{3,4} = 0$$

Diese zwei Gleichungen können mit einem Referenz- und dem dazu korrespondierenden Beobachtungspunkt erstellt werden. Durch ein Übertragen der Koeffizienten w und b in eine $(2 * k \times 12)$ Matrix *A*, lässt sich folgende Gleichung aufstellen:

$$A* \begin{pmatrix} m_{1,1} \\ m_{1,1} \\ \vdots \\ m_{3,4} \end{pmatrix} = 0$$

wobei *A* folgendermaßen aufgebaut ist:

$$A = \begin{pmatrix} w_{0,1} & w_{0,2} & w_{0,3} & 1 & 0 & 0 & 0 & 0 & -b_1 w_{0,1} & -b_1 w_{0,2} & -b_1 w_{0,3} & -b_1 \\ 0 & 0 & 0 & 0 & w_{0,1} & w_{0,2} & w_{0,3} & 1 & -b_2 w_{0,1} & -b_2 w_{0,2} & -b_2 w_{0,3} & -b_2 \\ \vdots & & & & & & & & & & & \vdots \\ w_{m,1} & w_{m,2} & w_{m,3} & 1 & 0 & 0 & 0 & 0 & -b_1 w_{m,1} & -b_1 w_{m,2} & -b_1 w_{m,3} & -b_1 \\ 0 & 0 & 0 & 0 & w_{m,1} & w_{m,2} & w_{m,3} & 1 & -b_2 w_{m,1} & -b_2 w_{m,2} & -b_2 w_{m,3} & -b_2 \end{pmatrix}$$

mit $m = k - 1$

Die Anzahl der Punkte k richtet sich dabei an der Anzahl von Unbekannten, die berechnet werden sollen. In diesem Fall sind es 12. Zusätzliche Punkte können allerdings zu genaueren Resultaten führen.

Um die Parameter $m_{1,1}$ bis $m_{3,4}$ zu erhalten, muss lediglich der Nullraum von *A* bestimmt werden. Dies ist, wie bereits erwähnt, mit Hilfe der SVD möglich. Der Vorteil der SVD gegenüber einer einfachen Nullraum-Funktion liegt darin, dass ungenaue Daten, z.B. Koordinaten, die nicht exakt zusammengehören, was bei unkalibrierten Daten häufig der Fall ist, trotzdem zu einer Bestimmung des Nullraums führen. Die Nullraumbestimmung mittels SVD liefert immer eine Lösung. Die Qualität dieser hängt allerdings stark von den Singulärwerten ab. Je kleiner der letzte Singulärwert, desto genauer der resultierende Nullraum. Auf der anderen Seite hat die SVD auch einen Nachteil. Sie funktioniert nur für lineare Zusammenhänge. Sobald nichtlineare Faktoren hinzukommen, was beim gaze tracking besonders entlang der vertikalen Achse vorkommen kann, sowie an den seitlichen Bereichen auf dem Bildschirm, kann auch die SVD keine gute Kalibrierung garantieren.

Bei der Implementierung dieser Methode wurde statt einer (3×4) Projektionsmatrix eine (3×3) Matrix verwendet. Da nur neun Unbekannte berechnet werden mussten, hat die *A* Matrix auch nur 9 Spalten und 16 Zeilen. Da die dritte Koordinate der Weltpunkte konstant ist, folgt aus der Reduktion von 3D nach 2D auf 2D nach 2D keine schlechteren Ergebnisse.

Zur Berechnung der SVD unter C++ wurde das Template Numerical Toolkit (TNT) und das JAMA/C++ Linear Algebra Package verwendet.

(weitere Informationen zu TNT unter http://math.nist.gov/tnt/)

Folgender Code veranschaulicht die Berechnung der SVD und die Bestimmung des Nullraums:

```
//Compute homogeneous coordinate
float homCoord = proMatrix[2][0] * x + proMatrix[2][1] * y
+ proMatrix[2][2];

//if homogeneous coordinate 0, then set it to small number
if (homCoord == 0) homCoord = 0.00000000000001;

//compute x coord and divide it by homogeneous coordinate
float xPoint = (proMatrix[0][0] * x + proMatrix[0][1] * y
+ proMatrix[0][2])/homCoord;

//compute y coord and divide it by homogeneous coordinate
float yPoint = (proMatrix[1][0] * x + proMatrix[1][1] * y
+ proMatrix[1][2])/homCoord;

newPoint->setX((int)xPoint);
newPoint->setY((int)yPoint);
```

Die kalibrierten Koordinaten werden dann an den Cursor übergeben.

3.3 Evaluierungsprogramm „Eprogat“

Ein Problem beim gaze tracking ist die Verifikation der Genauigkeit des Verfahrens. Daher ist die Option einer Evaluierung sinnvoll, um die Genauigkeit des Systems zu prüfen. Dabei ist großer Wert darauf zu legen, dass die Ergebnisse derart aufgelistet werden, so dass sie mit Resultaten anderer Systeme vergleichbar sind, um letztlich Aussagen über die Güte der einzelnen Verfahren machen zu können. Das Programm „Eprogat“, welches für „Evaluation program for gaze tracker“ steht, soll diese Anforderungen erfüllen.

Das Programm wurde in drei Level eingeteilt, die sich jeweils auf ein Thema beziehen:

- Bestimmung der Blickrichtungsgenauigkeit bei ruhendem Auge
- Bestimmung der Blickrichtungsgenauigkeit bei der Objektverfolgung
- Bestimmung der Treffsicherheit mittels dwell time

Im Folgenden soll auf den Ablauf, die Ziele und die Implementierung der einzelnen Level eingegangen werden.

3.3.1 Level 1: Bestimmung der Blickrichtungsgenauigkeit bei ruhendem Auge

Im ersten Teil der Evaluierung soll die Genauigkeit des gaze trackers -oder besser: die Genauigkeit der verwendeten Kalibrierung- bei ruhendem Auge gemessen werden. Dazu werden 9 Ziele in Form von Dartscheiben nacheinander für fünf Sekunden eingeblendet. Die Größe eines Ziels ist mit 100×100 Pixeln festgelegt. 500 Millisekunden nachdem eine Dartscheibe eingeblendet wurde, startet die Aufnahme der Cursorpositionen für eine Dauer von vier Sekunden, so dass die erste und letzte halbe Sekunde keinen Einfluss auf die Berechnung nimmt. Die Aufnahmefrequenz wurde mit 10 Aufnahmen pro Sekunde gewählt, so dass letztlich 40 Positionen pro Dartscheibe in die Bewertung einfließen. Um bei gaze trackern mit einer Verzögerung von mehr als 500 Millisekunden zu gewährleisten, dass die Ergebnisse nicht verfälscht werden, kann mit der Einstellung „use delay“ eine Verzögerung von bis zu 5 Sekunden ausgeglichen werden.

Nachdem die Aufnahme der Koordinaten beendet ist, erfolgt die Auswertung. Dazu wird die durchschnittliche Abweichung vom Bullseye eines jeden Zieles sowie der Mittelwert aller Abweichungen ermittelt. Darüber hinaus, wurde ein Punktesystem integriert, um der Evaluierung einen Spielecharakter zu verleihen. So gibt es 100 Punkte wenn man das Bullseye um weniger als 5 Pixel verpasst und staffelt sich dann, so dass es noch 10 Punkte gibt, wenn man das Ziel um maximal 50 Pixel verfehlt.

Sowohl die eingeblendeten Ziele als auch die erreichte Punktzahl und die durchschnittliche Abweichung zum Ziel werden am Schluss von Level 1 in einer QPixmap abgebildet und ausgegeben. Diese kann dann als png-Datei gespeichert werden.

3.3.2 Level 2: Bestimmung der Blickrichtungsgenauigkeit bei der Objektverfolgung

Im zweiten Teil der Evaluierung wird die Methode aus Level 1 auf eine Variante mit beweglichem Ziel übertragen. Ziel dieser Untersuchung ist es, herauszufinden, ob und wie weit sich die Genauigkeit der Blickrichtung ändert, wenn das fixierte Ziel in Bewegung ist. Dazu werden vier Szenarien eingeblendet. Zuerst wird ein horizontaler Balken gezeichnet, auf dem von links nach rechts, über eine Dauer von acht Sekunden, mit konstanter Geschwindigkeit ein roter Punkt wandert. Der Unterschied bei der Berechnung des mittleren Abstandes zum vorherigen Level liegt darin, dass nun neben der aktuellen Cursorposition auch die aktuelle Zielposition übergeben werden muss. Abgesehen davon wurde die Aufnahmefrequenz auf 5 Positionen pro Sekunde reduziert, um ein übersichtlicheres Resultat zu erzielen, und trotz der doppelten Dauer wie in Level 1, auch 40 Positionen als Grundlage zur Berechnung der Abweichung heranzuziehen. Dann wird wieder der Abstand zum Ziel in einem Array gespeichert, um nach dem letzten Durchlauf ausgewertet zu werden.

Dieses Szenario wiederholt sich dreimal: einmal für die vertikale Richtung von oben nach unten und entlang der beiden Diagonalen.

Nach dem letzten Szenario kann das Ergebnis wieder angezeigt werden. Dabei werden alle beobachteten Punkte mit den korrespondierenden Positionen auf dem Balken mit einer Linie verbunden. Positionen, die vom Ziel weiter als 100 Pixel entfernt sind, und somit in

diesem Level keine Punkte bekommen, werden rot, die anderen grün gezeichnet.

Eine Berücksichtigung der Verzögerung wie in Level 1 beschrieben ist nicht implementiert, wäre aber für die Zukunft wünschenswert, da sonst keine zuverlässigen Messungen möglich sind.

Denkbar wäre es ebenso, diesen Abschnitt durch Bewegungen auf elliptischen oder anderen runden Pfaden zu erweitern.

3.3.3 Level 3: Bestimmung der Treffsicherheit mittels dwell time

Im letzten Abschnitt der Evaluierung wird nun die praktische Tauglichkeit der Kalibrierung getestet. Ziel von gaze trackern ist es letztendlich Programme zu bedienen. Dafür müssen auch gewöhnliche oder spezielle Tasten fixiert und betätigt werden können. Realisiert wird dies mit Hilfe der dwell time. Sie ist in Level 3 auf 500 ms fest eingestellt.

Um zu testen, ob mit der gewählten Kalibrierung Tasten ausgelöst werden können, werden 10 100×100 Pixel große runde Buttons eingeblendet. Deren Positionen verteilen sich mit Hilfe von Zufallszahlen über den ganzen Bildschirm. Buttons, die einen roten Kreis enthalten sollen innerhalb von fünf Sekunden so genau fixiert werden, das ein Tastendruck ausgelöst wird. Gelingt dies, so verfärbt sich der Button kurz nach grün als Bestätigung und verschwindet dann. Mit jedem erfolgreichen Tastendruck erhöht sich das Punktekonto um 1000 Punkte.

Da aber ebenso vermieden werden soll, dass unabsichtlich eine Taste gedrückt wird, werden auch Buttons mit roten Quadraten eingeblendet, die beim erfolgreichen Tastendruck zu einem Punktabzug von 1000 Punkten führen. Diese Buttons verfärben sich rot und verschwinden dann ebenfalls nach kurzer Zeit.

Im Anschluss an Level 3 liefert das Resultat eine Zusammenfassung aller Ergebnisse von „Eprogat“. Diese Liste kann dann als txt-Datei gespeichert werden.

3.4 Spiel „ButtonUp“

Neben den eher technischen Werkzeugen sollte auch ein spielerischer Umgang mit gaze trackern demonstriert werden. Auf die Problematik von einer Spielsteuerung durch die Blickrichtung, sei sie nur partiell oder die grundlegende Steuerung, wurde ja bereits in Kapitel 2 behandelt.

Daher wurde eine Anforderungsliste erstellt, die das zu entwickelnde Spiel erfüllen soll, um dennoch ein nutzbares Resultat zu erzielen:

- Das Spiel muss ausschließlich mit den Augen gesteuert werden können.
- Das Spiel muss intuitiv gesteuert werden können.
- Das Spiel muss sehr übersichtlich gestaltet werden.
- Zu jedem Zeitpunkt soll nur ein Ereignis sichtbar sein
- Die Anzahl der unterschiedlichen Ereignisse soll möglichst gering sein.
- Die Anzahl der Ereignisbereiche soll auf vier begrenzt sein.
- Die Ereignispositionen sollen in der unteren Bildschirmhälfte geschehen.
- Die Ereignispositionen sollen zentriert sein und einen deutlichen Abstand zu den Bildschirmrändern an den Seiten haben.
- Das Spiel soll hauptsächlich durch eine horizontale Blickrichtungsänderung bedient werden können.

Da eine zwischenzeitliche Steuerung über Shortcuts oder die Maus beim gaze tracking eher störend ist, und das Programm auch auf Veranstaltungen präsentiert werden soll, bei denen Interessierte Zugang zu gaze trackern bekommen sollen, die körperlich nicht zu einer mechanischen Steuerung in der Lage sind, wurde großen Wert darauf gelegt, dass das Spiel komplett mit den Augen gesteuert werden kann. Dies wurde durch eine 2-Tasten-Bedienung umgesetzt. Die zum starten und beenden des Spiels notwendigen Tasten Start

und Stop wurden als eine Ampelgrafik realisiert. Diese zweistufige Ampel wurde horizontal im zentralen oberen Bildschirmbereich positioniert, so dass sie sich deutlich vom eigentlichen Spielgeschehen abgrenzt, um das Auslösen einer ungewollten Aktion zu vermeiden.

Der Spielinhalt soll intuitiv erfassbar sein und ein kurzes und spontanes Erlebnis ermöglichen. Dies ist besonders bei Veranstaltungen wichtig, bei denen Interessierte spontan am Spiel teilnehmen möchten, ohne dass sie zunächst eine Einweisung oder gar längeres Training benötigen, um das Spiel beherrschen zu können.

In Kapitel 2 wurde auf die Problematik der Ablenkung der Augen eingegangen. Damit das Auge nicht von mehreren parallel ablaufenden Aktionen abgelenkt wird und damit eventuell eine ungewollte Spielaktion ausgeführt wird, wurde in diesem Spiel die Anzahl der Aktionen pro Zeiteinheit auf eins reduziert, so dass sich der Spieler auf die zu erledigenden Aufgabe konzentrieren kann.

Als Ereignisse wurde das hoch- bzw. herunterfahren von Buttons, die als Grafiken eingebunden wurden, gewählt. Daher lässt sich auch der Name des Spiels ableiten: „ButtonUp“! Ziel des Spielers ist es, die Buttons während sie noch sichtbar sind zu treffen. Dazu muss der Spieler die virtuelle Taste so lange fixieren, bis die dwell time ausgelöst werden kann. Die Verweildauer kann beliebig eingestellt werden, ist aber auf 50 ms als Standard eingestellt.

Trifft der Spieler einen Button, so erhält er Punkte; bei einem fehlerhaften Tastendruck werden Punkte abgezogen und die Anzahl der Leben, die zu Spielbeginn mit fünf vorgegeben sind, um eins reduziert. Der Erfolg eines Treffers wird durch ein Unscharfzeichnen des Buttons visualisiert. Zusätzlich erscheint an der betreffenden Cursorposition ein grüner Kreis mit der Beschriftung „+2“, die die gewonnenen Punkte symbolisieren. Nachdem ein Button getroffen wurde, ist dieser deaktiviert, so dass erst ein neuer Button erfolgreich getroffen werden kann. Bei einem Tastendruck, der ins Leere trifft erscheint dagegen ein roter Kreis mit der Beschriftung „-1“.

Es wurden nur vier verschiedene Positionen gewählt, an denen die Buttons auftauchen können, damit genügend Platz für hinreichend große Tasten vorhanden ist. Dadurch wird gewährleistet, dass eine Fixation der Buttons erleichtert wird, besonders bei gaze trackern

mit einer geringeren Genauigkeit.

Tests mit dem gaze tracker aus [Gei07] haben ergeben, dass eine Ansteuerung im mittigen und unteren Bereich des Bildschirms entlang der horizontalen Achse wesentlich genauer und flüssiger möglich war, als entlang der y-Achse. Daher wurde das Spiel so konzipiert, dass die Buttons aus dem unteren Bildschirmrand auftauchen, nach oben wandern, so dass die obere Kante des Buttons in etwa in der Bildschirmmitte liegt und nach einer gewissen Zeitspanne wieder nach unten verschwinden. So ist es möglich, dass Spiel lediglich durch Blickrichtungsänderungen nach links und rechts zu bedienen, wenn es erstmal gestartet wurde.

Kapitel 4

Experimente und Ergebnisse

4.1 Versuchsaufbau

Die Ergebnisse dieser Arbeit wurden in einer Evaluierungsphase am Aufbau des gaze trackers aus [Gei07] getestet. Die Kamera wurde zusammen mit den Infrarotdioden auf eine Halterungsvorrichtung angebracht, so dass sie in etwa mittig auf Höhe der unteren Bildschirmkante positioniert werden konnte. Das System lief dabei auf einem Pentium 4 Rechner mit einer Taktung von 2 GHz. Bild 4.1 veranschaulicht den Aufbau der Hardware.

4.2 Versuchsdurchführung

Ziel der Versuchsreihe war es, die Güte unterschiedlicher Kalibrierungsmethoden anhand des gaze trackers aus der Diplomarbeit von Thorsten Geier zu bestimmen. Neben dem subjektiven Eindruck soll das in Kapitel 3 vorgestellte Evaluierungsprogramm „Eprogat“ Aufschluss über die Qualität der Kalibrierung geben. Der genaue Versuchsablauf unterteilt sich in folgende Phasen:

1. Lokalisation der Augen im gaze tracker und anschließende Kalibrierung als Grundvoraussetzung für die anschließende Evaluierung.

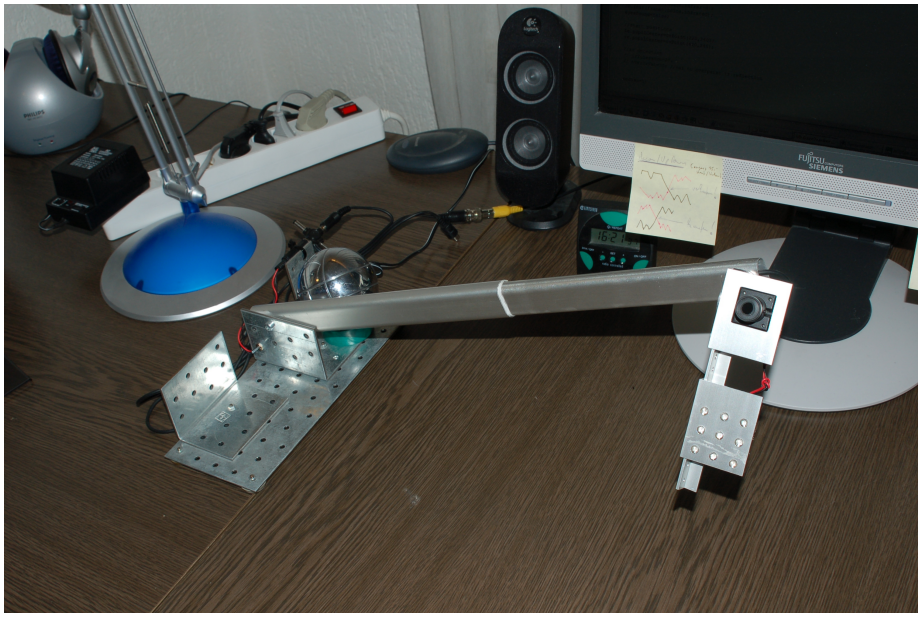

Bild 4.1: Versuchsaufbau für den Demonstrator. Die verwendete Hardware ist Bestandteil aus [Gei07].

2. Evaluierung der Genauigkeit ohne zusätzliche Kalibrierung: Es wird nur die Kalibrierung des gaze trackers verwendet.

3. Evaluierung der Genauigkeit mit zusätzlicher Kalibrierung. Dabei unterteilt sich diese Phase in folgende Stufen:

   (a) Kalibrierung mittels linearer Skalierung

   (b) Kalibrierung mittels Achsenprojektion

   (c) Kalibrierung mittels SVD

Dabei wurde in Phase 3 eine Verzögerung von 2000 ms berücksichtigt.

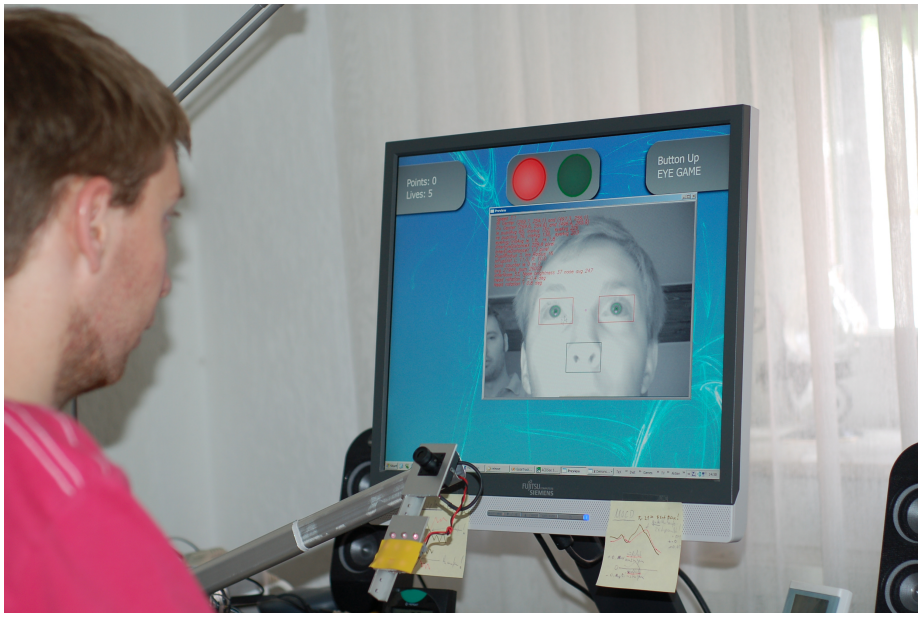

Bild 4.2: Phase 1 der Evaluierung; Lokalisation der Augen.

4.3 Ergebnisse

In Phase 1 ergaben sich keine Probleme. Die Augen wurden sehr schnell und stabil erkannt. Nach zwei bis drei Kalibrierungen im gaze tracker System, war eine gute Berechnung der Blickposition möglich, so dass der Windows-Mauszeiger mit den Augen gesteuert werden konnte.

In Phase 2 wurde diese vom gaze tracker ausgehende Kalibrierung mittels „Eprogat“ evaluiert. Dabei stellte sich heraus, dass aufgrund einer Verzögerung von knapp 2 Sekunden, eine Objektverfolgung, wie in Level 2 vorgesehen, unmöglich war, da der Testpunkt dem berechneten Punkt jedes Mal so weit voraus war, dass es zu einer Verfälschung der Daten kam. Subjektiv kann man der Kalibrierung aber eine gute Objektverfolgeung bescheinigen; denn obwohl die Abstandsberechnungen schlechte Werte lieferten, konnte beobachtet werden, dass der berechnete Punkt mit der vorhandenen Verzögerung sehr gut der Linie gefolgt ist.

Verlässlicher hingegen kann man die Ergebnisse aus Level 1 werten. Die durchschnittliche

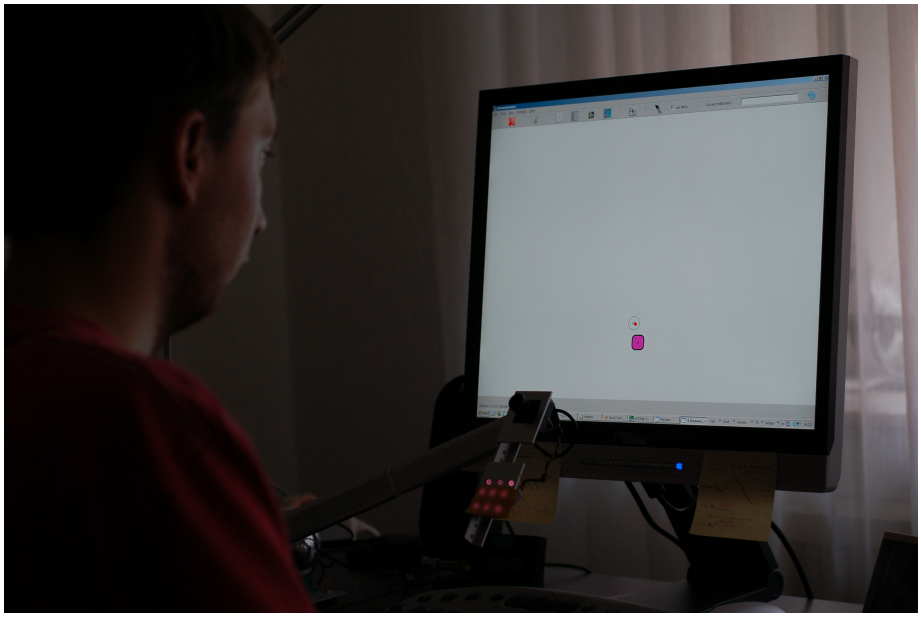

Bild 4.3: Phase 3 der Evaluierung; Kalibrierung nach drei unterschiedlichen Methoden.

Abweichung vom idealen Punkt beträgt dabei 138 Pixel, wobei die Werte auf den Hauptachsen und den Hauptdiagonalen in etwa gleich gut ausfallen. Insgesamt ergab sich ein ausgewogenes und gutes Resultat, mit dem man verlässlich Objekte verfolgen und betätigen kann. Dies wurde in Level 3 untermauert, indem alle vorgegebenen Ziele getroffen wurden.

In Phase 3 wurde nun untersucht, ob sich diese Kalibrierung durch eine zusätzliche einheitliche Kalibrierung im Demonstrator verbessern lässt. Wie in Phase 2 wurde auch hier nur Level 1 ausgewertet, da eine Objektverfolgung bei einer Verzögerung zu keinen authentischen Ergebnissen führte. Im ersten Teil, wurde die Kalibrierung mittels linearer Skalierung angewandt. Dabei zeigte sich teilweise eine deutliche Verbesserung entlang der Hauptachsen, wohingegen sich das Ergebnis entlang der Hauptdiagonalen verschlechtert hat. Besonders in den oberen Ecken waren die Ergebnisse deutlich schlechter als zuvor.

Auch die Kalibrierung mittels Achsenprojektion lieferte gute Werte entlang der Hauptachsen. Entlang der Hauptdiagonalen hingegen lieferte diese Methode schlechtere Ergebnisse, als die Kalibrierung mittels linearer Skalierung. Mit Werten von bis zu 500 Pixel Ab-

Methode	Gesamt	Hauptachsen	Hauptdiagonalen
keine Kalibrierung	138	122	135
lineare Skalierung	127	53	181
Achsenprojektion	145	67	211
SVD	221	129	284

Tabelle 4.1: Ergebnisse aus Eprogat Level 1

weichung, war eine korrekte Ansteuerung von Objekten in den oberen Ecken nicht mehr möglich.

Die letzte Methode, die Kalibrierung mittels SVD, schnitt am schlechtesten ab. Sowohl entlang der Hauptachsen als auch entlang der Hauptdiagonalen bildete dieses Verfahren das Schlusslicht in dieser Versuchsreihe. In Level 3 konnten nur 3 von 7 Objekte getroffen werden. Vielleicht lag es an der nichtlinearen Verteilung der Punkte. Vielleicht wurden die Referenzpunkte auch nicht geeignet gewählt. Hier wäre sicherlich noch Bedarf, sich dem Problem näher zu widmen, da sich dieses Ergebnis noch verbessern lässt.

Tabelle 4.1 fasst die relevanten Ergebnisse noch einmal zusammen. Die angegebenen Werte sind die durchschnittlichen Abweichungen in Pixel.

Die Angabe des Fehlers in Pixel ist sicherlich kein geeignetes Maß, um die Qualität von gaze tracker zu ermitteln. Da es hier aber um die Eignung von Kalbrierungen geht, Programme -und damit Tasten- zu bedienen, macht die Bewertung in Pixel dennoch Sinn.

Insgesamt lieferte die Kalibrierung des gaze trackers ohne zusätzliche Kalibrierung im Demonstrator ein sehr gutes und ausgewogenes Resultat. Daher wäre bei einer guten Kalibrierung eine zusätzliche Kalibrierung nicht nötig. Da die Qualität der Kalibrierung von System zu System stark variieren kann, und somit auch maßgeblich die Qualität des ganzen gaze tracker beeinflussen kann, bieten die Kalibrierungsmethoden des Demonstrators, besonders die Kalibrierung mittels linearer Skalierung, eine brauchbare Alternative.

4.4 Validierung / Verifikation

Die hier gewonnenen Ergebnisse sind allerdings nicht überzubewerten. Da während der Evaluierung, der Cursor mit der berechneten Blickrichtung beim Zielen auf die Dartscheiben zu sehen war, ist es nicht auszuschließen, dass mit den Augen unbewusst oder bewusst nachgeholfen wurde, um den Cursor näher zum Ziel zu positionieren. Besonders schwierig wurde es für die Augen, wenn sich das fixierte Bullseye und der sich in Bewegung befindliche Cursor in unmittelbarer Nähe befanden. Es ist kaum zu vermeiden, dass der Blick häufig den Fixationspunkt wechselt und somit das Ergebnis weiter verfälscht.

Die Verlässlichkeit der Ergebnisse, könnte dadurch gesteigert werden, indem der Cursor während der Evaluierung auf unsichtbar geschaltet wird. So könnte ebenfalls die Objektivität bei der Messung gesteigert werden.

Kapitel 5

Zusammenfassung

In dieser Arbeit wurde die Erstellung eines Demonstrators für gaze tracking Systeme beschrieben. Dabei wurde zunächst die Funktionsweise eines gaze trackers beschrieben, sowie auf bereits existierende Anwendungen für Menschen mit körperlicher Behinderung eingegangen. Die Einsatzgebiete für eine Benutzerschnittstelle über die Augen sind vielfältig und bietet noch viel Raum für weitere Anwendungen.

Auch wenn der Nutzen von gaze tracker in Computerspielen noch zweifelhaft ist, wurde ein kleines Spiel mit dem Titel „ButtonUp“ im Demonstrator implementiert. In einigen Versuchsreihen konnte die Tauglichkeit des Spiels festgestellt werden. Es war auch ohne Übung möglich, mehrere Minuten zu spielen und dementsprechend viele Punkte zu sammeln. Möglich wurde dies allerdings nur durch eine sehr einfache und übersichtliche Spielstruktur, die lediglich eine Aktion pro Zeiteinheit zulässt. Dadurch können die Augen immer nur auf den relevanten Inhalt fixiert werden. Um den Spielspaß zu erhöhen, müssten allerdings noch weitere Modifikationen bzw. Ergänzungen vorgenommen werden. So würde eine einfache Levelstruktur den Spielreiz wesentlich erhöhen. So könnten die Buttons mit jedem neuen Level schneller und für kürzere Zeit auftauchen. Auch wäre es interessant, die Spiellogik dahingehend zu verändern, dass der Spieler ein Leben verliert, sobald er ein Button nicht getroffen hat. Bislang verliert man ein Leben, sobald ein Tastendruck ausgelöst wird, wenn sich der Cursor nicht über einen aktiven Button befindet. Da bei der Fixation allerdings schnell mehrerer ungewollte Betätigungen der Taste erfolgen können,

was bei einer bestimmten Verzögerung nicht zu vermeiden ist, könnte dies den Spielspaß verringern. Darüber hinaus könnte die Kommunikation zwischen gaze tracker und Demonstrator verbessert werde. Bislang werden die berechneten Koordinaten des gaze trackers an den Mauszeiger übergeben und der Demonstrator reagiert lediglich auf die Mausposition. Das Problem dabei ist das Umschalten zwischen Menüsteuerung und eigentlichem gaze tracking, was bislang nur durch die Verwendung von Shortcuts im Demonstrator vermieden werden kann. Interessant wäre es nun, den Austausch der Koordinaten zwischen gaze tracker und Demonstrator über so genannte Pipes zu steuern.

Voraussetzung für die Steuerung des Spiels, wie auch jede andere Anwendung, ist eine gute Kalibrierung. In dieser Arbeit wurden drei verschieden Methoden zur Kalibrierung vorgestellt und implementiert. Am besten schnitt dabei die Methode ab, die die Eingabeparameter in vertikaler und horizontaler Richtung skalieren und den berechneten Mittelpunkt in den tatsächlichen Bildschirmmittelpunkt verschieben.

Weniger gut schnitt die Methode ab, die die Bildschirmecken als Grundlage für die Berechnung verwenden. Dies ist nicht verwunderlich, da Tests mit dem gaze tracker aus [Gei07] ergeben haben, dass die Ergebnisse in den Eckbereichen des Bildschirms am größten von der tatsächlichen Blickposition abweichen.

Warum die Methode mittels Singulärwertzerlegung so schlecht abgeschnitten hat, müsste noch genauer untersucht werden. Interessant wäre eine Untersuchung bei verschiedenen Positionen der Referenzpunkte.

Darüber hinaus ist die Kalibrierung im Demonstrator mit einer komfortablen Benutzeroberfläche ausgestattet, die es einem ermöglicht, jegliche Kalibrierung zu speichern und laden. Eingeblendete Referenzpunkte und die dazu korrespondierenden Beobachtungspunkte bieten nach einer Kalibrierung eine schnelle Überprüfung der Genauigkeit.

Für eine umfangreichere Evaluierung der Kalibrierung bietet sich das vorgestellte und implementierte Programm „Eprogat“ an. Es ermöglicht die Bestimmung der Genauigkeit von Kalibrierungen bei ruhender Fixation und bei der Objektverfolgung. Besonders die Objektverfolgung könnte noch weiter ausgebaut werden. Bislang werden nur Geraden verfolgt. Interessant wäre ebenfalls einer Untersuchung der Genauigkeit bei der Verfolgung von runden Formen. Da man bei gaze trackern immer eine bestimmte Verzögerung in Kauf

nehmen muss, müsste noch eine Funktion eingebaut werden, die eine Verzögerung bei der Objektverfolgung ausgleicht, da sonst die berechneten Werte verfälscht werden und kein Maß für die wirkliche Genauigkeit darstellen.

Alle Tests in dieser Arbeit bezogen sich auf den gaze tracker aus [Gei07]. Dieser wurde komplett unter Windows implementiert. Da der Demonstrator plattformunabhängig ist, wäre eine Einbindung des gaze trackers GoldenGaze aus [Fri05], welcher unter Linux realisiert wurde, ebenfalls ohne großen Aufwand möglich. Spannend wäre es zudem, GoldenGaze auf Windows zum Laufen zu bringen. Dazu müsste zunächst PUMA (Programmierumgebung für die Musteranalyse) unter cygwin kompiliert werden. Dies ist mit einigen Problemen behaftet. Um dennoch die Installation unter Windows zu erleichtern, befindet sich im Anhang B eine ausführliche Installationsanleitung, um Puma zumindest soweit zu kompilieren, dass mit GoldenGaze gearbeitet werden kann.

Literaturverzeichnis

[COG] COGAIN: *Communication by Gaze Interaction*, http://www.cogain.org/

[Fri05] FRITZER, Fabian: *Texteingabe per Augenbewegung*, Universität Koblenz-Landau, Campus Koblenz, Fachbereich 4 Informatik, Institut für Computervisualistik, Diplomarbeit, 2005. file:///lab/as/Archive/1802_ffko/Thesis/Fritzer2005.pdf

[Gei07] GEIER, Thorsten: *Gaze-Tracking zur Interaktion unter Verwendung von Low-Cost-Equipment*, Universität Koblenz-Landau, Campus Koblenz, Fachbereich 4 Informatik, Institut für Computervisualistik, Diplomarbeit, voraussichtlich 2007

[Geo] GEOCITIES: *The GUI Toolkit, Framework Page*, http://www.geocities.com/SiliconValley/Vista/7184/guitool.html

[IM] ISOKOSKI, Poika ; MARTIN, Benoit: *Eye Tracker Input in First Person Shooter Games*, http://www.cs.uta.fi/~poika/cogain2006/cogain2006.pdf

[Jö05] JÖNSSON, Erika: *If Looks Could Kill - An Evaluation of Eye Tracking in Computer Games*, School of Computer Science and Engineering, Stockholm, Department of Numerical Analysis and Computer Science, Diplomarbeit, 2005. http://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2005/rapporter05/jonsson_erika_05125.pdf

[Len] LENSSEN, Philipp: *Google Eyetracking*, http://blog.outer-court.com/archive/2005-03-03.html

[Sch07] SCHAEFER, Christoph: *Demonstrator für Interaktion durch Augenbewegung II*, 2007

[TTA] TOBII TECHNOLOGY AB, Schweden, http://www.tobii.com

Anhang A

Übersicht über die C++ Klassen

A.1 MainWindow

Die Klasse MainWindow ist der Ausgangspunkt des Programms. MainWindow, sowie alle anderen hier vorgestellten Klassen sind abgeleitet von QWidget. QWidget bezeichnet in Qt jedes Element, das ein Fenster bzw. ein Teil eines Fensters sein kann. Hier werden die Menüeinträge und die Toolbar generiert und mit Funktionen verknüpft. MainWindow generiert je eine Instanz der Klasse StatusWidget und CalibrationComputation. Auf diese kann dann von jedem Programm aus zugegriffen werden.

Dagegen wird für jedes Programm jeweils nur eine Instanz erzeugt, wenn dieses gewählt wird und beim Wechsel des Programms wieder zerstört.

A.2 CalibrationWidget

Die Klasse CalibrationWidget bezeichnet den Ausgangspunkt für Kalibrierungen. Als Parameter bekommt die Klasse Zeiger von MainWindow und statusWidget übergeben. In CalibrationWidget werden alle Buttons und Texte bezüglich der Kalibrierung festgelegt und ausgegeben. Auch die Resultate werden hier generiert. Der Austausch der Daten zwischen CalibrationPainting und CalibrationComputation geschieht mittels connect(source,

SIGNAL(signal()), target, SLOT(slot())).

A.3 CalibrationComputation

In der Klasse CalibartionComputation wird die Kalibrierung berechnet. Als Parameter wird MainWindow mit übergeben. Mit mainWindow->getCalibrationMode() kann die Methode zur Kalibrierung abgefragt werden.

QTimer regulieren die zeitliche Abfolge der Referenzpunkte, deren Positionen hier gespeichert sind.

getCalibratedPoint(x, y) liefert den kalibrierten Punkt als QPoint zurück.

A.4 CalibrationPainting

In der Klasse CalibrationPainting wird alles Relevante zur Kalibrierung gezeichnet.

Die in CalibrationComputation gespeicherten Positionen der Referenzpunkte werden über SIGNAL und SLOT nach CalibrationPainting übergeben und gezeichnet.

CalibrationPainting erzeugt eine Instanz der Klasse Cursor, damit dieser im paint-Fenster gezeichnet werden kann.

A.5 EprogatWidget

Die Klassen für Eprogat haben fast den gleichen Aufbau wie die der Calibration. Anstatt dass es diesmal drei verschiedene Methoden zur Kalibrierung gibt, hat Eprogat drei unterschiedliche Level. Daher müssen für jedes Level einige Variablen geändert werden. Zum Beispiel muss für Level zwei die Dauer des gezeichneten roten Punktes erhöht werden, dafür die Aufnahmefrequenz reduziert werden.

A.6 EprogatComputation

In der Klasse EprogatComputation werden die Ergebnisse der Evaluierung berechnet. Außerdem enthält die Klasse wieder alle Informationen zu den Positionen der Ziele.

A.7 EprogatPainting

In der Klasse EprogatPainting wird wieder alles Relevante gezeichnet (siehe Calibration-Painting).

A.8 ButtonUpPainting

Das Spiel ButtonUp benötigt nur zwei Klassen. In der Klasse ButtonUpPainting wird der Bildschirm gezeichnet. Die Buttons werden als Instanzen der Klassen Button geladen. Dabei liefert die Klasse Button alle wichtigen Informationen, die fürs Zeichnen nötig sind.

Darüber hinaus werden hier alle Informationen über den Spielverlauf ermittelt und ausgegeben.

A.9 Button

In der Klasse Button wird für jeden unterschiedlichen Button ein Bild als QPixmap geladen. QTimer regulieren für jede Instanz die aktuelle Position des Buttons. Außerdem wird hier gespeichert, ob der Button noch aktiv ist oder bereits getroffen wurde. Wenn der Button bereits getroffen wurde, wird ein unscharf gezeichnetes neues Bild geladen und über die alte Textur geschrieben.

Anhang B

PUMA Installation

B.1 Disclaimer

Diese Anleitung behebt nicht alle Fehler, um PUMA unter Windows mit allen Funktionalitäten betreiben zu können. Es reicht aber für die wichtigsten Funktionalitäten und um mit Puma entwickelte Programme wie GoldenGaze zum Laufen zu bringen.

Darüber hinaus kann keinerlei Garantie auf einen Erfolg gegeben werden, da je nach bereits vorhandenen Programmen die Installation beeinflusst werden kann.

B.2 Benötigte Software

Puma und damit auch GoldenGaze sind komplett unter Linux entwickelt worden. Damit die Programmierumgebung überhaupt unter Windows zum Laufen gebracht werden kann muss zunächst Cygwin installiert werden.

Puma benötigt *Windows 2000 Professional oder Windows XP* als Betriebssystem.

Damit Puma kompiliert werden kann müssen zunächst einige Schritte unternommen werden:

Schritt 1: Cygwin installieren

Lade die Setup Datei Setup.exe von der Seite http://www.cygwin.com/ herunter und folge der Installationsanleitung.

Cygwin besteht im Kern aus der Bibliothek cygwin.dll, damit aber weitere (nützliche) Programme laufen, müssen die gewünschten Pakete ausgewählt werden. Am einfachsten ist es, einfach alles auszuwählen, denn dann sind mit sehr hoher Wahrscheinlichkeit auch alle nötigen Pakete für Puma installiert. Allerdings brauchen alle Pakete zusammen etwa 2,5 GB!!! Wer eine eigene Auswahl an Paketen vornehmen möchte, sollte aber zumindest darauf achten, dass kein Compiler älter als gcc 3.3 installiert wird. Über setup.exe können jederzeit nachträglich Pakete installiert bzw. deinstalliert werden.

WICHTIG: Überprüfe, ob nicht schon eine ältere cygwin.dll vorhanden ist. Sonst funktioniert der gcc-Compiler u.U. nicht. Am besten man kopiert die cygwin.dll ins system32 Verzeichnis und überschreibt eine gegebenenfalls ältere Version.

Von nun an wird fast ausschließlich über die Cygwin bash gearbeitet (cygwin.bat öffnet die bash).

Schritt 2: sunrpc 4.0. (für xdr Bibliothek)

Lade sunrpc-4.0.cygwin1.bin.tar.gz von der Seite

http://www.nefo.med.uni-muenchen.de/~vog/source/xbinokel/cygnus_nt/

runter und extrahiere die Dateien ins Root-Verzeichnis. Kopiere dann den Inhalt von Root/usr/local/bin nach cygwin/usr/bin und den Inhalt aus Root/usr/local/lib nach cygwin/usr/lib.

Schritt 3: KONIHCL auschecken

Damit Puma überhaupt funktioniert, muss zunächst KONIHCL installiert werden.

Erstelle zunächst ein Puma-Verzeichnis, in dem später alle Programme und Puma selbst laufen sollen (z.B. C:/pumadev). Wechsel in das Verzeichnis (cd /cygdrive/c/pumadev) und checke dort KONIHCL aus:

cvs -d :pserver:$USER@cvs.uni-koblenz.de:/lab/as/REPOSITORIES/puma co KONIHCL

Wenn du Puma beim Weiterentwickeln helfen möchtest, kannst du dir ein Benutzernamen und Passwort anfordern lassen. Ansonsten ersetze *$USER* einfach durch *anonymous* und KONIHCL wird ins aktuelle Verzeichnis kopiert (Bei der Frage nach dem Passwort einfach

Enter drücken).

Schritt 4: Puma auschecken

Im Verzeichnis C:/pumadev muss nun wie folgt Puma ausgecheckt werden (Benutzername: anonymous, Passwort einfach leer lassen):

export CVSROOT=:pserver:$USER@cvs.uni-koblenz.de:/lab/as/REPOSITORIES/puma

cvs login

cvs co -r development-branch-0-99 puma

B.3 Kompilieren von KONIHCL

Zunächst muss KONIHCL installiert werden, bevor Puma installiert werden kann. Für ausführliche Installationsdetails (auf englisch) siehe INSTALL im KONIHCL-Verzeichnis.

Führe zunächst

./autogen.sh

dann

./configure –disable-dependency-tracking

und anschließend

make

aus. Vermutlich werden einige Fehlermeldungen beim Aufruf von make auftauchen (man kann sich die Fehlerdatei auch in eine Datei umleiten lassen mittels: make 2> makeError wobei makeError der Dateiname ist, in der die Fehler geschrieben werden). Das Hauptproblem wird sein, dass es sowohl unter Windows als auch unter Cygwin Dateien gibt, die den gleichen Namen haben, aber unterschiedlich arbeiten. Generell müssen die Dateien von Cygwin und nicht die von Windows verwendet werden. Also die betreffende Datei suchen und den include-Pfad direkt angeben. Dies trifft auf folgende Dateien zu:

In cygwin/lib/gcc/i686-pc-cygwin/3.4.4/include/c++/cstring Zeile 51 den genauen Pfad zur Headerdatei string.h aus cygwin angeben (z.B. C:/cygwin/usr/include), da sonst String.h

aus KONIHCL includiert wird (Windows unterscheidet nicht zwischen Groß/Kleinbuchstaben!).

Das gleiche gilt für die Datei time.h in:

- cygwin/usr/include/sched.h
- cygwin/lib/gcc/i686-pc-cygwin/3.4.4/include/c++/ctime
- cygwin/usr/include/sys/select.h
- cygwin/usr/include/sys/stat.h

Wenn die Fehlerdatei dann nur noch aus folgender Nachricht besteht

Info: resolving _optind by linking to __imp__optind (auto-import)

ist KONIHCL ausreichend installiert.

Um zu überprüfen, ob KONIHCL wirklich richtig kompiliert hat, kann man folgende Schritte machen:

Ins Verzeichniss test wechseln und

make testit

eingeben. Wenn viele Ausgaben kommen, war das Kompilieren erfolgreich!

B.4 Kompilieren von PUMA

Wenn KONIHCL fehlerfrei kompiliert hat, kann nun Puma installiert werden! Wechsel ins Verzeichnis von Puma (z.B. C:/pumadev/puma). Zunächst müssen folgende Umgebungsvariablen gesetzt werden:

export PUMADIR=/cygdrive/c/pumadev/puma

und

export PUMAARCH=cygwin

Dann den Befehl

./autogen.sh

ausführen (Wenn aclocal nicht gefunden wurde, nach automake in Cygwin suchen und ausführen).

Nun muss configure ausgeführt werden, welches überprüft, ob alle Pakete und Resourcen für Puma vorhanden sind. Über:

./configure –help

kann die Hilfe für configure aufgerufen werden. Da einige Programme und Funktionen bei der Installationen Fehler verursachen würden, werden einige Komponenten beim kompilieren deaktiviert. Daher reicht folgender configure Aufruf aus:

–without-magick -without-gtkmm -without-qt -without-openCV -with-cpu-vendor=INTEL

Zuvor muss allerdings die configure datei in C:/pumadev/puma modifiziert werden. Dazu müssen im Abschnitt, in dem auf „case sensitives“ untersucht wird, alle 4 „{ (exit 1); exit 1; };“ Aufrufe gelöscht werden.

Wenn configure erfolgreich durchgelaufen ist und KONIHCL gefunden hat, kann mittels 'make' nun Puma installiert werden.

Bevor 'make' aber ausgeführt wird, sollten noch folgende Modifikationen vorgenommen werden:

- Erstelle in cygwin/usr/include eine Datei values.h mit folgendem Inhalt:

```
// values.h
#include <limits.h>
#include <float.h>

#ifndef MAXINT
# define MAXINT INT_MAX
#endif

#ifndef MAXDOUBLE
# define MAXDOUBLE DBL_MAX
```

```
#endif
```

- in puma/modules/hippos/hippos/temd.c alle *_C* in *_CC* und alle *_X* in *_XX* umbenennen
- in cygwin/usr/include/string.h die Zeilen 83 bis 89 auskommentieren
- in puma/config/puma.mk ergänze Zeile 48 durch: *$(shell $(PUMADIR)/config/pumatool –ldflags) -lrpclib*
- Animals werden für GoldenGaze nicht benötigt. Daher wird empfohlen diese auszuschalten: in src/Makefile die Zeile unter PROGDIRS mit einen # auskommentieren:

```
PROGDIRS = \ # animals \ $(DONE)
```

Danach sollte Puma ausreichend kompiliert sein. Im optimalen Fall müssten die Dateien im lib-Ordner etwa 168 MB groß sein; aber auch bei etwas mehr als 100 MB wird es für die meisten Anwendungen reichen.

B.5 Kompilieren von GoldenGaze

Sichergehen, dass Umgebungsvariablen PUMADIR noch gesetzt ist und um export PUMA_APP_NAME=GoldenGaze hinzufügen.

dann:

./autogen.sh

und

./configure

eingeben. Anschließend müssen noch folgende Änderungen vorgenommen werden:

- in src/classes/GoldenGazeInterface.C Zeile 17 *cfmakeraw(&tis);* auskommentieren

- in src/progs/GoldenGazeMain.C und src/progs/GoldenGazeTest.C alle *abs* durch *fabs* ersetzen und *math.h* includieren
- in src/progs/Makefile füge nach der Zeile *CC = $(CXX)* eine neue Zeile mit *LDLIBS += -lkonihcl -lrpclib* ein.

Zum Schluss mit

make

kompilieren.

Zum Testen von GoldenGaze kann zunächst ein Bild als Eingabe genutzt werden. Das Bild muss genau folgende Anforderungen erfüllen:

- PNG-Format
- Größe von 768 * 576
- Grauwerte

Dieses Bild dann in src/progs ablegen. Außerdem wird voraussichtlich noch ein Template fehlen. Daher ein 30x30 Pixel großes binäres pgm-Bild mit einer Pupillenform erzeugen, wobei die Pupille schwarz ist und der Rand weiß, und ebenfalls in src/progs ablegen.

Zum Ausführen des Testprogramms

./GoldenGazeTest -loadfilewithname bild.pgm

eingeben. Neben einigen Informationen zur Laufzeit sollten einige pgm Bilder der Berechnung ausgegeben werden.