Institut für Visualisierung und Interaktive Systeme
Abteilung Mensch-Computer-Interaktion
Universität Stuttgart
Pfaffenwaldring 5a
D–70569 Stuttgart

Bachelorarbeit Nr. 12

Belastung als eine Eingabemodalität zur Interaktion mit graphischen Benutzungsoberflächen

Marius Kleiner

Studiengang:	Informatik
Prüfer:	Prof. Dr. Albrecht Schmidt
Betreuer:	Dipl.-Inf. Bastian Pfleging Dr. Niels Henze
begonnen am:	21. Mai 2012
beendet am:	12. November 2012
CR-Klassifikation:	H.1.2, H.5.2, J.4, D.2, D.3

Kurzfassung

Viele der heutigen Computerprogramme sind für eine große Zielgruppe ausgelegt. Vom unerfahrenen Heimanwender bis hin zum Experten in einem Unternehmen werden zum Teil dieselben Programme genutzt. Der knifflige Trade-Off zwischen einfacher Erreichbarkeit und guter Übersichtlichkeit erschwert es die Programmoberflächen für jeden Anwender so zu gestalten, dass die Bedienung ausreichend intuitiv und effizient funktioniert. Eine faszinierende Lösung für diese Problematik könnte eine sich selbst an die Bedürfnisse des Nutzers anpassende grafische Nutzeroberfläche bieten. Die Cognitive Load Theory von John Sweller beschreibt ein Konzept der kognitiven Belastung. Durch das kontinuierliche Messen der kognitiven Last kann ermittelt werden, ob der Nutzer beim Arbeiten mit dem Programm über- oder gar unterfordert ist. Um korrekte Anpassungen vornehmen zu können müssen zuverlässige Werte für die kognitive Belastung eines Nutzers vorliegen. Es gibt verschiedene Möglichkeiten den Grad der Anstrengung und den Stresslevel von Personen zu messen. Da einige Körperfunktionen vom Menschen nicht bewusst beeinflusst werden können und sie auf Stress reagieren, können diese zur Messung der Arbeitslast genutzt werden. Unter anderem eignen sich EKG, EEG, Hautleitwert, Transpiration und Hauttemperatur um die Arbeitslast zu bestimmen. Leider ist es häufig nötig, Messinstrumente anzulegen (z.B. EKG oder EEG). Mit einer Infrarotwärmebildkamera ist es jedoch möglich, die Hauttemperatur kontaktlos zu messen. In dieser Arbeit liegt der Fokus auf der Bestimmung der kognitiven Last mit Hilfe einer Infrarotwärmebildkamera. Mittels Gesichtserkennung auf den Bilddaten einer visuellen Kamera werden Gesichtsbereiche bestimmt. Durch das Vergleichen der Temperaturen verschiedener Gesichtsbereiche kann auf die Arbeitsbelastung eines Nutzers geschlossen werden. Im Rahmen dieser Arbeit entstand ein Programm, welches die Bild- und Temperaturdaten einer visuellen Kamera und einer Infrarotwärmebildkamera ermittelt und verarbeitet. Verschiedene Gesichtsbereiche werden im visuellen Bild bestimmt und die zugehörigen Temperaturwerte des Wärmebilds zugeordnet. Das hier entstandene Tool kann folglich die Temperaturen verschiedener Gesichtsbereiche ermitteln und als Grundlage für eine automatische Bestimmung der kognitiven Last eines Computernutzers dienen.

Abstract

Many of today's computer programs are designed for a large user group. Both, less experienced home-users and professionals sometimes use the same programs. The tricky trade-off between accessibility and clearness additionally complicates to create a graphical user interface, that makes handling of a program sufficent intuitive and efficient. A fascinating solution for this problem could be a graphical user interface that automatically adapts to the users needs. The Cognitive Load Theory by John Sweller describes a concept of cognitive load that is used in cognitive psychology. By continuously measuring the cognitive load it is possible to determine if the user is overextended or even underchallenged while using a computer program. In order to make proper adjustments it is necessary to have reliable information about the cognitive load of a user. There are several ways to measure the degree of effort and the stress level of people. Because some bodily functions can not be influenced consciously and as they react to stress, they can be used to measure the workload. Among other things, ECG, EEG, skin conductance, transpiration and determination of skin temperature suit for measuring workload. Unfortunately, it is often necessary to wear measurement devices (e.g. ECG, EEG). By using an infrared thermal imaging camera, it is possible to measure the skin temperature contactless. In this work the focus is on the determination of cognitive load using an infrared thermal imaging camera. Face regions are beeing determined by conducting facial recognition on the image data of a visual camera. The users workload can be determined by comparing the temperatures of different areas of the face. As part of this work a program which determines and processes the image and temperature data of a visual camera and an infrared thermal imaging camera. Different facial areas are determined in the visual image and assigned to the corresponding temperature values of the thermal image. The tool that has been developed here can thus determine the temperatures of different areas of the face and serve as a basis for automatic determination of the cognitive load of a computer user.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Wer kennt es nicht? Der Computer scheint zu machen was er will und jeder Versuch, den Rechner dazu zu bewegen, die gewünschte Aktion auszuführen, bringt am Ende doch ein anderes Ergebnis. Unzählige Menüs und Bestätigungsdialoge stapeln sich und scheinen um den raren Platz auf dem Bildschirm zu kämpfen. Die Oberfläche und der Programmaufbau mögen für so manchen Computernutzer unsinnig und überfordernd wirken. Die Vielzahl der Buttons und Bediensymbole fordern einem Gelegenheitsnutzer alles ab. Bestimmt hat beinahe jeder Anwender eines Computers schon einmal entnervt aufgegeben und ohne getane Arbeit den Rechner abgeschaltet. Das Problem wird dann zuweilen der grafischen Benutzeroberfläche (GUI - engl. Graphical User Interface) zugeschrieben. Die Benutzeroberfläche besteht aus der Gesamtheit der Symbole, Menüs und Buttons. Die GUI stellt dabei die Schnittstelle zwischen einem Computerbenutzer und dem eigentlichen Programm dar und bietet dem Nutzer die Möglichkeit Eingaben zu tätigen oder dem Programm zu sagen, was zu tun ist.
Doch die Anordnung und Funktionalität der GUI wirkt auf den ersten Blick oft nicht einleuchtend und bedarf einiger Einarbeitung oder Schulung. Ein Begriff, der in diesem Zusammenhang in den letzten Jahren immer häufiger genannt wird, ist „Intuitivität“. Obwohl es dieser Begriff noch nicht in den Duden geschafft hat dürfte seine Bedeutung im Bereich Computer den meisten klar sein: „Der Aufwand der benötigt wird, um sich die Bedienung eines Programms anzueignen. Je einfacher dies ist, desto intuitiver ist die Bedienung.“ [(Uso9] Obwohl in diesem Bereich viel geforscht wird und viele Erkenntnisse gesammelt werden, gibt es noch immer GUI's, deren Funktionalität nicht für jeden sofort ersichtlich ist. Selbst, oder vielleicht sogar speziell, viele Programme der namhaften Hersteller wie Microsoft oder Adobe bringen viele Nutzer bis an den Rand der Verzweiflung. Dabei müssten doch gerade von einer breiten Masse genutzte Programme besonders intuitiv bedienbar sein. Beim Betrachten der Nutzer-Zielgruppen der gängigsten Textverarbeitungsprogramme, kann festgestellt werden, dass sowohl einfache Heim-Nutzer, welche im Schnitt 1 mal pro Monat einen Brief tippen, als auch professionelle Anwender in einem großen Betrieb, welche zum Beispiel Kettenbriefe mit automatisch ausgefüllten Textelementen an hunderte Empfänger verschicken, die selben Programme nutzen. Die grafische Benutzeroberfläche muss also sowohl dem ungeübten Benutzer einen einfachen Einstieg bieten, als auch dem Profi-Anwender komplexe Funktionen einfach erreichbar zur Verfügung stellen. Das Problem ist, dass durch die Voraussetzung, dass viele Funktionen einfach erreichbar sind, die Übersichtlichkeit stark einschränkt wird. Je mehr Elemente sich in einer Button-Leiste oder in einem Menü befinden, die ein Normalanwender eventuell gar nicht benötigt, umso schneller fühlt sich ein Nutzer überfordert.
Doch wie viele Elemente pro Menü sind sinnvoll? Bekannte Forschungsarbeiten zur „magischen Sieben“ von Simon & Miller [Mil56] legen nahe, dass das menschliche Gehirn

im Schnitt eine Menge von sieben gleichen Elementen am besten verarbeiten kann. Wenn die Anzahl der Elemente in einem Menü auf sieben beschränkt wird, entstehen zahllose Untermenüs und die hohe Schachtelung führt zur umständlichen Erreichbarkeit mancher Funktionen. Die Zahl Sieben muss hier nicht als festgelegter Richtwert aufgefasst werden. Es gibt Gegenthesen die besagen, dass ein Programmnutzer sich die Menüelemente nicht merken muss, sondern diese ständig auf der Benutzeroberfläche sehen kann. [Gó] George Miller selbst sagt, dass die „magische Sieben “ nichts mit der Fähigkeit von Personen zu tun hat, Text zu erkennen, zu ordnen und zu verstehen. Bei einer Umsetzung auf dieser Basis bedarf es ausreichen Recherchen und Tests.
Dennoch wäre eine Möglichkeit interessant, die Benutzeroberfläche für jeden Benutzer anzupassen. Viele Programme bieten solche Möglichkeiten. Shortcut-Leisten können bearbeitet und Buttons können nach Belieben hinzugefügt und entfernt werden. Fenstergrößen lassen sich anpassen und sogar als Layout abspeichern. Doch welcher Nutzer macht das schon? Viele der normalen Nutzer erwarten, dass die grafische Benutzeroberfläche von vornherein optimal auf ihn abgestimmt ist. Eine faszinierende Lösung wäre eine sich automatisch an die Bedürfnisse des Benutzers anpassende grafische Benutzeroberfläche. Der Benutzer sollte niemals unter- oder überfordert sein.
In diesem Zusammenhang stellen sich folgende Fragen. Wie wäre so eine Umsetzung möglich? Wie kann der Grad der Belastung des Nutzers bestimmt werden? Ist es überhaupt, zumindest mit ausreichender Präzision, möglich? Die dazu benötigten Grundlagen werden in dieser Bachelorarbeit herausgestellt. Dazu soll diese Idee genauer behandelt werden, wobei der Fokus auf den Sensoren und der Ermittlung der kognitiven Belastung liegt.

1.1 Lösungsansatz

Eine sich automatisch an die Bedürfnisse des Benutzers anpassende grafische Benutzeroberfläche könnte die Lösung, oder zumindest eine Teillösung, für das Problem des Trade-offs zwischen Übersichtlichkeit und Erreichbarkeit der Bedienelemente sein. Zunächst benötigt der Computer Informationen darüber, wie leicht dem Benutzer die Bedienung fällt. Die einfachste Methode wäre den Benutzer direkt zu Fragen oder ihm ein entsprechendes Eingabefenster anzubieten. Dieses Vorgehen würde aber sehr subjektive Daten liefern. Wie bereits festgestellt wurde sind die meisten Menschen im Bezug auf den Umgang mit dem PC eher faul und erwarten, dass Anpassungen von allein einstellen. Einen besseren Ansatz bietet vermutlich das Messen der vegetativen Körperfunktionen. Das sind Funktionen, welche der Mensch nicht bewusst steuern kann. Dazu zählen zum Beispiel Herzschlag, Blutdruck, Hauttemperatur oder Transpiration. Da diese Körperfunktionen unbewusst durch das vegetative Nervensystem gesteuert werden und eine Person selbst keinen Einfluss darauf hat, bietet es sich an, diese Daten für eine objektive Bewertung des Anstrengungs- oder Stressgrades zu verwenden.
Das Messen von EKG[1], EEG[2], Hautleitwert, Transpiration und Hauttemperatur erfordert

[1] Elektrokardiogramm
[2] Elektroenzephalografie: Dient zur Messung der elektrischen Aktivität des Gehirns.

häufig das Anlegen von Messinstrumenten wie zum Beispiel ein EKG Messgerät. Das in den Alltag zu integrieren stellt eine Herausforderung dar. Der Benutzer soll von der Messung im Optimalfall gar nichts mitbekommen. Eine Möglichkeit, dies zu erreichen, ist die Verwendung von Wärmebildkameras. Diese können berührungslos und somit ohne das Anlegen eines Messgerätes die Hauttemperatur ermitteln. Der Fokus dieser Arbeit liegt daher auf der Messung der Hauttemperatur mit Hilfe einer Wärmebildkamera zur Ermittlung der kognitiven Belastung. Es gibt bereits Ansätze die kognitive Belastung mit Hilfe der Hauttemperatur an verschiedener Körperstellen zu ermitteln. In dieser Arbeit wird speziell ein Ansatz verfolgt, welcher auf die Auswertung der Temperaturen der verschiedenen Gesichtsbereiche aufbaut.
Benötigt wird eine Wärmebildkamera und eine visuelle Kamera zum Bestimmen der Gesichtsregionen. Der Computer wertet die gesammelten Daten aus und könnte eine sich automatisch anpassende grafische Benutzeroberfläche auf die Bedürfnisse des Nutzers abstimmen. Ist der Benutzer überfordert so könnte beispielsweise die GUI vereinfacht und die Anzahl der Menüelemente reduziert. Das würde nicht nur die GUI auf potentielle verschiedene Benutzer automatisch abstimmen, sondern auch die aktuelle Verfassung, zum Beispiel nach einem zehnstündigen Arbeitstag, berücksichtigen.

1.2 Aufgabenstellung

Im Rahmen dieser Bachelorarbeit soll untersucht werden, ob durch eine geeignete Kombination von Sensoren die Auswirkung verschieden komplexer Darstellungen und Oberflächen auf den Anwender ermittelt werden kann. Es soll die Frage beantwortet werden, durch welche Sensoren (z.B. EKG, EEG, BVP, Hautleitwert, Eye-Tracking, Brain-Computer-Interface, Wärmebildkamera) ermittelt werden kann, wie stark ein Anwender bei der Verwendung einer bestimmten Benutzeroberfläche oder beim Betrachten einer bestimmten Visualisierung belastet wird. Kernstück dieser Arbeit ist ein Framework[3], an welches verschiedene Sensoren angebunden werden können. Diese erlauben das Sammeln verschiedener Daten, wie zum Beispiel physiologische Signale. Dieses Framework verfolgt das Ziel einen Zusammenhang zwischen Sensordaten und Belastung, Stress und kognitiver Last des Nutzers festzustellen. Im Falle einer schnell gelingenden Anbindung der Sensoren soll zur Auswertung der Daten eine Nutzerstudie erarbeitet werden, sodass die zuverlässige Bestimmung des Stresslevels eines Nutzers evaluiert werden kann.
Bei raschem Fortschritt kann optional eine Testumgebung entwickelt werden, welche einerseits eine variable Arbeitsbelastung (Stress) vorgibt und andererseits eine Adaption der Benutzerschnittstelle ermöglicht. Das Ziel wäre ein Prototyp, der die visuelle Darstellung (z.B. Detailierung einer Darstellung, Hilfsinformationen) adaptiv an die Belastungssituation des Anwenders anpassen kann. Ob sich dieses Ziel im Rahmen dieser Bachelorarbeit erreichen lässt wird sich während der Bearbeitung zeigen.

[3]Ein Framework (englisch für Rahmenstruktur) ist ein Programmgrundgerüst, welches als Basis für Software benutzt werden kann. Es kann in diesem Zusammenhang auch als Ordnungsrahmen bezeichnet werden.

Zunächst werden solche Etappenziele verfolgt, welche zum fertigen und nutzbaren Framework führen. Dazu zählen das Erarbeiten der relevanten Literatur, die Suche nach verwandten Arbeiten und die Anbindung einer Wärmebildkamera. Dabei soll das Framework die Verarbeitung des Bildes zur Erfassung der Reaktion des Anwenders beherrschen, sodass alle Informationen zur Berechnung des Stresslevels vorhanden sind. Anschließend kann je nach Ergebnis der vorherigen Arbeit die Durchführung und Evaluierung von Nutzerstudien zur Untersuchung des Zusammenhangs zwischen Sensordaten und Stress/kognitiver Last des Nutzers erfolgen. Nach einer erfolgreichen Auswertung kann der Entwurf eines Konzepts für ein System entstehen, das verschiedene Visualisierungen und graphische Benutzeroberflächen darstellen kann und die Reaktion des Anwenders auf diese Darstellungen aufzeichnen und auswerten kann. Dieses Konzept kann dann als funktionaler Prototyp implementiert werden, der die Adaption einer Benutzerschnittstelle durchführen kann oder die Visualisierung bei einer bekannten Gesamtbelastung beispielhaft zeigt.

1.3 Gliederung der Arbeit

Zunächst werden in **Kapitel 2 – Hintergrund** die Grundlagen der Arbeit erläutert. In groben Umrissen wird die Funktion einer Infrarotwärmebildkamera geschildert. Um das Konzept der kognitiven Belastung verständlicher zu machen wird auf die *Cognitive Load Theory* eingegangen. Da die Implementierung in C++ umgesetzt wurde, wird eine kleine Einführung in C++ vorgestellt. Ein kleiner Einblick in die für C++ verfügbare Open Source Computer Vision Library (*openCV*) zeigt, was die *openCV* genau ist und was sie bietet.
Nach dem Erarbeiten von ausreichendem Grundwissen kann das eigentliche Thema angegangen werden. Einige wissenschaftliche Arbeiten zu diesem Thema können einen Überblick verschaffen. Im **Kapitel 3 – Verwandte Arbeiten** werden einige Publikationen behandelt und wichtige Ansatzpunkte und Informationen herausgearbeitet.
Anschließend sollte das erarbeitete Grundwissen ausreichen um die Idee von der Sensoranbindung über die Datensammlung und -Auswertung bis hin zum geplanten Ergebnis durchzugehen. Das **Kapitel 4 – Das Konzept** beschreibt, was im Optimalfall erreicht werden könnte. Was genau im Rahmen dieser Arbeit entstehen wird muss nicht unbedingt dem Konzept entsprechen.
Im **Kapitel 5 – Umsetzung und Implementierung** wird die Umsetzung des geplanten Konzepts erläutert. Zuerst erfolgt eine Bestandsaufnahme der vorhandenen Hardware. Die in C++ geschriebene Applikation, welche die Wärmebildkamera und eine visuelle Kamera anbindet und dessen Programmaufbau werden in diesem Kapitel beschrieben. Auf einige wichtige Programmteile wird hier genauer eingegangen.
Das **Kapitel 6 – Fazit und Ausblick** fasst die Arbeit zusammen, reflektiert die Ergebnisse der Arbeit und vergleicht diese mit den Zielen. Es wird geprüft, an welcher Stelle die Ziele auf Grund der Erkenntnisse, welche bei der Arbeit gesammelt wurden, angepasst werden mussten. Abschließend zeigt ein Zukunftsausblick mögliche Entwicklungen in diesem Gebiet und auch speziell zu dieser Arbeit auf.

2 Hintergrund

Ein fundiertes Hintergrundwissen und ausreichende Grundlagen helfen dabei diese Arbeit besser zu verstehen. In diesem Kapitel werden die notwendigen Hintergrundinformationen vermittelt, die das Verständnis dieser Arbeit erleichtern. Zunächst wird die Funktionsweise einer Infrarotwärmebildkamera und die Infrarotstrahlung an sich betrachtet werden. Da der Stresslevel berührungslos und ohne das Anlegen von Messinstrumenten bestimmt werden soll liegt der Fokus auf dem Arbeiten mit einer Infrarotwärmebildkamera.
Die *Cognitive Load Theory* ist ein wichtiger Bezugspunkt für diese Arbeit. Sie erklärt das Konzept der kognitiven Belastung und ist die Grundlage der möglichen Auswertungen. Darauf folgt eine Einführung in C++ und die dazugehörigen Windows Forms, mit welchen die Oberfläche des Programms umgesetzt wurde. Die Open Source Computer Vision Library bringt wichtige Funktionen für Bildbearbeitung, Gesichtserkennung und Video-Capturing[1] mit.

2.1 Temperaturmessung mit Hilfe einer Infrarotwärmebildkamera

Es gibt eine Reihe mechanischer und elektrischer Verfahren zur Temperaturmessung. Viele der mechanischen Verfahren basieren auf der wärmebedingten Ausdehnung von Flüssigkeiten oder Festkörpern. Um diese Eigenschaft entsprechender Materialien nutzen zu können ist jedoch meist die direkte Berührung mit dem Messobjekt nötig. Diese Verfahren gehören zur Kontaktthermometrie. Das Ziel dieser Arbeit ist hingegen eine berührungslose Temperaturmessung. Die Strahlungsthermometrie ermöglicht es die Temperatur von Objekten berührungslos zu messen. Eine Möglichkeit ist die Verwendung einer Infrarotwärmebildkamera. Die Wärmemessung in dieser Arbeit wird ausschließlich über eine solche Kamera stattfinden.

2.1.1 Entdeckung und Eigenschaften der Infrarotstrahlung

Friedrich Wilhelm Herschel (1738 bis 1822) entdeckte im Jahre 1800 durch einen Zufall die Infrarotstrahlung, welche damals Ultrarotstrahlung genannt wurde. Auf der Suche nach neuen optischen Materialien lenkte er Sonnenlicht durch ein Prisma. Er erkannte, dass die Wärme vom violetten bis hin zum roten Bildbereich kontinuierlich zunahm. Auf der Suche

[1] Als Video-Capturing wird der Prozess der Umwandlung des analogen Videosignals in ein digitales Videosignal bezeichnet. Es wird auch als Synonym für das deutsche Wort Bildaufnahme verwendet.

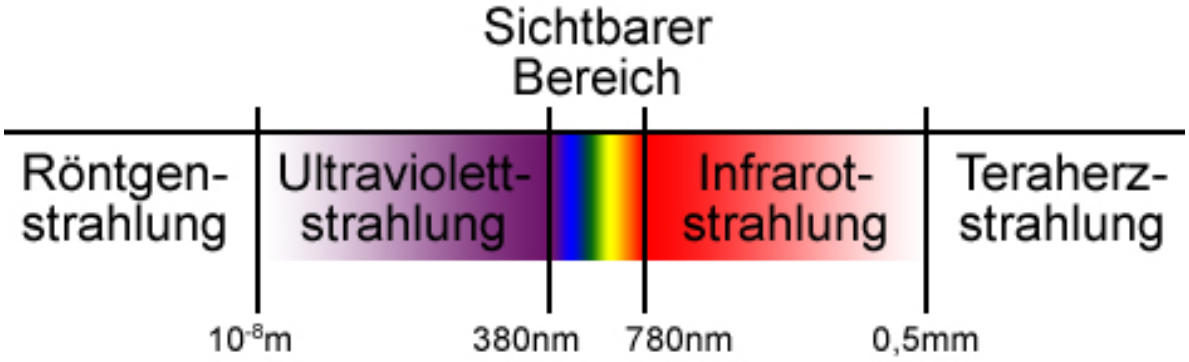

Abbildung 2.1: Einordnung des Wellenbereichs des Infrarotlichts

nach dem Punkt der maximalen Erwärmung fand er diesen weit hinter dem roten Sichtbaren Bereich. Es muss eine energiereiche Strahlung jenseits des Wellenlängenbereichs des für den Menschen sichtbaren roten Lichtbereichs liegen [Holo9].
Infrarotstrahlen sind elektromagnetische Wellen im Spektralbereich[2] mit einer Wellenlänge von 780 nm bis zu 1 mm. Die Abbildung 2.1 illustriert die Einordnung des Infrarotlichts. Die Infrarotstrahlung kann in nahes, mittleres und fernes Infrarot eingeteilt werden. Nahes Infrarot deckt einen Wellenlängenbereich von 780 nm bis zu 3000 nm ab. Bei einer Wellenlänge von 3000 nm bis zu 0,5 mm spricht man von mittlerem Infrarot. Strahlen im Wellenlängenbereich zwischen 0,5 mm und 1 mm werden dem fernen Infrarot zugeordnet. Da jeder Körper abhängig von seiner Temperatur eine bestimmte Menge infraroter Strahlung aussendet, können diese genutzt werden, um Oberflächentemperaturmessungen vorzunehmen.

2.1.2 Funktion einer Infrarot-Digital-Wärmebildkamera

Eine Infrarotwärmebildkamera ist ein optoelektronischer Sensor. Die *Optoelektronik* beschäftigt sich mit der Umwandlung von elektronisch erzeugten Daten in Lichtemissionen und umgekehrt, wobei auch Licht im Spektralbereich gemeint ist. Bei einer Infrarotwärmebildkamera handelt es sich somit im Prinzip um einen optoelektronischen Detektor. Dabei werden Sensoren benutzt, welche einen elektrischen Widerstand abhängig von den eingefangenen Lichtstrahlen zeigt. Eine Kamera, welche einen optoelektronischen Sensor nutzt, hat die gleiche Funktionsweise wie eine visuelle, bildgebende (Digital-)Kamera. Mit verschiedenen Objektiven und Einstellungen lassen sich zum Beispiel Brennweite und Weitwinkel variieren. Anstatt das Bild auf einen Film abzulichten, befindet sich bei einer Digitalkamera ein Raster

[2]Lichtwellenbereich

aus Sensoren hinter dem Kameraobjektiv. Die Sensorsignale werden mit Hilfe eines Analog-Digital-Wandlers in digitale Daten umgewandelt, welche dann abgespeichert oder mit einem Rechner weiterverarbeitet werden können.

2.2 Cognitive Load Theory

Die Aufgabe dieser Arbeit besteht darin, die kognitive Belastung eines Computerbenutzers durch eine Kombination verschiedener Sensoren zu messen. Bevor jedoch darüber nachgedacht werden kann, wie eine Messung durchgeführt werden soll, muss zunächst der zugehörigen Hintergrund verstanden werden. Um die Zusammenhänge zwischen geistiger Anstrengung, Stress und den Reaktionen des Körpers darauf richtig verstehen und deuten zu können, wird hier zuerst mit einer Theorie zur Erfassung der kognitiven Last begonnen. Ein Modell, welches dabei hilft geistige Anstrengung zu verstehen, wird in der *Cognitive Load Theory* [SMP98] beschrieben.
Die *Cognitive Load Theory* (kurz *CLT*) ist ein Modell, welches zur Beschreibung von kognitiver Last beim Lernen dient. Es weist Parallelen zur „Cognitive Theory of Multimedia Learning“ von Richard E. Mayer [May01] auf. Eigentlich ist für diese Arbeit geplant die kognitive Belastung beim Arbeiten mit dem PC zu messen. Die *CLT* bietet einige für diese Arbeit gut anzuwendende Aspekte. Sie umfasst ein ausgedehntes Wissensgebiet, weshalb hier nur die für das Verständnis dieser Arbeit wichtigen Aspekte dargestellt werden. Weitere Informationen können der Ausarbeitung „Cognitive Load and Instructional Design“ von John Sweller [SMP98] oder weiteren zur Erarbeitung von Grundlagen geeigneten Arbeiten [SAK11] [Kir02] [SG73] entnommen werden.

2.2.1 Die Struktur des menschlichen Gedächtnis

Die *CLT* teilt das menschliche Gedächtnis in verschiedene Bereiche ein, das Arbeitsgedächtnis (engl. Working Memory) und das Langzeitgedächtnis (engl. Long-Term Memory). Die einzelnen Bereiche werden in den folgenden Abschnitten erläutert.

2.2.1.1 Das Arbeitsgedächtnis

Nach J. Sweller kann das Arbeitsgedächtnis mit dem Bewusstsein gleichgestellt werden. Wir sind uns nur solchen Dingen wirklich bewusst, welche sich in unserem Arbeitsgedächtnis befinden. Alle Informationen und Denkschemata, welche sich in unserem Langzeitgedächtnis befinden, können nur abgerufen werden, wenn sie davor in das Arbeitsgedächtnis verlagert werden. Das Arbeitsgedächtnis hält die Informationen, welche es zu verarbeiten gilt. Dabei beschränkt sich die Arbeitsaktivität nicht nur auf das kurzzeitige Merken von Informationen. Schwierigere Aufgaben erfordern mehr Kapazität des Arbeitsgedächtnisses. Müssen Informationen nicht nur im Gedächtnis behalten, sondern auch miteinander verknüpft werden, erhöht das die kognitive Last. George A. Miller führte 1956 in diesem Zusammenhang die

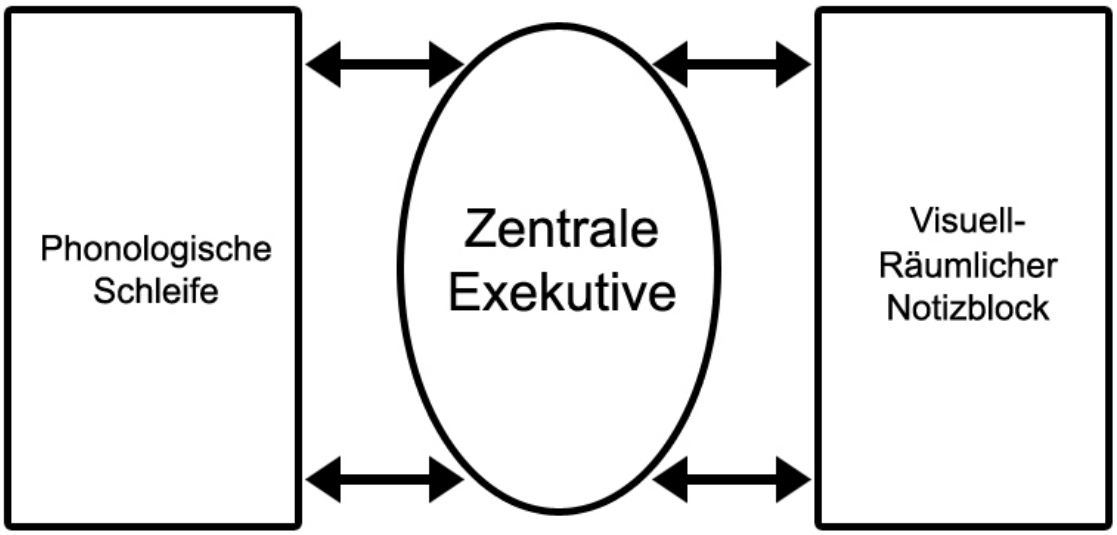

Abbildung 2.2: Aufbau/Funktion des Arbeitsgedächtnis nach Alan Baddeley [Bad10]

„magische Nummer Sieben“ ein. Er erkannte, dass ein normaler Erwachsener sich durchschnittlich sieben verschiedene Dinge merken kann, unabhängig davon, ob es Zahlen, Wörter, Buchstaben oder andere Einheiten sind. Erwähnenswert dabei ist auch, dass der Mensch sich mehr Dinge gleichzeitig merken kann, wenn diese logisch zusammenhängen. Wenn aber zum Beispiel Zahlen nicht nur im Gedächtnis behalten sondern auch miteinander verglichen werden sollen, fällt die Zahl der Dinge, die gleichzeitig verarbeitet werden können. Das Vergleichen selbst kostet zusätzlich „Platz“ im Arbeitsgedächtnis. Alan Baddeley beschreibt im Artikel „Working Memory: Theories, Models, and Controversies“ [Bad10] eine etwas genauere Einteilung des Arbeitsgedächtnisses. Wie in Abbildung 2.2 zu sehen besteht diese aus drei Komponenten. Die zentrale Exekutive entspricht einer Recheneinheit. Hier werden Informationen verarbeitet, oder, wie eben angesprochen, Zahlen verglichen. Die beiden anderen Komponenten, „Phonologische Schleife“ und „Visuell-Räumlicher Notizblock“, kann mit zwei Speichereinheiten verglichen werden. Dort werden akustische und visuelle Informationen gespeichert. Die Annahme, dass nur eine Operationsart zu einem Zeitpunkt ausgeführt werden kann, gilt hier nicht. Mehrere verschiedene Operationen können ausgeführt werden, sofern sie Ressourcen von verschiedenen Komponenten beanspruchen. Mehrere Operationen des gleichen Typs können dagegen nur schwer oder gar nicht gleichzeitig durchgeführt werden. Später (November 2000) wurde dieses Modell um die weitere Komponente „der episodische Puffer“ erweitert. Diese Komponente wird benötigt um zu erklären, aus welchem Grund der Mensch sich mehr Informationseinheiten merken kann, wenn diese in einem logischen Zusammenhang stehen.

2.2.1.2 Das Langzeitgedächtnis

Das Langzeitgedächtnis kann mit der Festplatte eines Computers verglichen werden. Es speichert Daten und gibt diese bei Bedarf wieder. Es gibt aber einen gravierenden Unterschied. Im Gegensatz zu einer PC-Festplatte ist ein Mensch sich nicht bewusst welche Daten sich genau im Langzeitgedächtnis befinden. Erst dann, wenn die Daten mit dem Arbeitsgedächtnis verarbeitet werden, ist eine Person sich des Wissens wirklich bewusst. In diesem Zusammenhang publizierten Herbert A. Simon und Kevin Gilmartin 1973 den Artikel „A simulation of memory for chess positions“ [SG73]. In verschiedenen Versuchen wurden langjährig erfahrene und gelegentlich spielende Schachspieler getestet. Die verschiedenen Probanden sollten versuchen einen Spielstand aus einem realen Schachspiel nachzuvollziehen und zu ermitteln, wie diese Konstellation zustande kam. Die erfahrenen Schachspieler konnten eine Vielzahl dieser Probleme lösen, während die Gelegenheitsspieler nur für ein paar der Konstellationen eine Lösung fanden. Es kann auch davon ausgegangen werden, dass das Arbeitsgedächtnis der erfahreneren Schachspieler nur besser für diese Aufgabe trainiert ist. Bei einem weiteren Test sollten zufällig erstellte Konstellationen ausgewertet werden. Hier zeigte sich, dass die erfahrenen Spieler keinen Vorteil mehr hatten. Der Vorteil beschränkte sich auf reale Schachspiele. Daraus folgerten Simon und Gilmartin, dass die Überlegenheit bei den realen Situationen mit einem angehäuften Wissen möglicher Schachfigurenkonstellationen zu erklären ist. Auf Nachfrage konnte jedoch keiner der erfahrenen Schachspielern erklären, warum er bei einer Aufgabe weniger Probleme hatte. Keinem war bewusst, welche Spielzüge und -Abläufe er sich genau gemerkt hatte.

2.2.2 Entwicklung kognitiver Schemata

Die Schemaentwicklung wird hier nur sehr kurz behandelt. Nach der Schema-Theorie [Mar95] werden Informationen im Langzeitgedächtnis als Schemata abgespeichert. Sich ähnelnde Erfahrungen und Informationen werden in Schemata gespeichert. Ein Schema ist ein verallgemeinertes Konzept oder ein konzeptionelles System um etwas verstehen zu können. Dadurch können der Erfahrungsschatz und das vorhandene Wissen einfacher und schneller abgerufen werden. Ein häufig auftauchendes Beispiel um diesen Sachverhalt zu erklären, ist das Schema eines Hundes. Eine Person hat eine allgemeine Vorstellung davon was ein Hund ist. Er kann Bellen, hat vier Beine, Zähne, Haare und einen Schwanz. Zusätzlich hat diese Person unter Umständen vielleicht Informationen über verschiedene Hundearten oder einzelne bekannte Hunde aus dem Umfeld. Merkt sie sich jetzt den Namen und das Aussehen eines bisher unbekannten Hundes oder lernt eine neue Hunderasse kennen, werden diese Informationen zu diesem Schema hinzugefügt und können so einfacher gemerkt und später auch einfacher wieder abgerufen werden, da die Informationen mit dem Schema „Hund“ verknüpft sind.

2.3 Ein kleiner Exkurs in C++

An dieser Stelle vermittelt ein kleiner Exkurs in die Programmiersprache C++ einige Grundkenntnisse. Hier werden einige Grundlagen über die C++ Programmierung gezeigt. Diese werden später dabei helfen, den Aufbau des während dieser Bachelorarbeit entstandenen Programms, zu verstehen. Weniger an der Programmierung interessierte Leser können diesen Teil überspringen.

2.3.1 Was ist C++

In den frühen 1970ern entwickelte Dennis Ritchie eine Portierung und Erweiterung der Programmiersprache B, welche später C genannt wurde. Mit C++ (1985) gelang die Erweiterung von C um das Konzept der objektorientierten Programmierung. Zuvor wurde es als „C mit Klassen“ bezeichnet. Bei der Entwicklung wurde auf die Kompatibilität mit C großen Wert gelegt und die objektorientierte Erweiterung machte es möglich, das Klassen-Konzept zu nutzen. [Lou08].
C++ ist eine objektorientierte, low-level[3], ANSI[4] und ISO[5] standardisierte Programmiersprache. Als low-level Programmiersprache eignet sich C++ sehr gut um schnelle und effiziente Programme zu schreiben. Nicht umsonst ist es eine der beliebtesten und meist genutzten Programmiersprachen für alle Arten und Größen von Programmen. C++ wird sowohl zur Anwendungsprogrammierung als auch zur Systemprogrammierung, wozu auch Betriebssysteme, eingebettete Systeme, virtuelle Maschinen und Treiber zählen, eingesetzt [Dav04].
Die Vorteile und Eigenschaften von C++ liegen auf der Hand. Einige davon sind:

- schnelle und effiziente Programme (da low-level)
- viele Features von high-level Programmiersprachen
- objektorientiert
- unterstützt dynamische Speicherverwaltung
- gut standardisiert
- Aufbau basiert auf ANSI-C
- es gibt Compiler für alle bedeutenden Betriebssysteme
- Nachfolger von, und somit im Aufbau ähnlich wie, C

[3] Eine low-level Programmiersprache arbeitet mit Befehlen, welche sehr nahe an den Architekturbefehlssatz angelehnt sind.
[4] American National Standards Institute
[5] Internationale Organisation für Normung

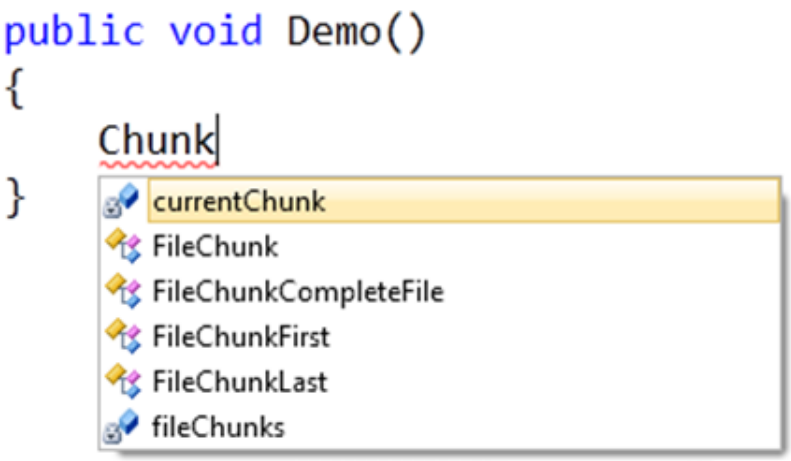

Abbildung 2.3: IntelliSense Funktion im Microsoft Visual Studio [Mic12b]

2.3.1.1 Der Editor: Microsoft Visual Studio 2010 Ultimate

Für die Entwicklung des Programms zur Sensoranbindung und zur Ermittlung der gewünschten Daten wurde *Microsoft Visual Studio 2010 Ultimate v10.0.40219.1 SP1 Rel* benutzt. Microsoft Visual Studio ist eine integrierte Entwicklungsumgebung für verschiedene Hochsprachen: C++, C++/CLI, C# , F# , C, Visual Basic .NET.
Visual Studio vereinfacht Softwareentwicklung durch eine Reihe von unterstützenden Funktionen. Neben der Multi-Monitor-Unterstützung, einem 64-Bit Compiler und Autovervollständigung bietet es auch die IntelliSense Funktion. Abbildung 2.3 zeigt ein Beispiel dazu. Microsoft beschreibt die Funktion folgendermaßen:

> „IntelliSense unterstützt eine Reihe von Features, die die Sprachreferenzen sehr benutzerfreundlich gestalten. Bei der Codierung müssen Sie das Code-Editor- oder Unmittelbarer Modus-Befehlsfenster nicht verlassen, um Sprachelemente zu suchen. Die gewünschten Informationen können im aktuellen Kontext gesucht, Sprachelemente direkt in den Code eingefügt und mithilfe von IntelliSense sogar Eingaben vervollständigt werden “ *[Mic12a]*.

Der Editor gilt als komfortabel und stellt noch weit mehr den Programmierer unterstützende Funktionen bereit.
Des Weiteren bietet Visual Studio einen Rückblick-Debugger. So ist es möglich während des Debuggens auch Laufzeitinformationen von Code vor dem Haltepunkt anzeigen zu lassen.

2.3.1.2 Grundlagen & Hello World

Nachdem der Editor betrachtet wurde folgt ein Kapitel über die C++ Programmierung selbst. Bevor damit begonnen wird wirklich Programmcode zu produzieren wird zunächst ein Blick auf einige Begrifflichkeiten geworfen. Dazu sind einige frei übersetzte Erklärungen aus „C++ for Dummies“[Dav04] zu Semantik und Syntax hilfreich.

Semantik: Ein Vokabular an Befehlen, welches für Menschen verständlich ist und ausreichend einfach in Maschinencode übersetzt werden kann.

Syntax: Eine Programmstruktur (oder Grammatik), welche es Menschen erlaubt, C++ Befehle so zu kombinieren, dass ein (möglicherweise) sinnvolles Programm entsteht.

Programm: Eine Textdatei, welche eine Sequenz von C++ Befehlen enthält, welche die C++ Syntax befolgend angeordnet sind. Die Textdatei wird auch Quelldatei *(engl. Source File)* genannt und hat die Dateiendung *.cpp*. Durch C++ Programmierung wird eine Sequenz von C++ Befehlen geschrieben, welche in Maschinencode übersetzt werden können.

Diese Erklärungen für Syntax und Semantik sind etwas schwer zu greifen. Vermutlich einfacher und intuitiver greifbar, aber unkonkret können diese Begriffe so beschrieben werden:

Semantik: Der Sinn der hinter dem Getippten steht.

Syntax: Syntax ist was getippt wird (die Syntaxregeln befolgend).

Das Betrachten eines ersten Programms hilft dabei diese Begrifflichkeiten zu verdeutlichen. Nach einer Konvention ist das erste Programm eines Tutorials (oder in diesem Fall einer kleinen Einführung) ein „Hello World!“-Programm. Dieses tut nichts anderes als auf der Konsole „Hello World!“ auszugeben.
Folgendes Codebeispiel zeigt ein solches Programm in C++:

```
1  // C++ Hello World
2
3  #include <iostream>
4
5  int main()
6  {
7
8          std::cout << "Hello World!";
9
10         return(0);
11
12 }
```

Listing 2.1: C++ Codebeispiel: Hello World!

Die einzelnen Bestandteile dieses Programms werden durch das Aufteilen in seine einzelnen Codezeilen verständlich.
Bei der ersten Zeile handelt es sich um eine Kommentarzeile:

```
// C++ Hello World
```

Hier steht ein beschreibender Text, welcher mehr über das Programm oder den entsprechenden Programmteil aussagt. Kommentare werden später noch einmal genauer behandelt.
Die nächste Anweisung ist eine so genannte *Präprozessor-Direktive*.

```
#include <iostream>
```

Präprozessor-Direktiven sind Anweisungen, welche vom Compiler ausgeführt werden und beginnen immer mit einem # -Zeichen. Der Compiler ist dafür zuständig den C++ Code in Maschinencode umzuschreiben, sodass ein Computer damit arbeiten bzw. das Programm ausführen kann. Diese Anweisung fügt den Code welcher in dem Package *iostream* enthalten ist an dieser Stelle ein. Das wird benötigt um später Funktionen nutzen zu können, welche in *iostream* definiert wurden. Am Beginn jeder C++-Code-Datei stehen an oberster Stelle (mit Ausnahme von Kommentaren) einige Präprozessor-Direktiven.
Anschließend werden die verschiedenen Funktionen deklariert. Eine Funktion, die in jedem C++ Programm vorhanden sein muss, ist die *main*-Funktion.

```
int main()
{

        std::cout << "Hello World!";

        return(0);

}
```

Eine Funktion kann einen Rückgabewerttyp und Parameter besitzen. Der Typ des Rückgabewertes steht vor dem Funktionsnamen (hier *main()*). Der Rückgabewert der *main*-Funktion ist ein Integer, ein einfacher Datentyp welcher ganze Zahlen repräsentiert. Das wird mit dem Schlüsselwort *int* gekennzeichnet. Die *main*-Funktion gibt immer einen Integer zurück. Andere Funktionen können aber auch keinen Wert zurückgeben. Das beschreibt das Schlüsselwort *void*.
Da der Rückgabewert als Integer gekennzeichnet wurde muss auch einen Wert mit diesem Typ zurückgegeben werden. Der *return*-Befehl beendet die Ausführung der Funktion an dieser Stelle und gibt den als Parameter angegebenen Wert zurück.

```
        return(0);
```

Welcher Wert hier zurückgegeben wird, ist hier nicht weiter von Belang.
Vor dem *return*-Befehl kann eine Reihe von Befehlen stehen.

```
        std::cout << "Hello World!";
```

Diese Befehlszeile bewirkt, dass bei Ausführung des Programms auf der Konsole „Hello World!“ ausgegeben wird. Das Schlüsselwort, das die Ausgabe auf der Konsole bewirkt, ist *cout*. Die Zeichenkette *std::* beschreibt den sogenannten Namespace der *cout*-Funktion. Eine mögliche Erklärung was Namespaces sind ist folgende:

> „Namespaces allow to group entities like classes, objects and functions under a name. This way the global scope can be divided in „sub-scopes“, each one with its own name. “*[cpl12]*

Verschiedene Funktionen, Objekte und Klassen werden demnach durch Namespaces in kleinere Bereiche unterteilt. Die *cout*-Funktion wird dem *std*-Namespace zugeordnet.
Die Ausführung des Programms liefert folgendes Ergebnis:

```
Hello World!
```

Listing 2.2: Ergebnis der Ausführung des „Hello World!“ Programms.

Header- und Quelldateien Ein typisches C++ Programm enthält Header- und Quelldateien. In Headerdateien werden für gewöhnlich Datentypen, Variablen und Funktionen deklariert. Das bedeutet, dass an dieser Stelle dem Compiler der Datentyp, die Funktion oder die Variable bekannt gemacht wird.
In Quelldateien werden die Funktionen implementiert die in der zugehörigen Headerdatei deklariert wurden.
Ein Beispiel dazu:

```
1 class FooClass
2 {
3         FooClass();
4         ~FooClass();
5         void doFoo();
6 };
```

Listing 2.3: C++ Headerdatei Beispiel foo.h

In der Headerdatei wird eine neue Klasse namens *FooClass* deklariert. In dieser Klasse wird neben dem Konstruktor (*FooClass()*) und dem Destruktor (∼ *FooClass()*) die Funktion *dooFoo()* deklariert. Was diese Funktion macht, kann ohne Kommentare an dieser Stelle noch nicht erkannt werden. Konstruktoren und Destruktoren werden später behandelt.

```
1  #include "foo.h"
2  #include <iostream>
3
4  FooClass::FooClass()
5  {
6  }
7
8  FooClass::~FooClass()
9  {
10 }
11
12 void FooClass::doFoo()
13 {
14
15         std::cout << "foobar";
16
17 }
```

Listing 2.4: C++ Quelldatei Beispiel foo.cpp

Die zugehörige Quelldatei bindet zuerst die Headerdatei *foo.h* ein und definiert dann die Funktion *doFoo()* der Klasse *FooClass*. Die Zeichenkette *FooClass::* vor dem Funktionsnamen zeigt an, zu welcher Klasse die hier definierte Funktion gehört.

Kommentare Kommentare sind insbesondere bei größeren Softwareprojekten sehr wichtig. Die Funktion eines Programms ist nicht immer direkt aus dem Quellcode ersichtlich. Daher sollte der Code mit ausreichend Kommentaren versehen werden.
Bei C++ gibt es zwei Möglichkeiten Kommentare einzufügen. Die wohl einfachere Art ist der einzeilige Kommentar.

```
1 // Hier steht ein Kommentar
2 std::cout << "Das␣ist␣eine␣Codezeile␣nach␣einem␣Kommentar";
```

Listing 2.5: C++ einzeiliger Kommentar

Alle Zeichen einer Zeile, die nach der Zeichenfolge // stehen, werden vom Compiler ignoriert. Soll eine mehrzeiliger Kommentar mit dieser Kommentierungsart eingefügt werden, muss jede Zeile wieder erneut die Zeichen // enthalten.

```
1 // Hier steht ein mehrzeiliger Kommentar
2 // Das ist die zweite Zeile des Kommentars
3 std::cout << "Das␣ist␣eine␣Codezeile␣nach␣einem␣Kommentar";
```

Listing 2.6: C++ mehrzeiliger Kommentar

Mehrzeilige Kommentare können auch als Blockkommentar realisiert werden.

```
1 /**
2 *      Hier steht ein Blockkommentar
3 *      Das ist die zweite Zeile des Blockkommentars
4 */
5 std::cout << "Das␣ist␣eine␣Codezeile␣nach␣einem␣Kommentar";
```

Listing 2.7: C++ Blockkommentar

Die Zeichenfolge /** startet einen Blockkommentar. Alle nachfolgenden Zeichen werden auch über die entsprechende Zeile hinaus ignoriert, bis der Blockkommentar mit den Zeichen */ beendet wird.
Gute Kommentare sind wichtig um sich schnell im Quellcode zurechtfinden zu können. Für gewöhnlich stehen die Kommentare zu einer Codezeile, welche einen Kommentar benötigt, in der Zeile davor. Zu viele Kommentare sollten jedoch auch vermieden werden. Auf unnötige Kommentare zu selbsterklärenden Codeteilen sollte verzichtet werden.
Jede Datei sollte mit einem Blockkommentar beginnen. Darin wird der in dieser Datei enthaltenen Code beschrieben und Informationen etwa über Autor, Version und Änderungsdatum hinzugefügt.
Auch Funktionen sollten direkt vor der Deklaration so kommentiert werden, dass deutlich wird, wie diese Funktion arbeitet und wie sie aufzurufen ist. Rückgabewert und Parameter aufzulisten und zu erklären kann dabei helfen. Listing 2.8 zeigt ein Beispiel dazu.

Datentyp	Wertebereich
bool	true/false
(signed) char	-128 - 127
unsigned char	0 - 255
enum	-2.147.483.648 - 2.147.483.647
(signed) int	-2.147.483.648 - 2.147.483.647
unsigned int	0 - 4.294.967.295
short int	-32.768 - 32.767
(signed) long	-2.147.483.648 - 2.147.483.647
unsigned long	0 - 4.294.967.295
float	$3{,}4\times10^{-38}$ – $3,4\times10^{38}$
double	$1{,}7\times10^{-308}$ – $1,7\times10^{38}$
long double	$3{,}4\times10^{-4932}$ – $3,4\times10^{4932}$

Tabelle 2.1: Datentypen in C++

```
1  /**
2  *      Dies ist eine Headerdatei, welche die Klasse "FooClass" deklariert.
3  *      Diese Klasse tut nichts besonderes, da sie nur als Beispiel
4  *      entwickelt wurde.
5  *
6  *      \file foo.h
7  *      \author Max Mustermann
8  *      \date 01-07-2012
9  *      \version 1.0a
10 */
11 class FooClass
12 {
13
14         /**
15         *      Konstruktor
16         */
17         FooClass();
18
19         /**
20         *      Destruktor
21         */
22         ~FooClass();
23
24         /**
25         *      Diese Funktion gibt mehrfach "Foobar" auf der Konsole aus
26         *      Die Anzahl der Ausgaben wird vom Parameter <anzahl> bestimmt.
27         *
28         *      \return void
29         *      \param anzahl  Integer, welcher bestimmt wie oft
30         *                                  "Foobar" ausgegeben wird
31         */
32         void doFoo(int anzahl);
33
34 };
```

Listing 2.8: C++ Headerdatei Beispiel foo.h mit Kommentaren

Datentypen und String-Klassen Tabelle 2.1 zeigt einige Datentypen in C++. Variablen vom Typ *bool* speichern Wahrheitswerte, *true* oder *false* bzw. *wahr* oder *falsch*. Für Zahlenwerte gibt es eine Reihe von Datentypen mit verschiedenen Vor- und Nachteilen. Eine *char*-Variable besteht aus 8 Bit und repräsentiert ein darstellbares Zeichen.

Klassen, welche mehr Funktionalität bereitstellen als diese Standarddatentypen, sind letztendlich immer eine Kombination dieser Datentypen. Ein gutes Beispiel dafür ist die *String*-Klasse. Ein String ist im Prinzip einfach ein Array aus Variablen des Typs *char*. Doch die *String*-Klasse stellt noch weitere Funktionen bereit. Hinzu kommt, dass es in C++ verschiedene String-Klassen gibt, welche jeweils unterschiedliche Eigenschaften und Funktionen haben. Um einen String einer Klasse in einen String einer anderen *String*-Klasse zu konvertieren, kann die Marshal-Funktion von C++ benutzt werden (siehe Listing 2.9).

```
1 // Deklariere Variablen verschiedener String-Klassen
2 System::String^ string1;
3 std::string string2;
4 // Konvertiere string1 und weise dessen Wert der Variable string2 zu
5 string1 = "foobar";
6 string2 = msclr::interop::marshal_as<std::string>(string1);
```

Listing 2.9: C++ Marshal-Stringkonvertierung Beispiel

Ein *System::String^* wurde hier in ein *std::string* konvertiert. Würde der std::string nicht mit Hilfe der *marshal_as*-Funktion konvertiert werden, würde der C++ Compiler an dieser Stelle einen Fehler werfen.

Klassen und Instanzen Bei objektorientierten Programmiersprachen gibt es Klassen und Objekte. Die Klasse ist die zentrale Datenstruktur in C++ und kapselt die Funktionen und Daten einer Klasse vom Rest des Programms ab. Eine Klasse ist eine Vorlage, aus der sogenannte Instanzen zur Programmlaufzeit erzeugt werden können. Eine Instanz einer Klasse ist ein Objekt mit den Funktionen, Daten und Eigenschaften, die für diese Klasse definiert wurden.
Listing 2.3 und Listing 2.4 deklarieren und implementieren die Klasse *FooClass*. Um eine Instanz von dieser Klasse zu erstellen muss dessen sogenannten Konstruktor aufgerufen werden. Das ist eine Funktion, die den selben Namen hat wie die Klasse selbst. Im Falle unseres Beispiels *FooClass()*. Der Code im Konstruktor wird beim Erstellen einer Instanz der Klasse ausgeführt. Der Destruktor ist das Gegenstück dazu und unterscheidet sich im Namen nur durch eine Tilde am Anfang. Der Destruktor wird aufgerufen, wenn die Instanz zerstört werden soll. Listing 2.10 zeigt die Erstellung einer Instanz der *FooClass*-Klasse.

```
FooClass fooClassInstance = new FooClass();
```

Listing 2.10: C++ Instanz einer Klasse

Referenzen und Pointer Ein Variablenzugriff kann durch den direkten Aufruf einer Variable, aber auch einfacher durch Verwenden einer weiteren Variable, geschehen.

Die weitere Variable enthält die Speicheradresse der Variable, auf welche zugegriffen werden will. Das nennt man eine Zeigervariable, oder einfach Zeiger (engl. Pointer). Listing 2.11 zeigt ein Beispiel einer Zeigervariable. Beim Deklarieren einer Zeigervariable wird ein * vor den Namen der Variable eingefügt.

```
int *zeiger_auf_a;
```

Beim Zuweisen einer Speicheradresse zu einem Zeiger setzt wird ein & vor den Variablennamen gesetzt. Das bedeutet, dass nicht der Inhalt, sondern die Adresse der entsprechenden

```
1  // Deklariere normale und Zeiger int Variablen
2  int a;
3  int *zeiger_auf_a;
4  //  Flle  a mit Wert und lasse den Zeiger auf a zeigen
5  a = 42;
6  zeiger_auf_a = &a;
7  std::cout <<  a  <<  std::endl;
8  std::cout <<  *zeiger_auf_a << std::endl;
9  std::cout <<  zeiger_auf_a << std::endl;
10 // Schreibe einen neuen Wert in a mit Verwendung des Zeigers
11 *zeiger_auf_a = 12;
12 std::cout <<  a  <<  std::endl;
```

Listing 2.11: C++ Zeigervariable Beispiel

Variable übergeben wird.
Das Ausführen des Programmcodes aus Listing 2.11 führt zu einer Konsolenausgabe wie in in Listing 2.12.

```
42
42
0x1234
12
```

Listing 2.12: Ergebnis der Ausführung des Programms von Listing 2.12

Sowohl über den Gebrauch der Variable *a* direkt, als auch über den Zeiger *zeiger_auf_a* kann der Inhalt von *a* ausgegeben werden. Diese Codezeile greift über die Adresse, die in *zeiger_auf_a* gespeichert ist, auf den Inhalt von *a* zu:

```
std::cout <<  *zeiger_auf_a << std::endl;
```

Um das zu tun muss die Variable *zeiger_auf_a* dereferenziert werden. Das bedeutet, dass auf den Inhalt und nicht auf die Adresse selbst zugegriffen wird. Mit einem * vor dem Variablennamen wird eine solche Zeigervariable dereferenziert. Ohne das Dereferenzieren würde eine Speicheradresse ausgegeben, wie in der dritten Zeile der Ausgabe von Listing 2.11 zu sehen.

```
0x1234
```

Eine weitere Alternative, auf Variablen zuzugreifen, sind Referenzen. Diese unterscheiden sich von der Art der Verwendung nur wenig von Zeigern.
Beim Erstellen einer normalen Variable wird im Speicher Platz für den Inhalt dieser Variable reserviert. Für einen Zeiger wird nur Platz für das Speichern der Adresse reserviert. Eine Referenz zeigt ebenfalls auf ein Objekt, gibt bei direktem Aufruf aber nicht die Adresse des

Objekts, sondern das Objekt selbst, zurück. Das Beispiel in Listing 2.14 sollte das deutlicher machen.
Das Programm erzeugt folgende Ausgabe:

```
a=3 / b=9 / x=3
a=4 / b=9 / x=4
a=9 / b=9 / x=9
a=10 / b=9 / x=10
```

Listing 2.13: Ergebnis der Ausführung des Programms von Listing 2.14

a und b sind zwei normale Variablen für die im Speicher ein entsprechendes Integer-Objekt liegt, auf das sie zeigen. Die Referenzvariable x zeigt auf das selbe Objekt wie a. Daher ist bei der Ausgabe der Wert von a mit dem von x in dieser Ausgabe immer identisch.

```
int& x = a;
```

Eine Referenz muss beim Erstellen direkt zu einem bereits bestehenden Objekt zugewiesen werden.

Daher würde folgende Zeile an dieser Stelle einen Fehler werfen:

```
int& x;
```

2.4 Open Source Computer Vision Library

Während dieser Bachelorarbeit entstand ein Computerprogramm, welches die verschiedenen Sensoren (in diesem Fall eine visuelle und eine Infrarot-Wärmebildkamera) anbindet, um anschließend die Bilddaten auszuwerten. Dabei wurde die Programmiersprache C++ verwendet. Das automatisierte Arbeiten mit Bilddaten am Computer ist schwierig, da für eine Maschine ein Bild nur eine Liste oder ein Raster verschiedener Farbwerte ist.
Für Menschen ist es ganz natürlich in einem Bild nach bekannten Mustern zu suchen und diese verschiedenen Schemata zuzuordnen. Wir erkennen sofort, wenn ein Mensch vor uns steht und wir sehen sogar, ob es sich um eine reale Person oder um das Bild eines Menschen handelt. Das alles passiert unterbewusst. Wir Menschen müssen uns im Regelfall nicht speziell anstrengen um etwas auf einem Bild zu erkennen. Was ein Mensch ist und wie er erkannt werden kann, lernen wir Menschen im Laufe unserer ersten Lebensjahre. Während unseres Lebens sehen wir immer mehr Objekte und Dinge, welche wir immer schneller und besser zu erkennen und einzuordnen lernen, da wir auf einen immer größeren Erfahrungsschatz zurückgreifen können.
Maschinen beizubringen, ebenfalls Objekte auf einem Bild zu erkennen und diese dann einer bestimmten Art oder Gruppe zuzuordnen, ist ein schwieriges Unterfangen. Im Gegensatz zu

```
1  #include <stdio.h>
2  #include <conio.h>
3  #include <iostream>
4
5  int main()
6  {
7          // Deklariere zwei normale Variablen und eine Referenzvariable
8          int a = 3;
9          int b = 9;
10         int& x = a;
11         // Ausgabe der drei Variablen
12         std::cout << a << "/" << b << "/" << x << std::endl;
13         // Inkrementiere x
14         x++;
15         // Ausgabe der drei Variablen
16         std::cout << a << "/" << b << "/" << x << std::endl;
17         // Inkrementiere x
18         x = b;
19         // Ausgabe der drei Variablen
20         std::cout << a << "/" << b << "/" << x << std::endl;
21         // Inkrementiere x
22         x++;
23         // Ausgabe der drei Variablen
24         std::cout << a << "/" << b << "/" << x << std::endl;
25         getch();
26         return 0;
27 }
```

Listing 2.14: C++ Referenz Beispiel

Menschen lernen Computer für gewöhnlich nicht von alleine das Erkennen verschiedener Objekte. Es gibt bereits Versuche Computern das Lernen beizubringen. Die Google X Gehirnsimulation [Plu12] ist ein gutes Beispiel dafür. Dabei wurde versucht einem aus 1.000 Computern simulierten Gehirn verschiedene Bilder vorzulegen, auf welchen die Simulation Objekte erkennen und kategorisieren sollte. Das funktionierte mit einer erstaunlich hohen Trefferquote. Dennoch werden gerade Bilderkennungsalgorithmen noch von Menschen für den Computer geschrieben. Dem Computer muss ganz genau gesagt werden worauf geachtet werden muss, wenn zum Beispiel ein Gesicht erkannt werden soll. Das ist schwierig, da dem Computer dies mit Hilfe von Größenverhältnissen, geometrischen Eigenschaften und weiteren Merkmalen geschehen muss. Mit einer Beschreibung wie „Ein Kopf hat eine ovale Form“ kann ein Computer nur wenig bis gar nichts anfangen. Für einen Computer stellt sich das Problem anders dar. Eine ovale Form muss aus einem Haufen an Farbwerten erkannt werden.
Diese Aufgabe übernimmt glücklicherweise die *Open Source Computer Vision Library*.

```
1  #include <opencv2/opencv.hpp>
2  #include <opencv/highgui.h>
3  #include <conio.h>
4  #include "stdio.h"
5
6  int main(array<System::String ^> ^args){
7
8         // Erstelle ein openCV VideoCapture Objekt und
9         // baue eine Verbindung zum Video Device auf
10        int device_num = 0;
11        cv::VideoCapture* cap = new cv::VideoCapture(device_num);
12        // Deklariere ein cv::Mat Bild zum Speichern
13        // eines Bildframes
14        cv::Mat gatherimg;
15
16        // Mache nur weiter, wenn eine Verbindung zu einem
17        // Video Device hergestellt werden konnte
18        if (!cap->isOpened())
19               return 0;
20
21        // Benutze das VideoCapture Objekt um einen
22        // Frame aus dem VideoDevice zu laden
23        *cap >> gatherimg;
24
25        // Zeige das Bild in einem Fenster mit Hilfe
26        // einer openCV Funktion aus dem highgui Modul
27        // an
28        cv::imshow( "WebCam Snapshot", gatherimg );
29
30        // Warte bis eine beliebige Taste gedrckt wurde
31        cvWaitKey(0);
32
33        // Zerstre das Fenster und baue die Verbindung
34        // zum Video Device ab
35        cap->release();
36        cvDestroyWindow( "WebCam Snapshot" );
37
38        return 0;
39 }
```

Listing 2.15: C++ *openCV Video Capturing* Beispiel

2.4.1 Was ist openCV?

Die *Open Source Computer Vision Library* ist eine freie Programmierbibliothek. Sie stellt unter anderem solche Algorithmen zur Bildverarbeitung bereit, wie sie für diese Bachelorarbeit benötigt werden. *Open Source* bedeutet, dass der Quellcode dieser Library öffentlich ist und von jedem eingesehen werden kann. Eine Library (zu Deutsch Prorammbibliothek) ist eine Sammlung von Klassen, Methoden, Konstanten und Unterprogrammen. *Computer Vision* bedeutet auf Deutsch in etwa *maschinelles Sehen*.

2.4.2 Von openCV bereitgestellte Funktionen

Für *openCV* sind einige Module verfügbar, welche jeweils Algorithmen für andere Teilgebiete der Bildverarbeitung bereitstellen. Auf der *openCV*-Website [tea12a] werden unter anderem folgende Module vorgestellt.

core Dieses kompakte Modul beinhaltet Basisdatenstrukturen wie zum Beispiel die cv::Mat Klasse welche Bilder repräsentiert.

imgproc Ein Bildbearbeitungsmodul für lineare und nicht-lineare Bildfilterung, geometrische Bildtransformation, Histogramme und mehr.

video Ein Videoanalysierungstool für Trackingalgorithmen, Hintergrundentfernung und optische Flusserkennung.

calib3d Ein Modul für Kamerakalibrierung, 3D Rekonstruktion, Bildsynchronisierung und mehrdimensionale Geometriealgorithmen.

highgui Ein einfach zu benutzendes Interface für Bildaufnahme, Bild- und Video-Codecs und einfache Funktionen für UI-Umsetzung.

objdetect Bietet Objekterkennung für Gesichter, Augen, Mund, Nase, Personen, Autos und mehr.

GPU Enthält GPU Beschleunigungsalgorithmen für verschiedene andere Module.

Weitere Module Es gibt noch weitere Module, zu denen auch einige Helfer-Module gehören. *OpenCV* wird vor allem für seine Geschwindigkeit, seine modulare Struktur und die große Menge der Algorithmen aus neusten Forschungsergebnissen geschätzt.

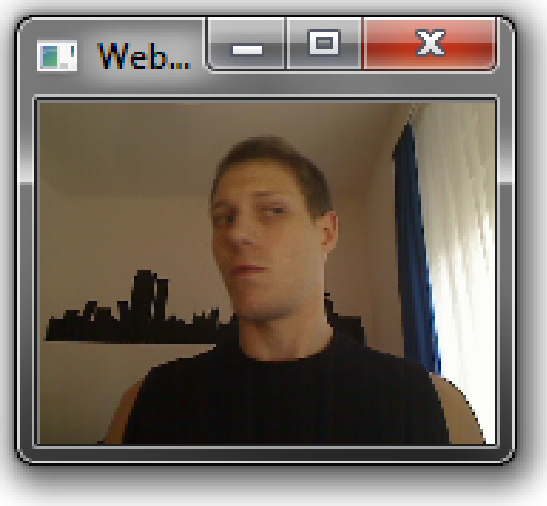

Abbildung 2.4: Ergebnis der Ausführung des Programms von Listing 2.15

2.4.2.1 Video Capturing

Mit *openCV* ist die Bildaufnahme (engl. Video Capturing) mit einem Bildaufnahmegerät wie zum Beispiel einer Webcam für einen Programmierer, der die Open Source Computer Vision Library verwendet, sehr einfach.
Listing 2.15 zeigt ein kleines Codebeispiel dazu. Das Programm erzeugt ein Fenster, welches ein Bild der Webcam anzeigt (siehe Abbildung 2.4).

Offensichtlich ist es sehr einfach mit *openCV* das Bild einer Webcam anzeigen zu lassen. Wenige Codezeilen waren bereits ausreichend.

```
*cap >> gatherimg;
```

Um den Videostream einer Webcam anzeigen zu lassen muss regelmäßig (abhängig von der Framerate) das neuste Bild des *cv::VideoCapture*-Objekts eingelesen werden.

2.4.2.2 Bildbearbeitung und Zeichnen auf Bildern

Mit *openCV* ist das Bearbeiten von Bildern mit nur wenigen Programmzeilen zu bewerkstelligen. Viele Funktionen für Transformationen und Filterung sorgen für eine einfache und schnelle Möglichkeit Bilder zu bearbeiten. Auch das Zeichnen einfacher Formen auf ein Bild ist mit *openCV* möglich. Für diese Arbeit werden nur wenige Funktionen benötigt werden.

cv::resize Eine der eben erwähnten Funktionen ist die *cv::resize()* Funktion. Sie ermöglicht ein Bild in der Größe zu verändern. Dazu wieder ein einfaches Codebeispiel:

```
1 // Benutze das VideoCapture Objekt um einen
2 // Frame aus dem VideoDevice zu laden
3 *cap >> gatherimg;
4
5 // Erzeuge ein neues cv::Mat Objekt mit einer anderen Gre
6 cv::Mat* resized_img = new cv::Mat(120, 160, gatherimg.type());
7
8 // Benutze die cv::resize Methode um
9 cv::resize(gatherimg, resized_img, out_img->size(), 1, 1, 1);
```

Listing 2.16: C++ *openCV cv::resize()*-Funktion Beispiel

Das Codebeispiel aus Listing 2.16 zeigt, wie ein Bild in der Größe auf 160 mal 120 Pixel skaliert und in einem anderen Bildobjekt gespeichert wird. Die ersten zwei Parameter der *cv::resize()*-Funktion sind das Input- und das Output-Bild. Der dritte Parameter gibt die Größe des Output-Bildes an. Die folgenden zwei Parameter geben den Skalierungsfaktor in die x- und y-Richtung an. Diese Parameter sind nicht von Belang, wenn der Dritte Parameter ungleich 0 ist. Der Letzte Parameter gibt die Art der Interpolation an, die bei der Größenveränderung angewandt wird.
Hier gibt es folgende Interpolationsvarianten (Auszug aus der openCV Dokumentation [tea12b] auf Englisch):

- **INTER _ NEAREST** nearest-neighbor interpolation
- **INTER _ LINEAR** bilinear interpolation (used by default)
- **INTER _ AREA** resampling using pixel area relation
- **INTER _ CUBIC** bicubic interpolation over 4x4 pixel neighborhood
- **INTER _ LANCZOS4** Lanczos interpolation over 8x8 pixel neighborhood

cvRectangle und cvCircle Um ein Viereck (engl. rectangle) oder einen Kreis auf ein Bild zu zeichnen, bedarf es ebenfalls nur jeweils einer Funktion. Wie diese beiden Funktionen zu benutzen sind sollte aus dem Beispiel in Listing 2.18 ersichtlich sein. Das Ergebnis sieht dann so aus wie in Abbildung 2.5.

2.4.2.3 Face Detection

Auch für das Erkennen von Gesichtern bietet *openCV* geeignete Tools. Listing 2.17 zeigt einen Codeausschnitt in dem mit *openCV* eine Gesichtserkennung durchgeführt wird.

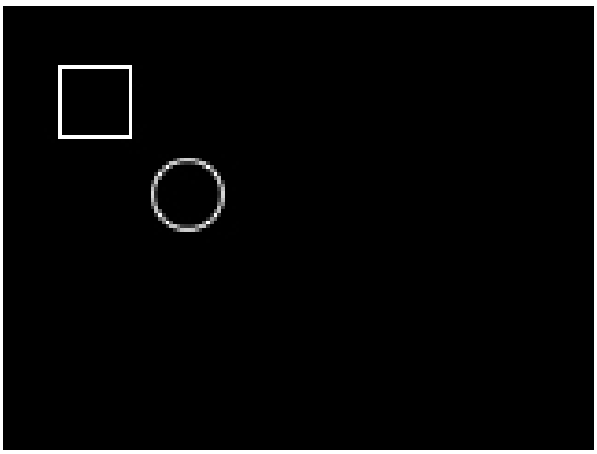

Abbildung 2.5: Ergebnis der Ausführung des Programms von Listing 2.18

```
1  // Benutze das VideoCapture Objekt um einen
2  // Frame aus dem VideoDevice zu laden
3  *cap >> gatherimg;
4
5  // Erstelle eine cv::CascadeClassifier Objekt,
6  mFaceDetector = new cv::CascadeClassifier();
7  // Lade eine Cascade-XML Datei
8  mFaceDetector->load( "cascades/frontalface.xml" );
9
10 // Erstelle ein vector-Objekt, welcher cv::Rect Objekte
11 // halten kann. Dort werden die gefundenen Gesichter
12 // als Viereck abgespeicehrt
13 std::vector< cv::Rect > faceVec;
14
15 // Suche Gesichter auf dem Bild gatherimg
16 mFaceDetector->detectMultiScale( gatherimg, faceVec, 1.0 );
```

Listing 2.17: C++ *openCV* Gesichtserkennung Beispiel

Es gibt verschiedene „*Haar-Classifier-Cascades* “. Diese XML[6] Dateien beinhalten die Information, die ein *cv::CascadeClassifier* Objekt benötigt, um verschiedene Objekte auf einem Bild zu erkennen. Im Falle unseres Beispiels enthält es Gesichtserkennungsinformationen. Diese Codezeile lädt die Datei mit dem Namen und dem Pfad, der im Parameter als String steht:

```
mFaceDetector->load( "cascades/frontalface.xml" );
```

[6]Extensible **M**arkup **L**anguage: Eine Dokumentformat beschreibende Sprache, welche zur Darstellung hierarchisch strukturierter Daten genutzt wird.

Die Funktion *detectMultiScale* sucht auf dem Bild, welches als erster Parameter übergeben wird, nach Objekten, die dem Muster in der zuvor geladenen XML Datei entsprechen. Die Funde werden als *cv::Rect* (als Viereck) in einem *vector* gespeichert. Dieses *vector*-Objekt wird als zweiter Parameter übergeben. Der dritte Parameter gibt an wie stark das Bild verkleinert wird bevor die Suche durchgeführt wird. Ein höherer Wert erhöht die Geschwindigkeit, verringert aber die Genauigkeit. Ergebnisse zur Gesichtserkennung können im **Kapitel 5 – Umsetzung und Implementierung** betrachtet werden.

```
1  // Erstelle cv::Mat Objekt
2  cv::Mat * image;
3
4  //  Flle  Bild mit schwarzem Inhalt
5  ...
6
7  // Zeichnet ein Rechteck. Die linke obere Ecke
8  // ist an der Koordinate (15,16), die rechte
9  // untere Ecke ist an der Koordinate (35,36).
10 // Die Farbe ist wei RGB(255,255,255)
11 cv::Scalar color(255,0,255);
12 cv::Point p1(15,16);
13 cv::Point p2(35,36);
14 cvRectangle(image, p1, p2, color);
15
16 // Zeichne einen Kreis an der Koordinate (50,51)
17 // mit dem Radius 10 und ebenfalls der Fabe wei.
18 cv::Point p3(50,51);
19 cvRectangle(image, p3, color);
```

Listing 2.18: C++ Beispiel zu den *openCV cvRectangle()*- und *openCV cvCircle()*-Funktionen

Zusammenfassung

In diesem Kapitel wurden wichtige Hintergrundinformationen für diese Arbeit erläutert. Die Funktionsweise der Infrarotwärmemessung erklärt, wie die berührungslose Bestimmung von Gegenständen bewerkstelligt werden kann. Die Cognitive Load Theory teilt die Struktur des menschlichen Gedächtnis in verschiedene Teile ein und kann dabei helfen Vorgänge im menschlichen Gehirn zu verstehen. Sie ermöglicht es verschiedene Interaktionen des Menschen mit Lernmaterialien und Nutzeroberflächen besser zu verstehen. Mit der Einführung in C++ und in die openCV Bibliothek erklärte dieses Kapitel auch einige Grundlagen für die Programmierung und Umsetzung.

3 Verwandte Arbeiten

In diesem Kapitel werden für dieses Thema relevante Arbeiten zu verwandten Themen vorgestellt. Diese können dabei helfen sich einen Überblick zu verschaffen. Die verschiedenen Arbeiten zu diesem Thema haben jeweils unterschiedliche Schwerpunkte, von welchen nicht alle ausführlich vorgestellt werden.

Einige dieser Arbeiten beschäftigen sich ebenfalls mit der Messung und Auswertung von Stressfaktoren, haben ihren Fokus aber auf anderen Sensoren und Messverfahren. Dennoch kann daraus wichtiges Wissen gewonnen werden. Zunächst werden zur Einarbeitung Arbeiten zu den Grundlagen behandelt.

3.1 Arbeiten zur Cognitive Load Theory

Wie bereits im **Kapitel 2 – Hintergrund** erwähnt wurde, haben J. Sweller et al. verschiedene Bücher und Artikel zur *Cognitive Load Theory* veröffentlicht. Ein guter Einstieg in das Thema bietet das Buch „Cognitive Load Theory“, welches unter anderem von John Sweller, einem Mitbegründer der *Cognitive Load Theory*, geschrieben wurde [SAK11]. Es bietet einen ausführlicheren Einblick und weiterführende Informationen zu dem im **Abschnitt 2.2 – Cognitive Load Theory** eingeführten Thema. Die Aufteilung der kognitiven Last in extrinsische, intrinsische und lernbezogene Belastung, die in diesem Buch beschrieben wird, kann dabei hilfreich sein, die *CLT* besser bei der Erstellung einer Studie anzuwenden. Die intrinsische, kognitive Belastung wird durch die Schwierigkeit und Komplexität des zu lernenden Stoffs bestimmt. Unter extrinsischer, kognitiver Belastung versteht man die Belastung, die von der Darstellung des Lernmaterials herrührt. Dieser Teil der Belastung ist bei der Anwendung auf Graphical User Interfaces besonders interessant. Auch die lernbezogene Belastung, welche sich auf die Anstrengung der Entwicklung von Denkschemata bezieht, kann hier von Bedeutung sein. Abhängig davon, ob ein Nutzer im Allgemeinen viel mit entsprechenden Programmen arbeitet, erhöht sich die Belastung beim Arbeiten mit der Benutzeroberfläche. Auf verschiedene Methoden zur Bestimmung der kognitiven Last wird in dem Buch „Cognitive Architecture and Instructional Design“ eingegangen [SMP98]. Die einfachste Möglichkeit Stress zu messen ist subjektive Faktoren zu verwenden. Die eigene Wahrnehmung zum Beispiel ist ein subjektiver Faktor. Subjektive Faktoren eignen sich auf Grund ihrer inter- und intraindividuellen Unterschiede nicht gut um kontinuierliche und verlässliche Messungen durchzuführen. Objektive Daten können durch Geschwindikeit und Fehlerrate beim Auführen einer Aufgabe bestimmt werden. Diese Daten so zu interpretieren, dass sich ein verlässlicher Wert für den Stresslevel bestimmen lässt, funktioniert für verschiedene Aufgaben unterschiedlich gut. Eine dritte Möglichkeit sind physiologische Merkmale. Beim Arbeiten

mit der *Cognitive Load Theory* spielen Effekte, die Einfluss auf Wahrnehmung, Konzentration und Vorgehenweise bei Problemlösungen nehmen, eine Rolle. Beide der eben genannten Bücher gehen auf einige dieser Effekte ein.
Aufbauend auf der *Cognitive Load Theory* gab es weitere Arbeiten [Kir02, SG73], auf welche bereits im **Kapitel 2 – Hintergrund** eingegangen wurde.

3.2 Messmethoden zur Bestimmung der kognitiven Last

Siyuan Chen, Julien Epps und Fang Chen veröffentlichten mit „A Comparison of Four Methods for Cognitive Load Measurement“ [CEC11] eine Arbeit über den Vergleich von vier Methoden um geistige Anstrengung zu messen. Das könnte dazu genutzt werden die Schwierigkeit einer Aufgabe am PC anzupassen, oder die Oberfläche einer GUI bezüglich ihrer Einfachheit zu beurteilen. Aufbau und Durchführung der Tests werden weiter unten beschrieben. Eine kleine Gruppe von 20 Testteilnehmern, von welchen fünf auf Grund einer Brille oder zu heller Augenfarbe aussortiert werden mussten, nahmen an einem Experiment teil. Während die Testkandidaten die ihnen gestellten Aufgaben bearbeiteten, wurde versucht, die geistige Anstrengung mit Hilfe der Messdaten der Zeit für das Beenden der Aufgabe, die Korrektheit der Antworten, den Pupillendurchmesser und die Häufigkeit des Blinzelns zu messen. Die Korrektheit der Antworten auszuwerten erwies sich als nur wenig geeignet. Die benötigte Zeit, der Pupillendurchmesser und die Häufigkeit des Blinzelns waren dagegen eher geeignet um die geistige Last zu berechnen. Am ehesten brauchbar waren die Augenmessdaten, da diese ohne Unterbrechung und während der Ausführung der Arbeit erfasst werden konnten. Dazu wurde ein *FaceLAB 4*-Eyetracker mit 60 Hz Taktung verwendet. Das Blinzeln der Augen wurde mit Hilfe einer Webcam und einem MATLAB[1]-Skript bestimmt. Diese Bachelorarbeit beschäftigt sich zwar mit anderen Sensoren, hat aber im Prinzip dasselbe Ziel. Auch hier wird versucht die geistige Last nach dem Vorbild der *Cognitive Load Theory* mit Hilfe verschiedener Sensoren zu messen. Einen weiteren, interessanten Ansatz zum Messen von mentalem Stress schildern Jacqueline Wijsman et al. in der Arbeit „Trapezius Muscle EMG as Predictor of Mental Stress“ [WGPH10]. Mit Hilfe von EMG Messungen am Trapezmuskel wurde versucht den mentalen Stresslevel zu bestimmen. Da Stress das sympathische Nervensystem[2] anregt müsste sich auch der Muskeltonus erhöhen. Daraus lässt sich folgern, dass sich der Stresslevel durch Messungen der Trapezmuskel-Anspannung bestimmen lässt. Für alle Tests wurden EMG Messwerte des oberen Trapezmuskels benutzt. In „Psycho-Physiological Measures for Assessing Cognitive Load“ [HKFD10] verwenden Eija Haapalanien et al. vier verschiedene psycho-physiologische Sensoren. Einer dieser Sensoren war ein kontaktloses Eye-Tracker System, welches aus zwei Kameras und zwei Infrarotblitzern bestand. Nach einer Kalibrierungsphase wurde die Änderungen der Pupillengröße während der Versuche gemessen. Ein EKG-Armband wurde im Bereich des Trizeps des linken Armes angebracht um verschiedene EKG Messwerte zu bestimmen. Desweiteren

[1] MATLAB ist eine Software zum Lösen mathematischer Probleme.
[2] Einer der drei Teile des vegetativen Nervensystems.

wurde ein kabelloses EEG-Headset und ein kabelloser Herzfrequenzmesser eingesetzt. Die besten Ergebnisse wurden mit der Herzfrequenz und mit dem EKG erzielt.

3.2.1 Hauttemperaturmessung zur Stressbestimmung

Die Arbeit von H. Kataoka et al. [KKY+98] beschreibt einen Versuch Stress und Anspannung beim Arbeiten an einem PC mit Hilfe einer Infrarot-Wärmebildkamera zu ermitteln.
In der Einleitung wird das Problem beschrieben, dass Gefühle und Gemütszustände wie Müdigkeit oder Stress nicht direkt messbar sind. Es ist jedoch möglich physikalische Größen zu messen, welche sich mit Gemütszuständen ändern. Das geht, da einige Auswirkungen des Körpers nicht bewusst steuerbar sind, sondern unbewusst vom sympathischen Nervensystem gesteuert werden. Für Stress oder geistige Anstrengung kommt unter anderem die Hauttemperatur in Frage. Die Hauttemperatur eignet sich sogar besonders gut, da mit einer Infrarotkamera diese „non-contact“, also ohne das Anlegen von Sensoren, ermittelt werden kann. H. Katoka et al. zählen einige Vorteile auf, die eine „non-contact “-Messung bietet. Eine eingeschränkte Bewegungsfreiheit der Testkandidaten erschwert das Durchführen der Versuche. Es wird kein zusätzlicher Stress oder Unbequemlichkeit durch anliegende Sensoren verursacht, welche das Testergebnis beeinflussen könnten.
Um das Vorgehen der Datenerhebung und die Berechnung des Stresslevels zu verstehen, muss man wissen, dass im Allgemeinen das Gesicht bei Aufregung heiß und die Fingerspitzen bei Anspannung relativ kalt werden. Anspannung führt zu weniger Durchblutung wovon besonders die Extremitäten betroffen sind. Weniger Durchblutung bedeutet kältere Hauttemperatur. Da die Nase weniger durchblutet ist als das Gesicht wirkt sich die Temperaturveränderung stärker auf die Nase aus als auf den Rest des Gesichts. Der Stresslevel wird also durch den Temperaturunterschied zwischen Nase und Stirn berechnet. Abbildung 3.1 wurde aus dieser Ausarbeitung übernommen um den Mechanismus der für die Veränderung der Hauttemperatur in diesem Zusammenhang zuständig ist deutlicher zu machen.

Der Versuch Die Testkandidaten sollten versuchen einem Ziel auf dem Bildschirm mit Hilfe eines Trackballs zu folgen. Nach 10 Minuten sollte mit geschlossenen Augen 5 Minuten ausgeruht werden. Durch eine Alarmanzeige welche nur durch Eingabe eines Passworts verlassen werden konnte wurde zusätzlicher Stress erzeugt. Abbildung 3.2 zeigt die Veränderung der Nasentemperatur während des Versuchsablaufs.

Es ist deutlich zu sehen, dass die Nase während der Ruhephasen wärmer und bei Stress kälter wird. Besonders deutlich wird das bei dem Knick, den die Kurve nach unten macht, wenn die „emergent condition“, also die zusätzlich stresserzeugende Situation, auftritt.

Gesichtsbestimmung Um die Temperatur an Nase und Stirn messen zu können, müssen mit einer visuellen Kamera die Stirn und die Nase erkannt werden. Wenn die visuelle Kamera und die Infrarotwärmebildkamera im gleichen Winkel zur Testperson stehen, vereinfacht das die Bildverarbeitung enorm. Daher benutzten H. Kataoka et al. einen Spiegel, welcher Licht im Infrarotbereich durchlässt, visuelles Licht aber spiegelt. (Siehe dazu Abbildung 3.3)

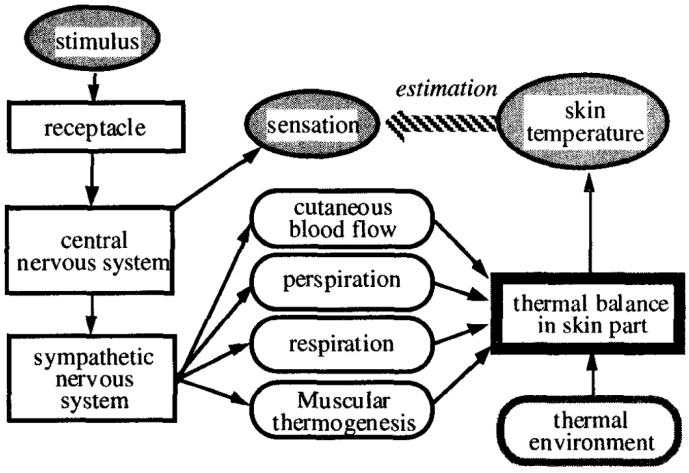

Abbildung 3.1: Diagramm des Mechanismus der Veränderung der Hauttemperatur [KKY+98]

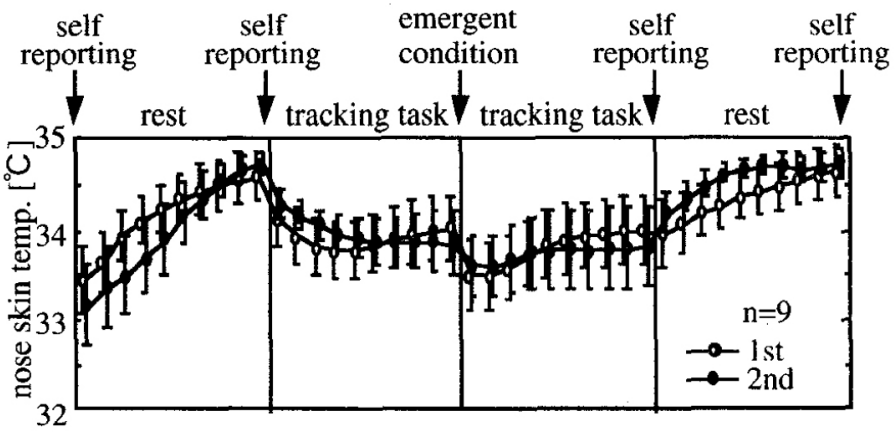

Abbildung 3.2: Temperaturänderung der Nase während des Versuchs [KKY+98]

Die Gesichtsbestimmung funktionierte dann so, dass zunächst das Wärmebild binarisiert wurde, sodass die gesamte Person in einer Farbe und den Hintergrund in einer anderen Farbe auf dem Bild zu sehen. Um die Gesichtsbestimmung einfach zu halten wurde innerhalb des aus dem binarisierten Bild bekannten Gesichtsbereich nach Augenbrauen und Lippen gesucht. Die restlichen Gesichtsbereiche wurden abhängig von den Gesichtsgrenzen und der drei ermittelten Punkte bestimmt. Das Auslesen der Temperatur ist dann sehr einfach, da das visuelle Bild und das Wärmebild auf Grund des Aufbaus direkt übereinandergelegt werden können.

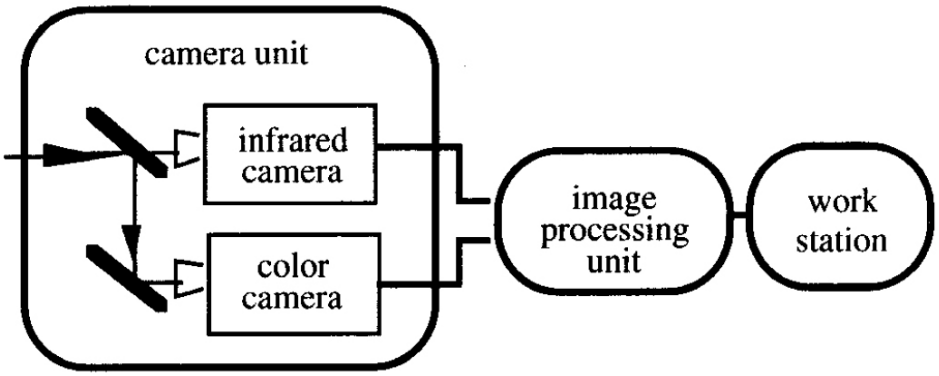

Abbildung 3.3: Aufbau der visuellen Kamera und der Infrarotwärmebildkamera [KKY+98]

Errechnung des Stressfaktors Der Stressfaktor wurde aus dem Temperaturunterschied zwischen Stirn (Th) und Nase (Tn) errechnet. Der Maximale Temperaturunterschied betrug höchstens ca. 2.98 Grad. Eine Formel, welche den Stressfaktor (S) auf einer Skala von 0 bis 100 anzeigt, ist die folgende:

$$S = 33.59 * (Th - Tn)$$

Folgerung Hier entstand ein interessanter Versuchsaufbau, mit welchem es möglich ist den Stresslevel auf einer Skala von 0 bis 100 zu berechnen. Dabei musste keiner Person ein Sensor angelegt werden. Natürlich handelt es sich hier nur um einen Prototyp. Genauigkeit und Einfachheit der Anwendung müssen weiter verbessert werden. Jedoch zeigt es, dass die Stressmessung mit Hilfe einer Infrarotwärmebildkamera ein vielversprechender Ansatz ist.

3.3 User Studies und Testaufbau zur Evaluierung der Messmethoden

Siyuan Chen, Julien Epps und Fang Chen setzten in ihrer Arbeit „A Comparison of Four Methods for Cognitive Load Measurement“ [CEC11] auf eine Testumgebung, bei welcher die Teilnehmer mathematische Aufgaben zu lösen hatten. Dabei mussten sie vier nacheinander erscheinende Zahlen aufsummieren und die richtige Antwort mit der Maus aus zehn möglichen, angebotenen Antworten auswählen. Anschließend musste die subjektiv wahrgenommene Schwierigkeit der eben gelösten Aufgabe angegeben werden. Nach einem kurzen Training musste jeder Testteilnehmer sieben Aufgabensätze durchführen, jeweils mit einer kurzen Pausenunterbrechung. Jeder dieser Sätze bestand aus zehn Aufgaben, wobei jeweils zwei Aufgaben aus jeder Schwierigkeitsstufe (1-5) in zufälliger Reihenfolge enthalten

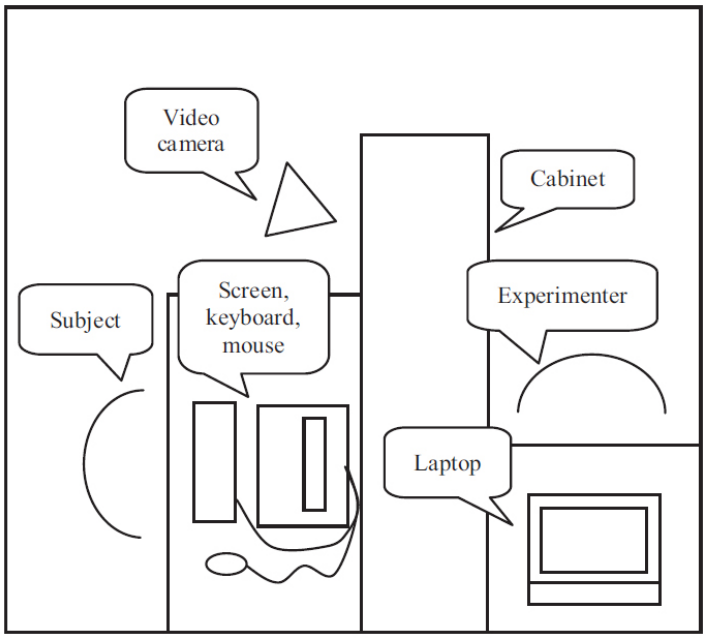

Abbildung 3.4: Skizze der Raumaufteilung für Experimentdurchführung [WGPH10]

waren. Der Schwierigkeitsgrad richtete sich dabei nach der Anzahl der Ziffern und der Überträge.

Ein Ziel der Arbeit „Trapezius Muscle EMG as Predictor of Mental Stress“ [WGPH10] war ein neues Protokoll zur Stressinduktion vorzuschlagen und zu validieren. 30 Testprobanden wurden für dieses Experiment rekrutiert, wobei davon 25 Männer und 5 Frauen waren. Personen mit Muskel- oder Herzleiden wurden ausgeschlossen. Die Testpersonen waren zwischen 19 und 53 Jahre alt. Wichtig war auch, dass alle Probanden Rechtshänder waren, da, um nur den linken Trapezmuskel messen zu müssen, während der Experimente die rechte Hand zur Durchführung benutzt werden sollte. Der Test wurde in einem ruhigen Raum an einem PC durchgeführt. Abbildung 3.4 zeigt eine Skizze der Raumaufteilung. Als Eingabegeräte wurden eine Maus und eine Tastatur zur Verfügung gestellt. Töne wurden durch einen Kopfhörer abgespielt und eine im Raum installierte Videokamera suggerierte, dass die gesamte Prozedur aufgenommen würde. Zum Ablauf des experimentellen Protokolls wurde zunächst der „Perceived Stress Scale“ (kurz *PSS*) ermittelt und es wurden ein paar allgemeine Fragen gestellt. Der *PSS* gibt an, wie stark jemand Stress wahrnimmt, oder als wie stark jemand den Stress einer Situation bezeichnen würde. Nachdem durch ein paar Tests Stress-Referenzwerte ermittelt wurden, sollten die Probanden anschließend drei verschiedene Tests durchlaufen. Die Reihenfolge der Tests wurde zufällig gewählt, um die Beeinflussungen durch den Übergang zwischen verschiedenen Tests zu minimieren. Abwechselnd mit den Stress-Phasen jedes Tests wurden Ruhephasen eingeplant. Jeweils nach einer Stress- oder Ruhe-Phase sollten die Testpersonen ihren subjektiven Stressgrad und ihre Stimmung durch einen auf Fragen basierten Self-Report angeben. Abbildung 3.5

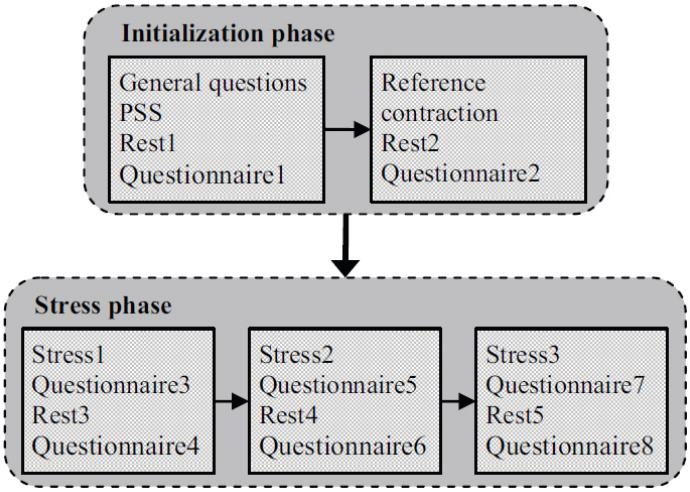

Abbildung 3.5: Skizze des Ablaufs der Testprozedur [WGPH10]

zeigt den schematischen Aufbau dieser Prozedur. Beim ersten Test handelte es sich um den „Norinder Test“[3][LHB82]. Dabei handelt es sich um einen auf Berechnungen basierenden Test. In 2:30 Minuten mussten 27 Berechnungen durchgeführt werden. Ein roter Bildschirm und ein Buzzer-Sound wiesen auf Fehler hin, um noch mehr Stress zu verursachen. Der zweite Test wurde erstellt um mentalen Stress zu verursachen. Er bestand aus 5 verschiedenen mental anstrengenden Aufgaben, wie zum Beispiel logische Puzzles. Der letzte Test dagegen sollte psychologischen und mentalen Stress verursachen. Nach dem zweiten Test wurde dem Testteilnehmer gesagt, er könne etwas gewinnen, wenn er diesen Test schnell genug abschließe. Nach besagtem Test wurde ihm mitgeteilt, dass er den Test ausreichend schnell abgeschlossen habe, er diesen Gewinn aber im Falle einer schlechten Leistung beim dritten Test wieder verlieren könne. Das erzeugte zusätzlichen Stress. Die Auswertung der 30 Testdatensätze, von welchen 8 im Voraus aussortiert werden mussten, ergab, dass die Amplitude des EMG Signals in Stresssituationen deutlich höher war. Das lässt darauf schließen, dass die EMG Signale tatsächlich von der Aktivität des sympathischen Nervensystem und somit auch von der Höhe des Stresslevels abhängen.
Eija Haapalanien et al. versuchten in dieser Arbeit [HKFD10] mit Hilfe verschiedener Sensoren und Messmethoden die kognitive Last zu ermitteln. Dabei entstand ein Testprotokoll mit verschiedenen Aufgaben für Testpersonen. Eija Haapalainen et al. beschäftigten sich ausführlich mit der *Cognitive Load Theory* und passten den Aufbau der Versuche gut auf deren Erkenntnisse an. Diese verschiedenen Versuchsarten können beim Entwickeln eines Konzepts für einen guten Versuchsaufbau helfen. Sechs verschiedene Aufgaben galt es zu lösen. Für jeden Test wurden maximal drei Minuten Zeit gewährt. Diese sechs Tests decken verschiedene Denkmuster und -Arten des Gehirns ab.

[3] Norinder Test

Gestalt Completion Test Bei diesem Test müssen unfertig gezeichnete Objekte oder Muster erkannt werden. Es darum schnell Zusammenhänge festzustellen, zu kombinieren und Informationen zu ordnen.

Hidden Pattern Test Hier muss aus einer Reihe von Bildern und Formen das zuvor gezeigte Bild oder die Form wiedererkannt werden oder es ist festzustellen, dass das gesuchte Objekt nicht vorhanden ist.

Finding A's Test Hier geht es um die Wahrnehmungsgeschwindigkeit. Die Häufigkeit des Auftretens des Buchstaben „A“ musste in verschiedenen Wörtern gefunden werden. Die Länge der Wörter bestimmt dabei den Schwierigkeitsgrad der Aufgabe.

Number Comparison Test Die Wahrnehmungsgeschwindigkeit wird bei diesem Test daran gemessen wie schnell die Testperson die Größe zweier Zahlen vergleicht.

Pursuit Test Mehrere Linien führen von jeweils einer Nummer auf der linken Seite des Bildschirms zu einem Buchstaben auf der rechten Seite. Jede Nummer musste einem Buchstaben zugeordnet werden. Die Schwierigkeit dabei war, dass sich diese Linien mehrfach überschneiden.

Scattered X's Test Auf dem Bildschirm verteilen sich viele Buchstaben. Die Aufgabe war es die Anzahl der *X*e zu zählen.

Zusammenfassung

In diesem Kapitel wurden verschiedene für diese Ausarbeitung relevante Arbeiten vorgestellt. Ähnliche Anwendungen der Cognitive Load Theory sowie das Arbeiten mit Infrarotwärmebildkameras zur Hauttemperaturbestimmung werden gezeigt. Zur Erstellung einer Nutzerstudie im Zusammenhang mit der Bestimmung der kognitiven Last beim Arbeiten mit dem Computer wurden ebenfalls Arbeiten behandelt.

4 Das Konzept

Vor der Beschreibung der Umsetzung wird in diesem Kapitel das zugehörige Konzept vorgestellt. Ein gutes Konzept ist eine Notwendigkeit und Voraussetzung für eine erfolgreiche und zufriedenstellende Umsetzung

Erklärt wird hier die Planung und die Beschreibung der Idee von der Sensoranbindung über die Datensammlung und -Auswertung bis hin zum fertigen Ergebnis. Zusätzlich werden Ansätze einer Nutzerstudie, welche zur Evaluierung dienen kann, entworfen. Es geht darum ein Programm zu entwickeln, welches die gewünschten Sensoren anbindet und auswertet. Der Stresslevel einer Person soll mit Hilfe dieser Sensoren ermittelt und angezeigt werden können. Dieser Wert kann dann dazu genutzt werden Anpassungen an dem momentan genutzten Programm durchzuführen.

4.1 Die Sensoren, Datenerhebung

In dieser Arbeit liegt der Fokus darauf die kognitive Last eines Nutzers per Wärmebildkamera zu ermitteln. Die Gründe dafür liegen auf der Hand. Es müssen keine Messinstrumente angelegt werden und die ermittelten Daten sind objektiv, da die Hauttemperatur nicht bewusst beeinflusst werden kann. Bei genaueren Überlegungen stellt sich die Frage, wie genau ein Computer die Informationen der Hauttemperatur zu einem Wert, welcher den Stresslevel angibt, umrechnen kann. Ein Versuchsaufbau könnte die Hauttemperatur von Testpersonen, welche durch bestimmte Aufgabenstellungen kognitiver Last ausgesetzt werden, messen. Diese Daten können dann im Bezug auf Muster und Gesetzmäßigkeiten der Hauttemperatur im Zusammenhang mit der kognitiven Last analysiert werden. Andererseits bestehen bereits Ansätze, wie mit Hilfe der Hauttemperatur des Gesichts auf die kognitive Belastung geschlossen werden kann. Im **Kapitel 3 – Verwandte Arbeiten** wurde bereits erwähnt, dass der Vergleich von Nasen- und Strintemperatur gute Ergebnisse erzielt. Dieser Berechnungsansatz kann als Referenz dienen um bei einem Versuch weitere vielversprechende Berechnungsmöglichkeiten zu ermitteln.
Beim Weiterverfolgen dieses Ansatzes wird zusätzlich eine visuelle Kamera benötigt um die verschiedenen Gesichtsbereiche bestimmen zu können. Das erste Ziel wird es sein die beiden Kameras an den Computer anzubinden, sodass jeweils auf die Daten zugegriffen werden kann.

4.2 Möglichkeiten das visuelle Bild und das Wärmebild anzugleichen

Mit Hilfe des visuellen Bildes sollen die Position der Stirn, Nase und weitere Gesichtsbereiche einer Person bestimmt werden. Anschließend können aus den Wärmedaten die Temperaturen der beiden Gesichtsbereiche ermittelt und angezeigt werden. Aus diesen Daten kann die kognitive Last berechnet und als Wert auf einer Skala von 0 bis 100 angezeigt werden. Voraussetzung für diese Berechnungen ist, dass aus Positionsdaten des visuellen Bildes auf Positionierungen des Wärmebildes geschlossen werden kann. Wie in der Arbeit „Development of a skin temperature measuring system for non-contact stress evaluation“ [KKY+98] zu sehen war (vgl. Abbildung 3.3), vereinfacht es das Arbeiten mit dem visuellen Bild und dem Wärmebild ungemein, wenn die Kameras im selbem Winkel zueinander aufgebaut sind und an der theoretisch selben Stelle stehen (evtl. abgelenkt durch Spiegel). Möglich wäre auch eine Verschiebung der Kameras herauszurechnen. Dafür müssten aber Informationen über die Platzierung beider Kameras vorhanden sein. Die relative Position der Kameras könnte durch eine Konstruktion fixiert werden. Diese Lösung wäre jedoch wenig flexibel und würde die Nutzbarkeit des Programms auf diese eine Kamerakombination beschränken.
Es gibt noch eine weitere, sehr aufwändige Methode. Es ist möglich ein visuelles Bild und ein Wärmebild so zu „matchen“, dass sie passend übereinander gelegt werden können. Aber auch das wäre mit einem unverhältnismäßig hohen Aufwand verbunden und vermutlich schwieriger umzusetzen als eine eventuelle Gesichtserkennung direkt im Wärmebild.

4.3 Datensammlung und -Verarbeitung

Als nächstes gilt es die Frage zu klären, welche Bildwiederholrate benötigt wird. Wie oft der Stresslevel pro Zeiteinheit berechnet werden soll spielt hierbei eine Rolle. Bei zu hohen Frequenzen steigt jedoch die Rechenlast enorm, da für jedes Bild die Kameraverschiebung berücksichtigt und die Gesichtsbereiche bestimmt werden müssen. Ist die Bildwiederholrate dagegen zu niedrig sind vermutlich nicht genügend Werte pro Zeiteinheit vorhanden und das „Video“ läuft für das menschliche Auge nicht mehr flüssig.
Um keine Informationen auf Grund zu klein gewählter Datentypen zu verlieren sollte darüber nachgedacht werden, wie die ermittelten Informationen und Bilder der Kameras als Daten repräsentiert werden. Die Repräsentation des Wärmebilds und der Wärmeinformation hängt von den Daten ab, welche die Wärmebildkamera liefert. Vermutlich ist es nötig eine eigene Klasse dafür zu schreiben, vielleicht reicht aber auch ein RGB-Bild aus.

4.3.1 Gesichtserkennung

Sind die beiden Bilder eingelesen und die nötigen Berechnungen durchgeführt, müssen die Positionen der Gesichtsbereiche ermittelt werden. Um das Rad hier nicht neu erfinden zu

müssen bietet es sich an eine Bibliothek zu verwenden, welche bereits Tools für die Gesichtserkennung bereitstellt. Im **Kapitel 2 – Hintergrund** wurde bereits eine solche Bibliothek angesprochen. Die Gesichtserkennung wird auf dem visuellen Bild ausgeführt. Anschließend kann der Wert eines entsprechenden Pixels des Wärmebilds ausgelesen werden.

4.3.2 Bestimmung der kognitiven Last

Wie H. Kataoka et al. die geistige Last berechnen wurde bereits im **Kapitel 3 – Verwandte Arbeiten** beim Betrachten der Arbeit „Development of a skin temperature measuring system for non-contact stress evaluation“ [KKY+98] gezeigt. Um einen Wert für die kognitive Last, vorzugsweise als Wert von 0 bis 100, zu erhalten müssten nur die Temperaturen zweier (oder mehrerer) Gesichtsbereiche verglichen werden. Der Prototyp einer solchen Formel ist schnell erstellt. Doch einige Tests müssten zeigen, wie zuverlässig der errechnete Wert ist. Durch genügend Test- und Formelmodifikationsiterationen könnte eine ausreichend gute Genauigkeit erzielt werden.

4.4 Tests und Prüfung der Konstruktion

Die gesamte Arbeit bis zu diesem Zeitpunkt muss detailiert geprüft und getestet werden. Zuverlässigkeit, Genauigkeit und Einfachheit in der Handhabung sollten oberstes Ziel sein. Wie eben schon erwähnt hängt die Genauigkeit des Stresslevels von der Formel und Berechnungsart ab. Aber auch die Genauigkeit der Sensordaten der Bildangleichung spielen hier eine Rolle. Es stellt sich die Frage, ob hier einige Selbsttests ausreichen oder ob mehrere Testpersonen benötigt werden. Diese könnten durch das Bearbeiten bestimmter Aufgaben künstlich Stress ausgesetzt werden. Die dabei ermittelten Daten können helfen verschiedene Faktoren der Genauigkeit zu verbessern.
Die Einfachheit der Handhabung zu verbessern ist sinnvoll. Zu bedenken ist hier aber, dass kein Produkt für eine Heimanwendung entstehen soll, sondern ein Testmodul für akademische Zwecke. Eine gewisse Einarbeitungszeit und Grundverständnis können hier vorausgesetzt werden. Dennoch sollte die GUI intuitiv bedienbar und einfach erlernbar sein. Nach einigen Iterationszyklen sollte die Konstruktion eine zufriedenstellende Qualität bezüglich der angesprochenen Anforderungen aufweisen.

4.5 Anwendungszenarien

Für die automatische Stresslevelermittlung gibt es viele Anwendungsmöglichkeiten. Bei den folgenden Szenarien ist die Nutzung des Stresswerstes an eine bestimmte Applikation oder an ein Gerät gebunden. Der Grund dafür ist, dass die beiden Kameras fest montiert sind und nur den Stresslevel von Personen in einem fixen Bildbereich messen können.

Automatische Anpassung der GUI Man stelle sich eine Applikation mit vielen Menüs und Shortcuts für den Computer vor. Ein Amateuranwender, der bisher noch nicht mit diesem Programm in Berührung gekommen ist, beginne damit seine Arbeit zu tätigen. Das Programm sei eher für Profianwender ausgelegt. Mit Hilfe der automatischen Bestimmung des *Cognitive Workloads* könnte festgestellt werden, dass das Programm wenig für den Nutzer geeignet ist und als Reaktion darauf die grafische Benutzeroberfläche anpassen. Auf einer Menüebene werden weniger Elemente dargestellt und gegebenenfalls dafür die Schachtelung erhöht. Zusätzliche Hilfetexte, ein angebotener Programmguide und ausführlichere *Mouse-Hover*-Nachrichten können die Bedienung weiter vereinfachen.

Lernanwendung Eine weitere Anwendungsmöglichkeit wäre der Einsatz bei einer Lernanwendung. Der aktuelle Lernfortschritt könnte anhand des Workloads bestimmt und so entschieden werden, ob mit einer schwierigeren Übung fortgesetzt werden kann.

Anpassung einer Computer-KI Theoretisch könnte diese Technologie auch bei Computerspielen Anwendung finden. Die Stärke der KI-Gegner könnte sich so besser automatisch an das Niveau des Spielers anpassen.

Leseunterstützung Ein Programm zur Textanzeige könnte bei hoher Last des Lesers zum aktuellen Artikel zusätzlich Informationen oder Definitionen und Erklärungen anbieten.

4.6 Planung einer Nutzerstudie

Eine Nutzerstudie könnte zeigen wie groß der Nutzen ist, der aus einer solchen Umsetzung gezogen werden kann. Für eine solche Studie sollten mindestens 20 Testpersonen verschiedenen Geschlechts herangezogen werden. Einige mathematische Aufgaben nach den Mustern des Experiments aus „Psycho-physiological measures for assessing cognitive load“ [HKFD10] könnten einem Testteilnehmer beliebig schwere Aufgaben verschiedenster Art stellen. So könnte die kognitive Last durch Variation des Aufgabenlevels konstant auf ca. 50 gehalten werden. Als Vergleichswerte würden die Bearbeitungszeit der Aufgaben und die Selbsteinschätzung der Testpersonen dienen. Da die Variation des Schwierigkeitsgrades durch Modifikation des Aufgabenlevels geschieht sollte auch bei diesem Versuch der Großteil der Belastung durch die Bearbeitung der Aufgaben und nicht durch die Bedienung des Computers oder anderen Störquellen hervorgerufen werden. Deshalb sollten alle Testpersonen erfahrene Computerbenutzer sein. Die Fähigkeit mathematische Aufgaben zu lösen sollte jedoch unterschiedlich ausgeprägt sein um die Aussagekraft der Studie zu wahren.

4.7 Mögliche Erweiterungen

Möglich wäre es ein Konzept für ein System zu entwickeln, welches verschiedene Visualisierungen und GUIs darstellen kann. Die Reaktionen des Anwenders auf die GUI können gemessen und basierend darauf Anpassungen vorgenommen werden. Das kann jedoch erst nach Auswertung der Nutzerstudie geschehen. Auch die Implementierung eines funktionalen Prototyps mit automatischer Adaption an der Nutzer benötigt die Daten der ausgewerteten Studie.

Zusammenfassung

In diesem Kapitel wurden die Sensoren vorgestellt, welche nach dem entwickelten Konzept voraussichtlich verwendet werden. Mögliche Probleme, die beim Angleichen des visuellen Bilds und des Wärmebilds auftreten können, wurden diskutiert. Der geplante Prozess der Datensammlung und -Verarbeitung, die Gesichtserkennung eingeschlossen, wurden erläutert. Dieses Kapitel zeigte auch verschiedene Anwendungsszenarien auf und erklärte die Planung einer Nutzerstudie.

5 Umsetzung und Implementierung

Nachdem das Konzept der Umsetzungsplanung behandelt wurde folgt die Ausführung der Umsetzung. Wie schon erwähnt stellt das Konzept nur den Optimalfall dar. Das Endresultat hängt von vielen Faktoren ab. Beispielsweise könnte während der Implementierung festgestellt werden, dass eine Umsetzung, wie sie im Konzept geplant wurde, technisch nicht möglich ist oder unverhältnismäßig viel Zeit in Anspruch nimmt.

Die benutzte Hardware wird hier vorgestellt und die Wahl genutzter Programmiersprachen und Bibliotheken erklärt. Einzelne Codeausschnitte verdeutlichen die Grundfunktionen des bei dieser Arbeit entstandenen Programms.

5.1 Die Hardware

Zuerst wird die Hardware, die hier zum Einsatz kommt, untersucht. Die Eigenschaften der Hardware haben einen sehr großen Einfluss auf das zu entwickelnde Computerprogramm, die Sensoranbindung, das Sammeln der Daten und die Auswertung. Verschiedene Hardware liefert verschiedene Daten oder entsprechende Daten in unterschiedlichen Formaten.

Wärmebildkamera Als Wärmebildkamera wurde eine Infrarotkamera von ©*optris GmbH* verwendet: **optris PI160**. Diese Kamera wird auf der Website der *optris GmbH* [Gmb12] als Einsteigermodell der *optris* Online-Infrarotkameras bezeichnet.
Die Kamera ist dank ihrer geringen Größe (siehe Abbildung 5.1 und Abbildung 5.2) sehr Mobil und kann einfach per USB an den Computer angeschlossen werden.

Auch für eine stationäre Anwendung eignet sich diese Kamera, da eine Halterung zur festen Montage und Justierung als Zubehör für Kameras dieser Baureihe erworben werden kann. Für diese Arbeit war jedoch nur das standardmäßig mitgelieferte Standbein vorhanden. Die verschiedenen Objektive machen es möglich den Bildausschnitt auf Objekte in verschiedenen Entfernungen anzupassen. Leider konnte nur mit dem Standardobjektiv (23°C x 17°C) gearbeitet werden, da keine weiteren zur Verfügung standen. Da aber alle Objektive einen anderen Bildwinkel als die zur Aufnahme der visuellen Bilder genutzten Kamera haben wäre der zusätzliche Nutzen eines anderen Objektivs eher gering.

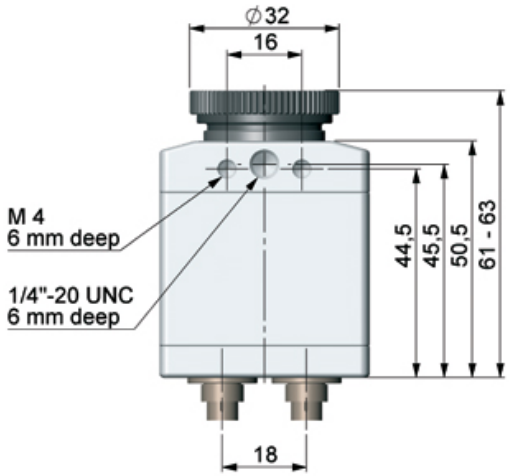

Abbildung 5.1: Maße der optris PI160 Wärmebildkamera von oben

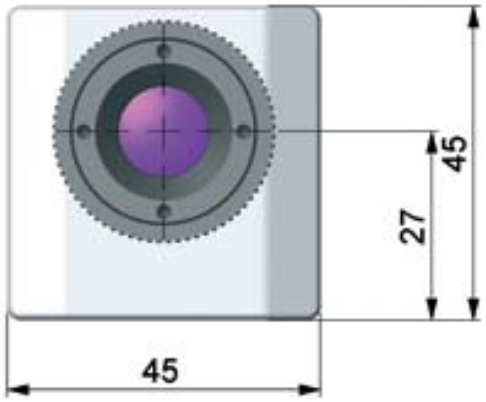

Abbildung 5.2: Maße der optris PI160 Wärmebildkamera in der Frontansicht

Die wichtigsten Informationen und Merkmale aus dem Datenblatt:

- Optische Auflösung: 160x120 Pixel
- Temperaturbereich von $-20°C$ bis $900°C$
- Spektralbereich 7,5 bis 13 μm
- 120Hz Bildwiederholfrequenz

Mit diesen Spezifikationen ist diese Infrarotwärmebildkamera (laut *optris*) hervorragend für „viele Anwendungen in der Prozessautomation, an Teststationen sowie in der F&E geeignet.“

Abbildung 5.3: Logitech QuickCam Pro 9000

Wie gut sich diese Kamera in diesem Fall einsetzen lässt, müssen Tests nach Fertigstellung der Testapplikation zeigen.

Die visuelle Kamera Eine **Logitech OEM QuickCam Pro 9000** wurde als visuelle Kamera bereitgestellt. Diese Webcam wird über USB an den Computer angeschlossen und hat eine Videoauflösung von 1600x1200 Pixeln. Damit hat sie eine 10-fach höhere Auflösung als die Wärmebildkamera. Bei einer Bildwiederholrate von 30Hz liefert sie zwar nicht so viele Bilder pro Sekunde wie die Wärmebildkamera, für die vorliegende Aufgabe jedoch genug. Abbildung 5.3 zeigt die *QuickCam Pro 9000*.

Die *QuickCam Pro 9000* verfügt über einige Eigenschaften für die Heim- und *InstantMessaging*-Anwendung [Log12]. Relevant sind für diese Arbeit hauptsächlich folgende Eigenschaften:

- Autofokus
- 2-Megapixe-HD-Sensor

Die Kamera eignet sich also gut für die Aufnahme von Gesichtern aus naher Distanz. Auf die Autofokus-Funktion muss leider verzichtet werden, da diese nur mit Hilfe des entsprechenden Treibers und der mitgelieferten Software genutzt werden kann. Durch das Auslesen der Daten mit *openCV* entfällt also die Möglichkeit den Autofokus zu nutzen.

Der Computer Als Rechner wurde ein **Acer Aspire 7750G-2434G50Mnkk** verwendet. Ein Intel® Core™ i5-2430M mit 2x2,4GHz, 4GB DDR3 RAM und eine AMD Radeon™ HD 6850M sollten genügend Rechenleistung für diese Aufgabe aufbringen können.

5.2 Umsetzung in C++

Nachdem ein erster Versuch die Kameras mit einem Java-Programm auszulesen fehl schlug, lag die Entscheidung, das Problem mit C++ zu lösen nicht fern. Auf der mitgelieferten CD

der *optris PI160* befinden sich einige Beispielprogramme, die in C++ geschrieben wurden. Diese Programme zeigen den Stream der Wärmebildkamera an. Eines dieser Programme bietet eine gute Basis um die Entwicklung einer entsprechenden Software zu beginnen. Folgende C++ Beispielprogramme befanden sich auf der CD:

- IPC2
- IPC2 VisCam
- MultiIPC2

Das Programm *IPC2* zeigt den Videostream der *optris* Infrarotwärmebildkamera und einige weitere Daten. *IPC2 VisCam* kann sowohl den Wärmebildvideostream als auch einen visuellen Videobildstream anzeigen. Dazu wird jedoch eine *optris PI200* benötigt. Die *optris PI160* hingegen besitzt keine integrierte visuelle Kamera. Ein integriertes visuelles Kamerabild wie es die PI200 liefert würde, die Arbeit extrem vereinfachen, da die visuelle Kamera der PI200 exakt den gleichen Bildbereich abbildet wie die Wärmebildkamera. Dabei würde jede Anstrengung, die verschiedenen Kamerabilder aufeinander abzustimmen, entfallen. Auch die Anbindung wäre bedeutend einfacher, da das *IPC2 VisCam* Programm alle nötigen Funktionen zum Auslesen beider Kamerainformationen unterstützt. *MultiIPC2* bietet die Möglichkeit mehrere *optris* Wärmebildkameras parallel auszulesen und den Videostream anzeigen zu lassen.

5.2.0.1 Zusammenhang zwischen ImagerIPC.dll und Imager.exe

Neben den Beispielprogrammen liegt der Kamera auch das Hauptprogramm bei: *Imager.exe*. Abbildung 5.4 zeigt einen Screenshot des *Imager.exe* Programms.

Es zeigt das Wärmebild, Temperaturwerte und einige weitere Informationen und zeigt diese in verschiedenen Diagrammen an.
Für diese Wärmebildkamera gibt es keine Dokumentation, die das direkte Auslesen der Daten beschreibt. Die *optris* Beispielprogramme greifen alle auf die *ImagerIPC.dll* zu um auf den Kamerastream zuzugreifen. *ImagerIPC.dll* ist eine Dynamic Link Library von optris. Sie stellt verschiedene Funktionen zur *Inter Process Communication* (IPC) mit dem *Imager.exe* Programm bereit. Also muss das Programm zusätzlich ausgeführt werden und eine Verbindung mit der Wärmebildkamera muss bestehen, sodass mit Hilfe der DLL der Kamerastream abgerufen werden kann (Siehe Abbildung 5.5). Für den direkten Zugriff auf die Kamera ist diese DLL nicht geeignet und *optris* stellt dafür auch keine bereit.

5.2.1 IPC VisCam als Basis

Als Basis für das Computerprogramm wurde das *IPC2 VisCam* Beispielprogramm von *optris* verwendet. Auf Abbildung 5.6 ist die Programmoberfläche zu sehen. Links wird das Bild der Wärmebildkamera angezeigt. Auf der rechten Seite würde bei der Verwendung der *optris*

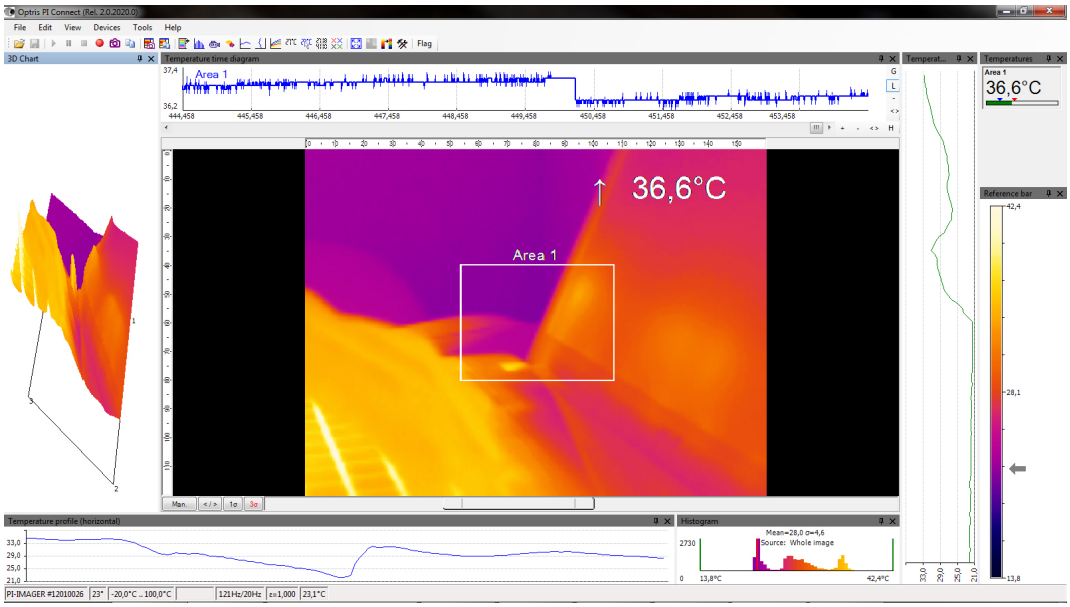

Abbildung 5.4: Screenshot des Imager.exe Programms zum Auslesen der Wärmebilddaten

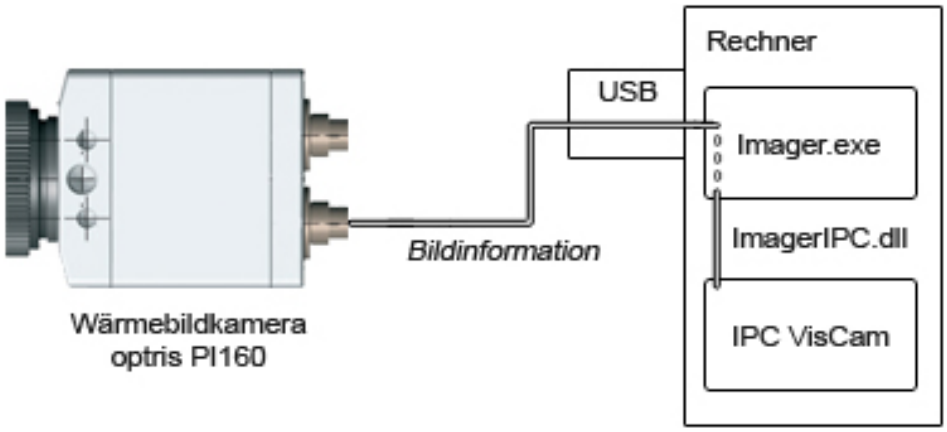

Abbildung 5.5: Verbindung für den Austausch der Wärmebilddaten zwischen dem Imager.exe Programm und der IPC VisCam Beispielanwendung

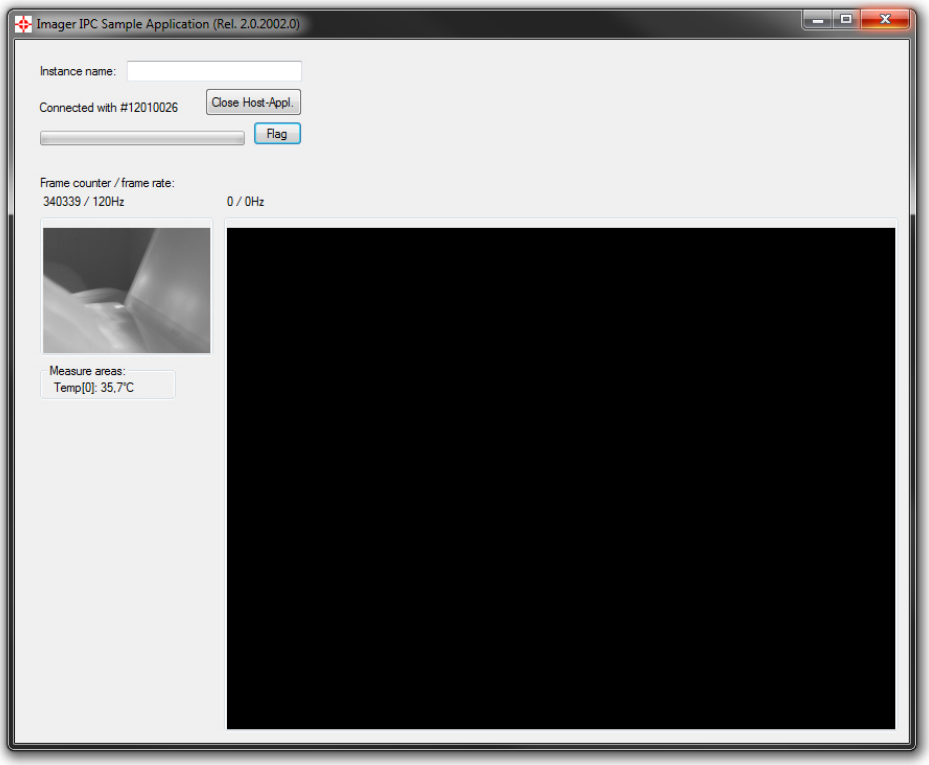

Abbildung 5.6: IPC VisCam Beispielprogramm von optris

```
1
2 InitVis(int frameWidth, int frameHeight, int frameDepth);
3
4 OnFrameVisInit(int frameWidth, int frameHeight, int frameDepth);
5
6 OnNewFrameVis(void * pBuffer, FrameMetadata *pMetadata);
7
8 FileOpen(void)
```

Listing 5.1: Entfernte Methoden der IPC VisCam Basisprogramms

PI200 das visuelle Kamerabild angezeigt. Da hier jedoch nur die *PI160* zum Einsatz kam wird dieses Fenster schwarz bleiben.

Im Folgenden soll das Hauptaugenmerk auf dem Teil dieses Programms liegen, der für das Wärmebild zuständig ist.
Zunächst mussten alle unnötigen Codeelemente, die nicht mehr gebraucht wurden, entfernt werden. Das betrifft alle Teile des Programms, die das visuelle Kamerabild der *PI200* auslesen und anzeigen. Einige Methoden wurden dadurch unbrauchbar und konnten entfernt werden, da sie überhaupt nicht mehr aufgerufen wurden. Siehe dazu Listing 5.1.

Da die Funktion Bilder und Videos zu öffnen nicht benötigt wurde konnte auch die *FileOpen()* Methode entfernt werden.
Übrig blieb ein Programm das nur den Wärmebildstream auslesen und anzeigen kann.

5.2.1.1 Datenrepräsentation des Wärmebilds

Die Repräsentation des Wärmebilds und der Wärmeinformation hängt von den Daten ab, welche die Wärmebildkamera liefert. Es gibt verschiedene Modi für den Datenexport:

- Farben
- Temperaturen
- ADUs (Analog-Digital-Umwandler)

Es ist möglich die Wärmebildinformation als RGB-Bild auszulesen, wobei jedem Farbwert ein Temperaturwert zugeordnet werden kann. Die Wärmebildkamera zeigt Temperaturen von ca. -100 bis 900 Grad Celsius an. Die Temperaturspanne (Ts) umfasst 1000 Grad. Bei 8 Bit pro RGB Wert sind das $(8^2)^3 = 256^3$ verschiedene Werte (W). Dann wäre der Temperaturabstand (Ta) der mit einem RGB-Bild dargestellt werden könnte folgender:

$$Ta = TS/W = 1000/256^3 = 1000/16777216 = 5,96 * 10^{-5} Grad$$

Ein RGB-Bild wäre also mehr als ausreichend um die Temperatur genau darzustellen. Die Umrechnung von den RGB-Werten auf einen Temperaturwert sollte einfach zu bewerkstelligen sein, sofern Informationen darüber vorhanden sind wie die Temperatur in den RGB-Wert umgerechnet wird.
Für diese Implementierung wurde der Datenimport per Temperaturinformation gewählt. Die Daten können nach dem Import zu einem 160x120 Pixel RGB-Bild konvertiert werden. Dafür ist die Methode *GetBitmap()* zuständig. Im Falle des Temperaturwertimports liefert *GetBitmap()* leider nur ein Graustufenbild. Um dafürzusätzlich die Temperaturwerte direkt auslesen zu können, wird eine weitere Funktion der ImagerIPC.dll benötigt. Die in Listing 5.2 zu sehende Methode liest den Temperaturwert einer Koordinate aus.

Es gibt zwei Temperaturmodi, welche in der Dokumentation genauer beschrieben werden. Die Variable *ValuesIR* ist ein *short-Array*. Dort sind die Temperaturwerte der Zeilen eines 160x120 Rasters aufgereiht. Aus diesem Grund berechnet sich der Index folgendermaßen:

```
int index = (y * 160) + x;
```

Der Wert muss anschließend je nach Temperaturmodus umgerechnet werden um die Temperatur zu erhalten. Eine Tabelle aus der Dokumentation der ImagerIPC.dll zeigt wie die Werte umzurechnen sind (siehe **Tabelle 5.1**).

```
1  /**
2  *      Gather temperature value of pixel(x,y).
3  *      The gcnew array<short>() ValuesIR contains the temperature data.
4  *
5  *      \param x      X-Coordinate
6  *      \param y      Y-Coordinate
7  *
8  *      \return Temperature of pixel(x,y) as float
9  */
10 float FormMain::getPixelTemp(int x, int y) {
11        // Calculate index of temperature data for pixel(x,y)
12        // The temperature matrix is always 160x120 sized
13        int index = (y * 160) + x;
14        int value = ValuesIR[index];
15        int mode = ((int) ipc->GetTempRangeDecimal(0,false));
16        float valueFloat;
17        // Convert to float (depending on decimal mode)
18        if (mode == 0) {      /** 1 decimal place */
19               valueFloat = ((float)(value - 1000) / (float)10);
20        }else {        /** if (mode == 1) // 2 decimal places */
21               valueFloat = ((float)value / (float)100);
22        }
23        return valueFloat;
24 }
```

Listing 5.2: getPixelTemp() Methode zum Ermitteln der Temperatur eines Pixels

Dezimalstellen	Formel für Temperaturberechnung	Beispiel
1	$T[^\circ C] = (value - 1000)/10$	$value = 1235$ -> $T = 23.5^\circ C$
2	$T[^\circ C] = value/100$	$value = 2357$ -> $T = 23.57^\circ C$

Tabelle 5.1: Umrechnung der short-Werte zu Temperaturwerten

5.2.2 Bildwiederholfrequenz

Es zeigte sich, dass bei einer Bildwiederholrate von 25Hz die Rechenkapazität der CPU ausreichend war. Bei dieser Frequenz werden Bildsequenzen im Allgemeinen als flüssig laufend empfunden. Stressmessungen im Bereich von über 25 Berechnungen pro Sekunde sollten nicht nötig sein. Folglich muss alle 40ms für jede Kamera ein Bild ausgelesen werden.

```
1  /**
2  *       Constructor
3  */
4  OcvCamImg::OcvCamImg(int device_num)
5  {
6          // Connect to Device
7          cap = new cv::VideoCapture(device_num);
8
9          ...
10
11         // Define Images (gather size and type)
12         cv::Mat gatherimg;
13         *cap >> gatherimg;
14
15         ...
16
17 }
```

Listing 5.3: Ausschnitt des Konstruktors der OcvCamImg Klasse

5.2.3 Die Rolle von openCV

Die Anbindung der visuellen Kamera muss für das Programm komplett neu geschrieben werden, da die vorhandene Funktionalität für die visuelle Kamera nur mit der *optris PI200* nutzbar ist. Dafür und auch für die Gesichtserkennung wird *openCV* zum Einsatz kommen.

5.2.3.1 Kameraanbindung mit openCV

Wie mit *openCV* eine Webcam ausgelesen werden kann, wurde bereits im **Kapitel 2 – Hintergrund** behandelt. Beim Starten des Programms wird zunächst eine Instanz der *OcvCamImg* Klasse erstellt. Diese Klasse wurde erstellt um in diesem Projekt die visuelle Kamera auszulesen und die Gesichtserkennung durchzuführen. Im Konstruktor dieser Klasse wird eine *cv::VideoCapture* Instanz erstellt (siehe Listing 5.3).

Im Konstruktor wird ebenfalls direkt ein erstes Bild ausgelesen. Ein Timer ruft alle 40ms die Methode *handleWebCamFrame()* in der Klasse *FormMain* auf. Diese ruft wiederum die *OcvCamImg::nextFrame()*-Funktion auf, in welcher dann das aktuelle visuelle Kamerabild ausgelesen wird (siehe Listing 5.4).

5.2.3.2 FaceDetection mit openCV

Für die Gesichtserkennung wird im Konstruktor (siehe Listing 5.5) der *OcvCamImg* Klasse für jeden Gesichtsbereich, der hier bestimmt werden soll, eine eigene *cv::CascadeClassifier* Instanz erstellt und das entsprechende XML-Profil geladen, dessen Pfad in einer Konstanten gespeichert ist.

```
1  /**
2  *       Get next frame from Video Capture Device and store it in
3  *       cv::Mat img Class Variable
4  */
5  void OcvCamImg::nextFrame(){
6         // Only capture image if device is connected
7         if (cap->isOpened()) {
8                 *cap >> *img;
9         };
10 }
```

Listing 5.4: OcvCamImg::nextFrame() Methode

```
1  /**
2  *       Constructor
3  */
4  OcvCamImg::OcvCamImg(int device_num)
5  {
6         // Connect to Device
7         cap = new cv::VideoCapture(device_num);
8         // Create Face Detector
9         mFaceDetector = new cv::CascadeClassifier();
10        mFaceDetector->load( CASCADE_FACE );
11        // Create Eye Detector
12        mEyeDetector = new cv::CascadeClassifier();
13        mEyeDetector->load( CASCADE_EYE );
14        // Create Eye Detector
15        mNoseDetector = new cv::CascadeClassifier();
16        mNoseDetector->load( CASCADE_NOSE );
17        // Create Eye Detector
18        mMouthDetector = new cv::CascadeClassifier();
19        mMouthDetector->load( CASCADE_MOUTH );
20        // Define Images (gather size and type)
21        cv::Mat gatherimg;
22        *cap >> gatherimg;
23        img = new cv::Mat(gatherimg.size(), gatherimg.type());
24        out_img = new cv::Mat(OUTPUTHEIGHT, OUTPUTWIDTH, gatherimg.type());
25        out_area_img = new cv::Mat(OUTPUTHEIGHT, OUTPUTWIDTH,
              gatherimg.type());
26        // Create Array and Vector
27        area_rects = new std::vector< cv::Rect >();
28        *area_available = new bool[4];
29 }
```

Listing 5.5: Konstruktor der OcvCamImg Klasse (komplett)

```
1 // Rectangle Vector containing (face_rect, eye_rect, mouth_rect, nose_rect)
2 std::vector< cv::Rect >* area_rects;
3
4 // Array that returns if [face,eye,mouth,nose] has been found this frame
5 bool* area_available[4];
```

Listing 5.6: Ausschnitt aus OcvCamImg Header

Der Aufruf der Methode *OcvCamImg::detectAreas(bool face, bool eye, bool mouth, bool nose)* startet die Gesichtserkennung. Durch den Aufruf der Methode in Listing 5.7 wird per openCV ein Viereck gesucht, welches die Grenzen des Gesichts absteckt. Für die restlichen Gesichtsbereiche existieren analoge Methoden.

Die gefundenen Vierecke werden dann in den Variablen gespeichert, welche in Listing 5.6 gezeigt werden.

In *area_rects* werden die Vierecke selbst als *cv::Rect* gespeichert und *area_available* gibt an, welche der Gesichtsbereiche gefunden wurden.

```
1 /**
2 *      Detect Faces in the parameter image using the face detector
3 *      which loaded the xml-face-haar-file
4 */
5 std::vector< cv::Rect > OcvCamImg::detectFace(cv::Mat mOrigImage){
6       std::vector< cv::Rect > faceVec;
7    mFaceDetector->detectMultiScale( mOrigImage, faceVec, SCALE_FACTOR_FACE
         );
8       return faceVec;
9 }
```

Listing 5.7: OcvCamImg::detectFace() Methode

```
1 /**
2 *      Get pointer to the captured image in output size. This is
3 *      stored in cv::Mat out_img Class Variable.
4 */
5 cv::Mat * OcvCamImg::captureCamMatImg() {
6       // Create return-image
7       cv::resize(*img,*out_img, out_img->size(),1,1,1);
8       return out_img;
9 }
```

Listing 5.8: OcvCamImg::captureCamMatImg() Methode

```
1  /**
2  *      Returns the calculated front measure points. These are the points
3  *      that are used to gather front temperature information.
4  */
5  std::vector< cv::Point > OcvCamImg::frontTempPoints(void) {
6          // Create return Vector
7          std::vector< cv::Point > returnVector;
8          // Check if face rect is available
9          if ((*area_available)[0]) {
10                 cv::Rect face_rect = area_rects->at(0);
11                 // Calculate values
12                 int x = face_rect.x;
13                 int y = face_rect.y;
14                 int x1 = x + (3 * face_rect.width / 8);
15                 int x2 = x + (face_rect.width / 2);
16                 int x3 = x + (5 * (face_rect.width / 8));
17                 int y1 = y + (face_rect.height / 8);
18                 // Create points
19                 cv::Point p1(x1,y1);
20                 returnVector.push_back(p1);
21                 cv::Point p2(x2,y1);
22                 returnVector.push_back(p2);
23                 cv::Point p3(x3,y1);
24                 returnVector.push_back(p3);
25         }
26         frontPoints = returnVector;
27         return returnVector;
28 }
```

Listing 5.9: OcvCamImg::frontTempPoints) Methode

5.2.3.3 Visualisierung der Gesichtsbereiche

Zusätzlich zu dem visuellen Bild können auch die Gesichtsbereiche angezeigt und eingezeichnet werden. Das einfache visuelle Bild des aktuellen Frames der Webcam kann mit der in Listing 5.8 gezeigten Methode abgerufen werden.

Um zusätzlich die Gesichtsbereiche einzeichnen zu lassen wurde die Methode *OcvCamImg::captureCamMatImgDetect()* geschrieben (siehe Listing 5.10).

Die zuvor in *area_rects* gespeicherten Vierecke werden hier eingezeichnet. Die Farbe des Vierecks hängt davon ab welcher Gesichtsbereich damit dargestellt werden soll.
Um auch die Punkte anzeigen zu lassen, die benutzt werden um die Temperatur auszulesen, wurde die Methode *OcvCamImg::captureCamMatImgPoints()* erstellt. Diese zeichnet die Punkte mit Hilfe der *openCV* Methoden ein. Die Koordinaten der Punkte werden abhängig von den Vierecken bestimmt. Wie genau das umgesetzt wurde sollte am Beispiel der Bestimmung Stirn-Temperatur-Punkte in Listing 5.9 klar werden.

Abbildung 5.7 zeigt wie die verschiedenen Anzeigemöglichkeiten aussehen.

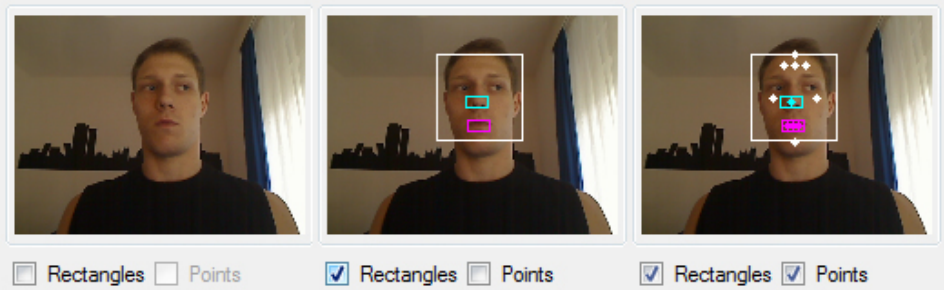

Abbildung 5.7: Visuelles Bild ohne eingezeichnete Gesichtsbereiche (links), mit Vierecken als Gesichtsbereichgrenzen (mitte) und mit Vierecken und Temperaturmesspunkten (rechts)

```
1  /**
2  *      Get pointer to the captured image with painted face areas rect-
3  *      angles in output size. This is stored in out_img Class Variable.
4  */
5  cv::Mat * OcvCamImg::captureCamMatImgDetect() {
6         // Set out_area_img image to resized copy of actual Frame Image
7         cv::resize(*img, *out_area_img, out_area_img->size(), 1, 1, 1);
8         // Loop -> draw any available rectangle in it's preset color
9         if (area_rects->size() > 0) {
10                int vector_index = 0;
11                for (int i = 0; i < 4; i++) {
12                       if ((*area_available)[i]) {
13                              cv::rectangle(*out_area_img,
                                    area_rects->at(vector_index),
                                    RECT_COLORS[i]);
14                              vector_index++;
15                       };
16                };
17        }; // Detect points only without drawing
18        if ((*area_available)[0])
19                frontPoints = frontTempPoints();
20        if ((*area_available)[0])
21                facePoints = faceTempPoints();
22        if ((*area_available)[1])
23                eyePoints = eyeTempPoints();
24        if ((*area_available)[2])
25                mouthPoints = mouthTempPoints();
26        if ((*area_available)[3])
27                nosePoints = noseTempPoints();
28        return out_area_img;
29 }
```

Listing 5.10: OcvCamImg::captureCamMatImgDetect() Methode

5.2.4 Das Programm und seine Funktionalität

Abbildung 5.11 zeigt einen Screenshot des fertigen Programms. Zunächst werden verschiedenen Funktionen betrachtet. Der Bildbereich 1 zeigt das Wärmebild. Im Bildbereich 2 ist das visuelle Bild der Webcam mit eingezeichneten Gesichtsbereichen zu sehen. Mit den Auswahlboxen unter dem Bild kann gewählt werden, ob die Vierecke, welche die Gesichtsgrenzen anzeigen und Temperaturmesspunkte eingezeichnet werden sollen. Mit Hilfe der Steuerelemente im Bildbereich 3 kann eine andere Webcam ausgewählt und eine Verbindung damit hergestellt werden. Das Programm zeigt zwei Bilder eines vorher aufgenommenen Logs. Dass sich das Programm im Log-Abspielmodus befindet, kann daran gesehen werden, dass im Bildbereich 4 der zweite Button von links aktiviert ist. Der Log mit dem Namen im Feld „Log Name“ wurde erfolgreich geladen und angezeigt. Wird dieser Modus deaktiviert, wird der Videostream der Wärmebildkamera und der visuellen Kamera angezeigt. Um einen Log aufzunehmen gibt man den gewünschten Log Namen in das Feld „Log Name“ ein und drückt den ersten Button von links um die Log-Aufnahme zu starten. Wie die Logging-Funktion im Detail arbeitet sollte für diese Ausarbeitung nicht von Belang sein. Im Bildbereich 5 sind verschiedene Auswahlboxen. Sie ermöglichen es anzugeben, welche Gesichtsbereiche im Bild gesucht werden sollen. Für die gefundenen Gesichtsbereiche wird dieTemperatur an den entsprechenden Punkten ermittelt und angezeigt. Um das visuelle Bild und das Wärmebild anzugleichen gibt es die Kalibrierungsfunktion (Bildbereich 6). Diese wird im folgenden Abschnitt beschrieben.

5.2.5 Angleichung des visuellen Bilds und des Wärmebilds

Ein großes Problem war, dass der Bildbereich der visuellen Kamera bedeutend größer war als der Bildbereich der Wärmebildkamera. Was das genau bedeutet sollte Abbildung 5.8 zeigen.

Die beiden Kameras in unseren Beispiel sind direkt nebeneinander aufgestellt. Kamera #1 hat einen kleineren Bildbereich (blau). Sie zeigt einen kleineren Bildausschnitt. Kamera #2 hat einen erkennbar größeren Bildbereich (rot). Abbildung 5.9 zeigt links das Bild von Kamera #1 und recht das Bild von Kamera #2.

Kamera #2 fängt auch einen größeren Umgebungsausschnitt ein. Zusätzlich ist das Bild im Vergleich zu dem von Kamera #1 leicht verschoben. Es werden jedoch zwei Bilder mit dem selben Bildbereich benötigt. Dazu gibt es die Kalibrierungsfunktion.
Manuell wählt man zwei markante Bildpunkte im ersten Bild. Dasselbe wiederholt man für das zweite Bild, wobei man die selben markanten Stellen des Bilds auswählt (siehe Abbildung 5.10. Dadurch kann der Ausschnitt der Kamera mit dem größeren Bildbereich berechnet werden, der den selben Inhalt zeigt wie der gesamte Bildbereich der anderen Kamera.

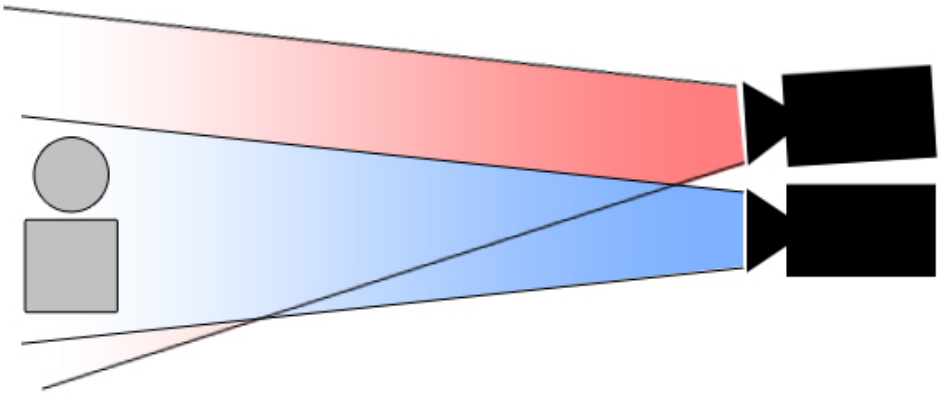

Abbildung 5.8: Bildbereiche zweier nebeneinander positionierter Kameras als Skizze von oben (Blau = Kleiner Bildbereich, Rot = Großer Bildbereich

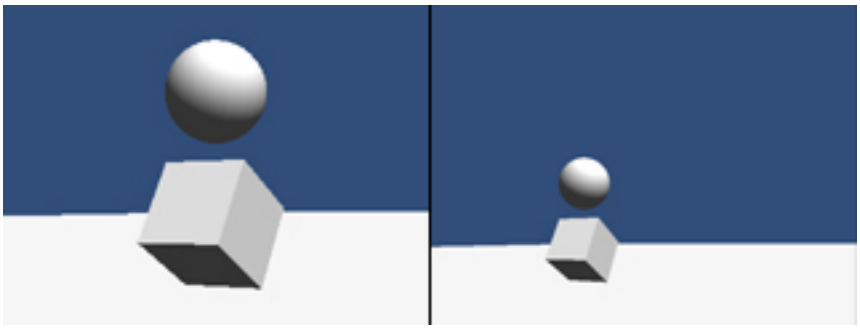

Abbildung 5.9: Bildbereiche zweier nebeneinander positionierter Kameras von vorn (Links: Kleiner Bildbereich, Rechts: Großer Bildbereich

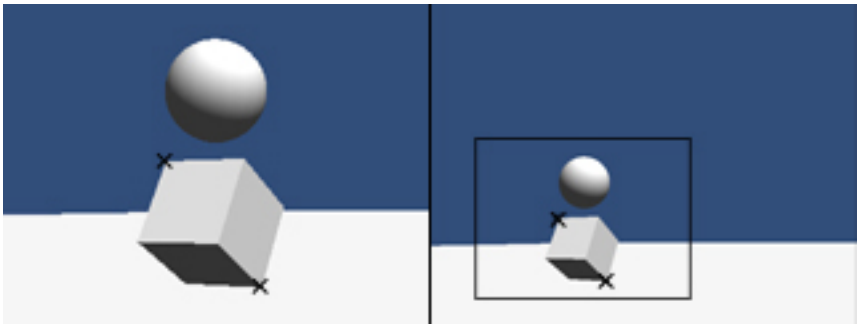

Abbildung 5.10: Kalibrierungsmarkierungen in Bildbereichen zweier nebeneinander positionierter Kameras (Links: Kleiner Bildbereich, Rechts: Großer Bildbereich

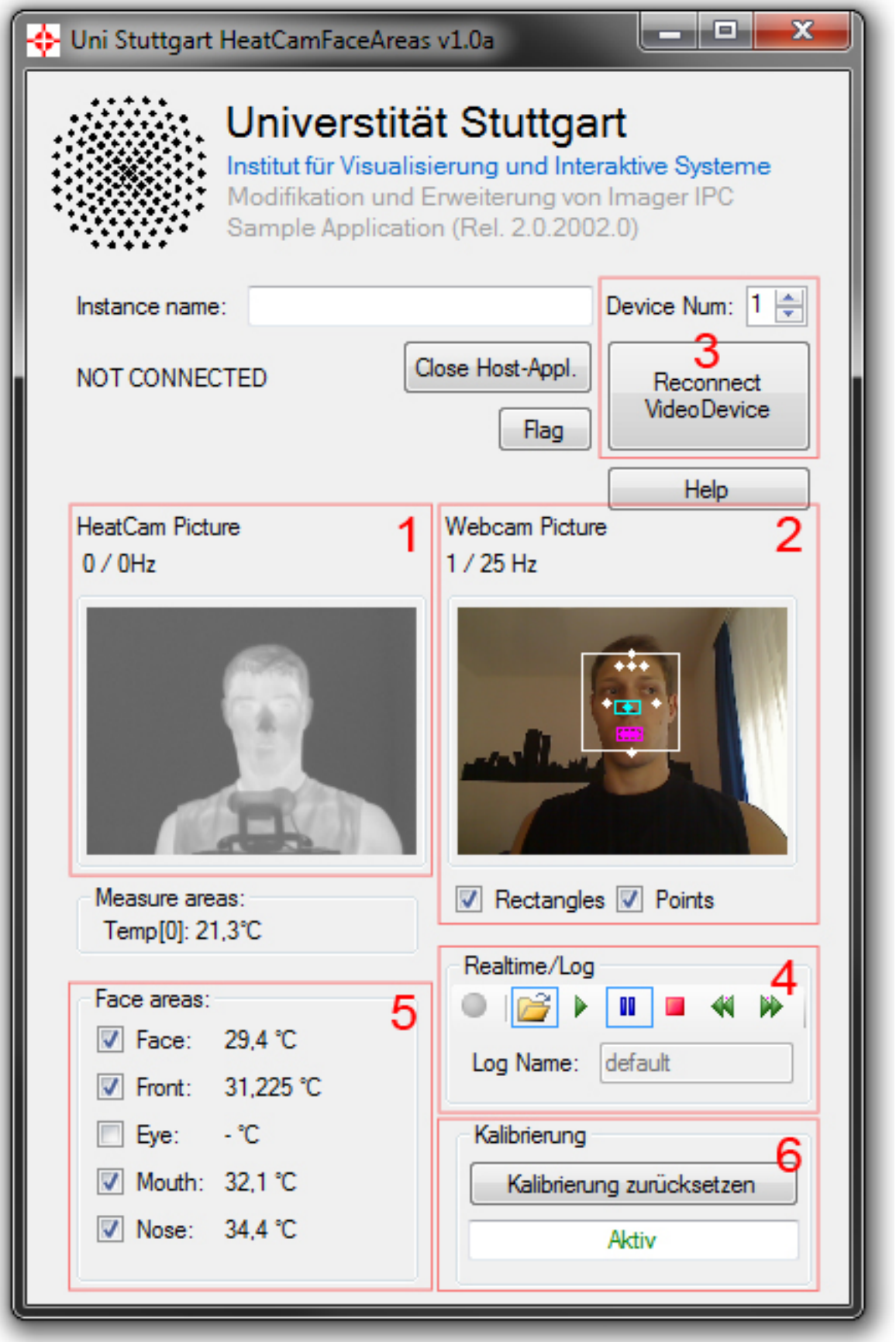

Abbildung 5.11: Screenshot des Programms

Zusammenfassung

Zuerst wurde in diesem Kapitel die genutzte Hardware begutachtet. Die Lösungen der im Konzept vorgestellten Probleme wurden erläutert. Grundlegende Teile des Programms, welche für die Sensoranbindung und die Datenerhebung, -Repräsentation und -Verarbeitung verantwortlich sind, wurden genauer erklärt und die Funktionalität des bei dieser Arbeit entstandenen Programms wurde erläutert.

6 Fazit und Ausblick

Bevor das Fazit gezogen wird folgt hier noch einmal eine kurze Zusammenfassung der Ziele dieser Arbeit. Nach dem Fazit zeigt ein Zukunftsausblick mögliche Änderungen und Verbesserungen für diese Arbeit.

6.1 Zusammenfassung

Das erste Etappenziel war es die Sensoren, in diesem Fall eine visuelle Kamera und eine Wärmebildkamera, anzubinden. Die Daten beider Kameras sollten dann ausgelesen und verarbeitet werden können. Eine der Möglichkeiten ist es die visuellen Bilder und die Wärmebilder so zu „matchen“, dass je ein Bildpunkt des visuellen Bildes einem Punkt des Wärmebildes zugewiesen werden kann. Nach der Gesichtsbestimmung müssen den verschiedenen Gesichtsbereichen Temperaturwerte aus dem Wärmebild zugewiesen werden. Die Temperaturwerte der verschiedenen Gesichtsbereiche sollen anschließend zu einem Stresswert verrechnet werden. Optional kann eine Nutzerstudie Aufschluss über Genauigkeit und Nutzbarkeit der entstandenen Konstruktion geben.

6.2 Fazit

Ein Programm welches sowohl das Wärmebild, als auch das visuelle Bild einlesen und anzeigen kann ist entstanden. Leider war es nicht möglich die Wärmebildkamera direkt auszulesen. Nur über die Inter-Prozess-Kommunikation mit *optris Imager.exe* kann die Wärmebildinformation ausgelesen werden. Das Ziel beide Kameras direkt auszulesen wurde nicht erreicht, doch die so entstandene Lösung sollte dennoch akzeptabel sein. Das Wärmebild steht nach der Konvertierung wie geplant als RGB-Bild zur Verfügung. Doch dieses liefert nicht die Temperaturinformationen. Die Temperaturwerte stehen in einem Array und werden als *short* repräsentiert, müssen aber vor Gebrauch umgerechnet werden.
Eine Forderung war eine Bildwiederholfrequenz von 25Hz. Alle 40ms wird ein weiteres Bild der Webcam geladen. Doch die Berechnungen für die Gesichtserkennung verlangen dem Rechner einiges ab und bremsen die Performance. Wirkliche 25Hz liefert das Programm leider nicht. Besonders deutlich wird das, wenn ein Log abspielt wird. Dieser wirkt beim Abspielen deutlich schneller als bei der Aufnahme.
Im visuellen Bild wird mit Hilfe von openCV nach Gesichtsbereichen gesucht und damit werden die Koordinaten der Temperaturmesspunkte berechnet. Dazu können das visuelle Bild und das Wärmebild mit Hilfe der Kalibrierungsfunktion so eingestellt werden, dass sie

denselben Bildbereich zeigen.
Diese Koordinaten werden benutzt um die Temperatur der Stirn, Nase und anderer Gesichtsbereiche zu bestimmen. Die *Cognitive Workload* wird nicht direkt angegeben. Das nachzuholen wäre kein großer Aufwand, da lediglich die erhobenen Temperaturdaten mit der bereits erwähnte Formel umgerechnet und angezeigt werden müssten. Es wurde jedoch zu diesem Zeitpunkt bewusst weggelassen, da die Korrektheit des Wertes noch nicht geprüft wurde. Das könnte zu irreführenden Missverständnissen führen. Die Nutzerstudie musste aus Zeitgründen aufgegeben werden.

6.3 Ausblick

Zunächst könnte die Einfachheit der Handhabung und die Genauigkeit der Anwendung verbessert werden. Eine Wärmebildkamera mit höherer Pixelauflösung und eine visuelle Kamera deren Bildbereich mit dem der Wärmebildkamera übereinstimmt könnte die Genauigkeit der Anwendung verbessern und eine Kalibrierung überflüssig machen.
Auch die Performance könnte noch verbessert werden. Durch effizientere Gesichtsfindungsalgorithmen, Feinabstimmungen und durch das Anwenden verschiedener Filter könnte das erreicht werden.
Anschließend könnte eine Nutzerstudie geplant und durchgeführt werden um herauszufinden wie zuverlässig die Messung des *Cognitive Workloads* mit dieser Methode funktioniert. Dazu sollten der errechnete Workload mit Vergleichswerten durch Selbstbeobachtung und Workloadberechnungen unter Verwendung anderer Sensoren verglichen werden. In einem nächsten Schritt könnte eine Testapplikation entstehen, welche die Schwierigkeit der zu bearbeitenden Aufgabe je nach Arbeitslast des Nutzers anpasst. An diesem Punkt könnte beurteilt werden, wie gut sich die Workloadmessung per Wärmebildkamera für die Anpassung der Benutzeroberfläche eines Programms eignet.

Danksagung

An dieser Stelle möchte ich allen Leuten herzlich danken, die mir bei der Gestaltung und Ausarbeitung dieser Bachelorarbeit geholfen und mich unterstützt haben.
Besonderer Dank gilt meinen Eltern die mich mit Geduld, Motivation und Vertrauen, aber auch finanziell, unterstützt haben.
Meinem Kommilitonen Kai Mindermann danke ich für die Hilfe mit **LaTeX** und für die ständige Verfügbarkeit für Formulierungs- und Formatierungsfragen, aber auch für die Unterstützung die er mir in *C++* Angelegenheiten entgegen brachte.
Auch bei Florian Straßer, einem weiteren Kommilitonen, möchte ich mich für die gute Zusammenarbeit bedanken.
Mein weiterer Dank gilt Bastian Pfleging, Niels Henze und Prof. Albrecht Schmidt für die Überlassung dieses Themas und die Hilfe bei der Programmierung.
Auch vielen namentlich nicht genannten Freunden und Bekannten möchte ich für die Hilfe, Beratung, Anregung und auch gelegentliche (notwendige) Ablenkung danken.

Literaturverzeichnis

[Bad10] A. Baddeley. Working Memory: Theories, Models, and Controversies. Band 20, S. R136 – R140. York YO10 5DD, UK, 2010. doi:10.1016/j.cub.2009.12.014. URL http://www.sciencedirect.com/science/article/pii/S0960982209021332. (Zitiert auf Seite 18)

[CEC11] S. Chen, J. Epps, F. Chen. A comparison of four methods for cognitive load measurement. In *Proceedings of the 23rd Australian Computer-Human Interaction Conference*, OzCHI '11, S. 76–79. ACM, New York, NY, USA, 2011. doi:10.1145/2071536.2071547. URL http://doi.acm.org/10.1145/2071536.2071547. (Zitiert auf den Seiten 40 und 43)

[cpl12] cplusplus.com. C++ Dokumentation, 2012. URL http://www.cplusplus.com/doc/tutorial/namespaces/. (Zitiert auf Seite 24)

[Dav04] S. Davis. *C++ For Dummies*. Wiley Pub., 2004. URL http://books.google.de/books?id=QuluB-uwHkEC. (Zitiert auf den Seiten 20 und 22)

[Gó] Z. Gócza. Myth 23: Choices should always be limited to 7+/-2. URL http://uxmyths.com/post/931925744/myth-23-choices-should-always-be-limited-to-seven. (Zitiert auf Seite 12)

[Gmb12] optris GmbH. www.optris.de, 2012. URL http://www.optris.de/infrarotkamera-pi160. (Zitiert auf Seite 53)

[HKFD10] E. Haapalainen, S. Kim, J. F. Forlizzi, A. K. Dey. Psycho-physiological measures for assessing cognitive load. In *Proceedings of the 12th ACM international conference on Ubiquitous computing*, Ubicomp '10, S. 301–310. ACM, New York, NY, USA, 2010. doi:10.1145/1864349.1864395. URL http://doi.acm.org/10.1145/1864349.1864395. (Zitiert auf den Seiten 40, 45 und 50)

[Hol09] J. Hollandt. Welt der Physik - Infrarotstrahlung, 2009. URL http://www.weltderphysik.de/gebiete/atome/forschung-mit-licht/elektromagnetisches-spektrum/infrarotstrahlung/. (Zitiert auf Seite 16)

[Kir02] P. A. Kirschner. Cognitive load theory: implications of cognitive load theory on the design of learning. *Learning and Instruction*, 12(1):1 – 10, 2002. doi:10.1016/S0959-4752(01)00014-7. URL http://www.sciencedirect.com/science/article/pii/S0959475201000147. (Zitiert auf den Seiten 17 und 40)

[KKY+98] H. Kataoka, H. Kano, H. Yoshida, A. Saijo, M. Yasuda, M. Osumi. Development of a skin temperature measuring system for non-contact stress evaluation. In *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 1998*, Band 2, S. 940 –943 vol.2. 1998. doi:10.1109/IEMBS.1998.745598. (Zitiert auf den Seiten 41, 42, 43, 48 und 49)

[LHB82] M. Loeb, D. H. Holding, M. A. Baker. Noise stress and circadian arousal in self-paced computation. Band 6, S. 43–48. Springer Netherlands, 1982. URL http://dx.doi.org/10.1007/BF00992136. 10.1007/BF00992136. (Zitiert auf Seite 45)

[Log12] Logitech. http://www.logitech.com, 2012. URL http://www.logitech.com/de-de/webcam-communications/webcams/quickcam-vision-pro-9000-mac. (Zitiert auf Seite 55)

[Lou08] D. Louis. *C, C++: Das komplette Programmierwissen für Studium und Job*. Markt + Technik, 2008. URL http://books.google.de/books?id=XPdOBeqXeZMC. (Zitiert auf Seite 20)

[Mar95] S. Marshall. *Schemas in Problem Solving*. Cambridge University Press, 1995. URL http://books.google.de/books?id=iKSYrsXe6f0C. (Zitiert auf Seite 19)

[May01] R. Mayer. *Multimédia Learning*. Cambridge University Press, 2001. URL http://books.google.de/books?id=ymJ9o-w_6WEC. (Zitiert auf Seite 17)

[Mic12a] Microsoft. Microsoft MSDN DE Library IntelliSense, 2012. URL http://msdn.microsoft.com/de-de/library/hcw1s69b.aspx. (Zitiert auf Seite 21)

[Mic12b] D. P. Microsoft. Microsoft Visual Studio 2010 IntelliSense Screenshot, 2012. URL http://blogs.msdn.com/b/dparys/archive/2010/01/29/visual-studio-2010-intellisense-quick-hit.aspx. (Zitiert auf Seite 21)

[Mil56] G. Miller. The magicalnumberseven, plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, S. "81–97, 1956. (Zitiert auf Seite 11)

[Plu12] W. Pluta. Google X simuliert Gehirn mit 16.000 Prozessorkernen, 2012. URL http://www.golem.de/news/maschinenlernen-google-x-simuliert-gehirn-mit-16-000-prozessorkernen-1206-92 html. (Zitiert auf Seite 31)

[SAK11] J. Sweller, P. Ayres, S. Kalyuga. *Cognitive Load Theory*. Explorations in the Learning Sciences, Instructional Systems and Performance Technologies. Springer, 2011. URL http://books.google.de/books?id=sSAwbd8qOAAC. (Zitiert auf den Seiten 17 und 39)

[SG73] H. A. Simon, K. Gilmartin. A simulation of memory for chess positions. *Cognitive Psychology*, 5(1):29 – 46, 1973. doi:10.1016/0010-0285(73)90024-8. URL http://www.sciencedirect.com/science/article/pii/0010028573900248. (Zitiert auf den Seiten 17, 19 und 40)

[SMP98] J. Sweller, J. van Merrienboer, F. Paas. Cognitive Architecture and Instructional Design. *Educational Psychology Review*, 10(3):251–296, 1998. doi:10.1023/A:1022193728205. URL http://dx.doi.org/10.1023/A:1022193728205. (Zitiert auf den Seiten 17 und 39)

[tea12a] openCV dev. team. openCV Website Introduction, 2012. URL http://docs.opencv.org/modules/core/doc/intro.html. (Zitiert auf Seite 33)

[tea12b] openCV dev. team. openCV Website Transformation Dokumentation, 2012. URL http://opencv.willowgarage.com/documentation/cpp/geometric_image_transformations.html. (Zitiert auf Seite 35)

[(Us09] A. (User). www.gutefrage.net Diskussion Intuition, 2009. URL http://www.gutefrage.net/frage/intuitivitaet-oder-intuition. (Zitiert auf Seite 11)

[WGPH10] J. Wijsman, B. Grundlehner, J. Penders, H. Hermens. Trapezius muscle EMG as predictor of mental stress. In *Wireless Health 2010*, WH '10, S. 155–163. ACM, New York, NY, USA, 2010. doi:10.1145/1921081.1921100. URL http://doi.acm.org/10.1145/1921081.1921100. (Zitiert auf den Seiten 40, 44 und 45)

Alle URLs wurden zuletzt am 14.10.2012 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

(Ort, Datum, Unterschrift)