Projektgruppe Bull's Eye

Abschlussbericht

2016 - 2017

Daniela Betzl
Tim Lukas Cofala
Aljoscha Niazi-Shahabi
Stefan Niewerth
Henrik Reichmann
Rieke von Bargen

Prof. Dr. Susanne Boll-Westermann
Betreuer: M.Sc. Tim Stratmann, M.Sc. Uwe Grünefeld
Organisation: M.Sc. Marion Koelle
Stand: 5.11.2017
Version: 1.1
Carl von Ossietzky Universität Oldenburg
Fakultät II Informatik, Wirtschafts- und Rechtswissenschaften

Danksagung

An dieser Stelle möchten wir unseren Betreuern M.Sc. Tim Stratmann und M.Sc. Uwe Grünefeld bedanken, die uns über das Jahr hinweg richtungsweisend und engagiert begleitet und unterstützt haben.

Des Weiteren möchten wir uns herzlich bei Prof. Dr. Susanne Boll-Westermann, die das Projekt ins Leben gerufen hat, sowie bei M.Sc. Marion Koelle für die Organisation bedanken. Ein Dankeschön gilt auch allen Mitarbeiter des OFFIS, die an unseren Studien teilgenommen haben und uns so Feedback zu unserem Framework sowie Prototypen gegeben haben.

Vielen Dank für eure Zeit und Mühe, die Ihr in unserem Projekt investiert habt!

gezeichnet,
die Projektgruppe Bull's Eye

Virtual und Augmented Reality befinden sich als Trends in einem Aufschwung, welcher durch Erfolge wie u.a. die Oculus Rift oder Microsoft HoloLens dazu führt, dass diese Technologien immer breitere Anwendungsgebiete erfahren. Auch der Forschungszweig Mensch-Maschine-Interaktionen beschäftigt sich zunehmend mit diesen Technologien und versucht neue, intuitive Methoden zum Erkunden und Interagieren mit den virtuellen Objekten zu realisieren. Ein vielversprechender Ansatz ist die Nutzung von Blickdaten des Anwenders als mögliche Interaktionstechnik. Erste Versuche Head-Mounted-Displays mit integriertem Eye-Tracking als Interaktionsmöglichkeit anzubieten, waren jedoch häufig in höheren Preisklassen angesiedelt und für viele Anwender damit nicht zugänglich. Im Rahmen eines einjährigen Projektes, war es Ziel der Projektgruppe Bull's Eye eine Lösung für eine Kombination aus virtueller und erweiterter Realität mit Eye-Tracking zu entwerfen, umzusetzen und zu evaluieren. Jedoch soll diese Lösung primär in Rapid-Prototyping-Prozessen zum Einsatz kommen und unter niedrigen Kosten und geringem Lernaufwand schnell, einfach und quelloffen zur Verfügung stehen. Hierfür wurde auf Basis des Google Cardboards ein eigenes Head-Mounted-Display konstruiert und ein Software-Framework geschrieben, welches eine intuitive Implementierung von Interaktionstechniken für das Eye-Tracking in Virtual und Augmented Reality ermöglicht.

Abkürzungsverzeichnis

App Applikation

AR Augmented Reality

ARBI Abteilung Rechner- und Netzbetrieb Informatik

CT Computertomographie

DIY Do-It-Yourself

GUI Graphical User Interface

HCD Human-Centered-Design

HCI Human–computer interaction

HMD Head Mounted Display

HUD Head-up-Display

IP Internet Protocol

IR Infrarot

LED Light emitting diode

MIT Massachusetts Institute of Technology

MR Mixed Reality

MRT Magnetresonanztomographie

NASA National Aeronautics and Space Administration

OLED Organic light emitting diode

SDK Software Development Kit

SSR Six-Segmented Rectangular

TCP Transmission-Control-Protocol

UDP User-Datagram-Protocol

USB Universal Serial Bus

VR Virtual Reality

WLAN Wireless Local Area Network

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Erweiterte und virtuelle Realität sind vielseitige Technologien und beeinflussen die unterschiedlichsten Bereiche des täglichen Lebens, der Wissenschaft und Wirtschaft. Auch der Forschungszweig Mensch-Maschine-Interaktionen beschäftigt sich zunehmend mit diesen Technologien und versucht neue, intuitive Methoden zum Erkunden und Interagieren mit diesen virtuellen Welten zu realisieren [WGA+17]. Ein vielversprechender Ansatz ist die Nutzung von Blickdaten des Anwenders, um solche Interaktionen zu ermöglichen. Die Blickdaten können über einen Eye-Tracker ermittelt und so ohne zusätzlich nötige Eingabegeräte genutzt werden.
Im Rahmen eines einjährigen Projektes, war es Ziel der Projektgruppe Bull's Eye ein Konzept für solch eine Kombination aus virtueller und erweiterter Realität mit Eye-Tracking zu entwerfen, umzusetzen und zu evaluieren. Dieser Bericht wird einen Überblick über den Verlauf und die Ergebnisse dieser Projektgruppe geben. Hierzu wird in diesem einleitenden Kapitel zunächst die Motivation hinter der Projektgruppe näher erläutert, sowie die Problemstellung und die Zielsetzung genauer beschrieben. Zum Abschluss folgt ein Überblick über den Aufbau des Berichtes mit einer Beschreibung der folgenden Kapitel.

1.1. Motivation

Auch wenn die Forschung zum Einsatz von Eye-Tracking und Blickkontrolle als Methode zur Mensch-Maschine-Interaktion in den letzten 20 Jahren vielversprechend war, haben es nur wenige Eye-Tracking-Systeme erfolgreich in die kommerzielle Produktion geschafft [Jac09]. Die Verwendung von Eye-Tracking ist jedoch gerade in einem Aufschwung [Goo17b]. Auch im Human–computer interaction (HCI)-Kontext finden sich eine steigende Anzahl von Studien, die sich mit der Augenverfolgung beschäftigen [Jac09]. Parallel befinden sich die Technologien Virtual Reality und Augmented Reality gerade in einem aufsteigenden Trend, welcher sich nicht nur auf wissenschaftliche Anwendungsgebiete erstreckt, sondern auch die Spieleindustrie veröffentlicht zunehmend mehr Anwendungen, die auf virtueller oder erweiterter Realität basieren [Jon13]. Für die Entwicklung von Virtual Reality (VR)/AR-Anwendungen, die in Verbindung mit Eye-Tracking-Systemen funktionieren sollen, muss als zentrale Herausforderung die verschiedenen Hard- und Software-Komponenten zu einem funktionierenden System verbunden wer-

den.
Da die meisten bis dato eher in höheren Preisklassen angesiedelt waren [SMH+12], entsteht mit der Einführung von preiswerteren Eye-Trackern und Open-Source-Projekten zudem das Potenzial, auf Blickinteraktionen beruhende Techniken einer viel breiteren Gruppe von Nutzern zu ermöglichen. Dieser Aufgabenstellung hat sich die Projektgruppe Bull's Eye, wie bereits zuvor erwähnt, innerhalb eines einjährigen Projektes angenommen. Die preiswerte und auf Open-Source-Projekten basierende Umsetzung war für unser Projekt insofern von hoher Bedeutung, da unsere primäre Anwender HCI-Wissenschaftler in Rapid-Prototyping-Prozessen sind. In diesem Kontext werden oft möglichst schnelle Lösungen mit zudem begrenzten finanziellen Mitteln benötigt. Genau hier soll unser Produkt ansetzen und potenziellen Anwendern eine möglichst einfache, schnelle und kostengünstige Methodik bieten, Eye-Tracking-Interaktionstechniken in AR und VR realisieren zu können.

1.2. Problemstellung und Zielsetzung

Ein kostengünstiges, schnell umzusetzendes Framework für die Kombination von Eye-Tracking mit Augmented und Virtual Reality könnte vor allem in der Forschung zu Mensch-Maschine-Interaktionen von großem Nutzen sein. Gerade bei der Planung und Durchführung von Experimenten würden Wissenschaftler in diesem Themenbereich stark von einer Möglichkeit profitieren, ihre Ideen vorab ausprobieren zu können. Mit einem kostengünstigen Prototyp könnten die Experimente im Vorfeld veranschaulicht und in ersten Testdurchläufen erprobt werden, ohne auf kostspielige Hardware zurückgreifen zu müssen. Ziel der Projektgruppe ist es, neben dem Entwurf einer kostengünstigen Hardwarelösung, ebenfalls ein Software-Framework für die schnelle Implementation von Interaktionen zwischen Augendaten und virtuellen Elementen zu realisieren. In der Kombination sollen Hard- und Software ganz im Sinne des Rapid-Prototypings stehen und die schnelle Erstellung eines ersten Prototypens zu Testzwecken ermöglichen. Durch umfassende Anleitungen und beispielhafte Anwendungen sollen einem potentiellen Nutzer zusätzliche Hilfen geboten werden. Weiterhin würde eine Veröffentlichung unter einer Open-Source-Lizenz, zusätzlich die Möglichkeit zur individuellen Anpassung und Erweiterung des Frameworks eröffnen und so zusätzlich das Rapid-Prototyping unterstützen. Aufgrund der geringen Anschaffungskosten und die unkomplizierte Implementation wäre unsere Lösung natürlich nicht nur für Wissenschaftler interessant sondern würde jeder an Eye-Tracking interessierten Person einen guten Einstiegspunkt bieten.

1.3. Aufbau des Abschlussberichtes

Die nachfolgende Ausarbeitung gliedert sich in sieben große Kapitel. Zu Beginn wird in Kapitel Grundlagen (2) die zentralen Begriffe Virtual Reality, Augmented Reality, Mixed Reality sowie Eye-Tracking erläutert und auf Software in diesen Themenbereich eingegangen.
Das nächste Kapitel (3) handelt vom Projektmanagement. Hierbei wird auf die Projektgruppe und die Rollen innerhalb dieser eingegangen. Anschließend werden die beiden verwendeten Verfahren Scrum und Human-Centered Design näher beschrieben und darauf eingegangen, wieso sich für diese Verfahren entschieden wurde. Es wird auf die Anwendung der Modelle eingegangen, sowie abschließend auf die vier Quartale Rückschluss gezogen.
Das nachfolgende Kapitel Anforderungen (4) beginnt mit einer Beschreibung der Durchführung der Nutzerinterviews und die daraus resultierenden Anforderungen für das Projekt.
Fortführend wird in den Kapiteln 5 und 6 auf die Hardware sowie die Software eingegangen. Hierbei wird im Kapitel Hardware die Zielsetzung sowie das Konzept erläutert. Danach folgen die vier Iterationen der Entwicklung des Hardware-Prototypen. Beim Kapitel Software wird zunächst auf die Zielsetzung, daraufhin auf die Konzeptentwicklung eingegangen. Anschließend wird die Implementierung bzw. die Umsetzung verdeutlicht.
Nachdem die vorherigen Kapitel Aufschluss über die Grundlagen und Herangehensweise geben, gibt das Kapitel 7 wieder, welche Rückschlüsse sich durch Evaluationen der Hardware sowie Software herauskristallisieren.
Das letzte Kapitel (8) beinhaltet ein Fazit der Projektgruppe sowie Resultate in Bezug auf den entstandenen Hardware-Prototypen und das dazu entstandene Software-Framework. Zudem gibt es einen Ausblick, der Erweiterungen in Hinblick auf Hard- sowie Software gibt und Abgrenzungen und Limitierungen aufzeigt.

2. Grundlagen

Dieses Kapitel beschäftigt sich mit Themen und Software, die als Grundlage für die nachfolgenden Kapitel dienen. Da Kern dieses Projektes der Anspruch ist, zwei unterschiedliche Welten an Technologien zu kombinieren, müssen diese zunächst einzeln definiert werden. Hierfür erfolgt zuerst eine Beschreibung der Virtual Reality, Mixed und Augmented Reality mit beispielhaften Anwendungen. Daraufhin soll ein Überblick über die Technik des Eye-Trackings und der Forschung zu Interaktionstechniken zwischen virtuellen Objekten und Augenbewegungen gegeben werden. Anschließend wird das Google Cardboard und die verschiedene Software, die im Rahmen des Projekts genutzt wurden, beschrieben, sowie Methoden zur Optimierungen von Objekten für die Nutzung in Augmented Reality und Virtual Reality vorgestellt. Der Inhalt dieses Kapitel besteht aus Seminararbeiten der einzelnen Projektgruppenmitglieder, die zu Beginn des Projektes und für das Projekt erarbeitet wurden, und wurde im Verlauf des Projektes erweitert.

2.1. Virtual Reality

Virtuelle Realität ist eine mittels Computer simulierte Wirklichkeit. In den letzten Jahren gab es viel Bestreben, Hard- und Software im Bereich der virtuellen Realität zu verbessern, um den Nutzer in eine möglichst realistische, virtuelle Welt eintauchen zu lassen. [Bri09] Zudem sollen dem Nutzer unterschiedlichste Möglichkeiten gegeben werden mit dieser Welt zu interagieren und sensorische Rückmeldungen aus dieser Welt zu erhalten. Im Idealfall soll es dem Nutzer möglich sein sich wie in seiner natürlichen Umgebung zu verhalten und damit stellt dieses Thema die Ultima Ratio der Mensch-Maschine-Kommunikation dar [Bri09]. Demzufolge muss die Interaktion in der virtuellen Realität durch möglichst viele Sinne stattfinden. Neben dem offensichtlichen Aspekt der visuellen Präsentation einer virtuellen Welt, spielen auch akustische und taktile Reize eine wichtige Rolle. Um eine Interaktion des Anwenders mit der virtuellen Welt zu ermöglichen gibt es ebenfalls unterschiedliche Optionen. So sind je nach verwendeten Geräten Eingaben über Handsteuergeräte, Spracherkennung, Bewegungen des Körpers oder Verändern der Position möglich. [Bri09]
Eine gängige Möglichkeit einen Nutzer in die virtuelle Realität zu versetzen ist die Verwendung sogenannter Head-Mounted Displays. Diese ähneln einer Brille,

die mit kleinen Displays ausgestattet ist. Sie nimmt dem Nutzer die Sicht auf die umgebende Welt und präsentiert eine virtuelle Welt auf Bildschirmen vor den Augen des Nutzers. Den beiden Augen des Nutzers wird dabei je ein leicht versetztes Bild dargeboten, wie es auch sonst der natürlichen Wahrnehmung der Augen entspricht. Dies ermöglicht eine Illusion einer dreidimensionalen Wahrnehmung. Zwischen Bildschirm und Augen sind dabei meist Linsen verbaut, die das Sichtfeld des Nutzers erweitern und so ein Gefühl vermitteln von dem Bild umgeben zu sein. Um sich in der virtuellen Welt zu orientieren, können Bewegungen des Kopfes durch das Head Mounted Display (HMD) verfolgt werden. [TPA+15]

2.1.1. Technik

Damit die Darstellung der virtuellen Realität für den Nutzer möglichst realistisch erscheint, gilt es einige technische Faktoren bei der Erstellung solcher Head-Mounted Displays zu beachten. Um den Immersionseffekt, also das Gefühl des Nutzers sich wahrhaftig in die Szene hineingezogen zu fühlen, zu verstärken, ist vor allem eine möglichst hohe Abdeckung des Gesichtsfeldes durch das dargestellte Bild von Nöten [Whe16]. Das Gesichtsfeld im Allgemeinen ist der komplette Raum den die beiden Augen erfassen können [Dud16]. Den Bereich den je ein Auge abdeckt wird monokulares Gesichtsfeld genannt. Die Schnittmenge der beiden monokularen Bereichen wird binokulares Gesichtsfeld genannt. Für das räumliche Sehen ist genau dieser ungefähr 114° große Bereich von Bedeutung [Whe16]. Aber auch eine Abdeckung der peripheren Bereiche kann Auswirkungen auf die Wahrnehmung der virtuellen Realität haben. So kann beispielsweise die Wahrnehmung von eigener Bewegung beeinflusst werden. Zudem könnte das Gesichtsfeld einen Einfluss auf die sogenannte "Simulator Sickness", also eine durch Simulation von Bewegung ausgelöstes Unwohlsein, beeinflussen [**?**].
Neben dem Gesichtsfeld spielt auch die Latenz zwischen den Bewegungen des Nutzers und den Wahrnehmungen seines sensorischen Systems eine wichtige Rolle. Das System muss in der Lage sein, die Bewegungen des Nutzers schnell wahrzunehmen und eine entsprechende Veränderung des dargestellten Bildes mit möglichst geringer Verzögerung zu berechnen. Ansonsten kann es zur "Simulator Sickness" kommen. Des Weiteren sind auch die Bildwechselfrequenz und die Bildwiederholrate von Bedeutung. Je besser diese sind, desto flüssiger wirkt das Bild für den Nutzer. [Whe16]
Um die Erfahrung für den Nutzer noch realistischer zu gestalten, muss ebenfalls die Pixeldichte des Bildschirms beachtet werden. Da sich der Bildschirm verhältnismäßig nah am Auge befindet, muss die verwendete Auflösung umso größer sein. Dabei werden in HMDs üblicherweise Organic light emitting diode (OLED)-Displays verwendet. Aktuelle Modelle wie beispielsweise die HTC Vive oder die Oculus Rift weisen dabei eine Auflösung von 1080x1200 Pixeln auf [Sta16].

Wie sich an den obigen Punkten erkennen lässt, stellt eine realistische Darstellung der virtuellen Realität hohe Anforderungen an die Hardware. Bei High-End HMDs wie beispielsweise der Oculus Rift, werden die Berechnungen der Bilder deshalb nicht in der Brille selbst durchgeführt, sondern in einen angeschlossenen Computer ausgelagert. Bei der Anschaffung einer dieser Brillen muss folglich auch die Anschaffung eines leistungsstarken Rechners mitbedacht werden. Ein vom Hersteller für die Oculus Rift empfohlener Computer ist mindestens mit einem Intel i5-4590 Prozessor ausgestattet, verfügt über 8 GB an Arbeitsspeicher und besitzt eine Grafikkarte, die der NVIDIA GTX 970 oder besser entspricht [Inc16].

2.1.2. Anwendungsfelder

Die Anwendungsmöglichkeiten der virtuellen Realität sind mannigfaltig. Primäre Anwendung vieler der zuvor vorgestellten Modelle ist wohl die Unterhaltungselektronik. Für die HTC Vive lassen sich beispielsweise über die eigene Software-Plattform SteamVR Computerspiele beziehen, die mit der Brille gespielt werden können. Andere Software wie beispielsweise Tilt Brush von Google lässt einen im dreidimensionalen Raum mit unterschiedlichen Hilfsmitteln Bilder und Objekte zeichnen [Cor16]. Des Weiteren ist es beispielsweise über die Samsung Gear VR möglich 360 Grad Videos zu schauen [Gmb16].
Doch die Anwendungsmöglichkeiten gehen weit über lediglich Unterhaltung hinaus. So gibt es eine Reihe von Untersuchungen zur Verwendung von virtueller Realität in der Medizin und Psychologie. Virtuelle Realität kann bei den unterschiedlichen kognitiven Therapien zum Einsatz kommen, zum Beispiel, wenn es um Konfrontationen von Angstpatienten mit dem Angst auslösenden Objekt geht [NN16]. Eine weitere Studie zeigt den Einsatz von virtueller Realität bei der Behandlung von posttraumatischen Belastungsstörungen [RPJ+14]. Als eine beispielhafte Nutzung der virtuellen Realität in medizinischen Kontexten sei hier die Therapie von Patienten, die an Folgen eines Schlaganfalls genannt. Henderson, Korner-Bitensky und Levin liefern in einem Artikel Evidenz für einen gesteigerten Therapieerfolg bei der Anwendung von virtueller Realität nutzenden Therapien bei der Rehabilitation von Patienten, die nach einem Schlaganfall an Hemiparese (einer halbseitigen Lähmung) leiden [HKBL14]. Diese Artikel seien hier nur beispielhaft für eine Vielzahl an wissenschaftlichen Artikeln genannt, die momentan zu diesem Thema publiziert werden. Sie zeigen das Potential bei der Anwendung von virtueller Realität im klinischen Kontext.
Ein weiteres großes Anwendungsgebiet der virtuellen Realität sind Lern- und Trainingsprogramme, da sie die Möglichkeit bietet Lerninhalte direkt zu erleben. So benutzt die National Aeronautics and Space Administration (NASA) beispielsweise ein virtuelles Labor zum Training der Astronauten [Pso95]. Aber auch im Training von Chirurgen kann die virtuelle Realität eingesetzt werden. Seymour et.

al. zeigten eine signifikant bessere Leistung von Ärzten im Operationsraum, wenn diese vorher in einer ähnlichen virtuellen Umgebung trainiert hatten [SGR+02].
Auch die Industrie hat Anwendungsmöglichkeiten für die virtuelle Realität gefunden. Beim virtuellen Prototyping wird ein virtueller vor einem physischen Prototyp erzeugt. Damit spielt die virtuelle Realität eine entscheidende Rolle bei dem Design von neuen Produkten. Die Designer können den Prototyp als dreidimensionale Skizze in einer virtuellen Welt erstellen. An diesem Modell können dann beispielsweise Gelenke ausprobiert werden, Interaktion mit einem Menschen erprobt, Teile der Prototypen auseinandergenommen oder zusammengesteckt werden. Da Prototyping eine immer größere Bedeutung in Entwicklungsprozessen neuer Produkten gewinnt, ist dies eine von ihrer Bedeutung her nicht zu unterschätzende Anwendungsmöglichkeit der virtuellen Realität [JCL97]. Aber auch in anderen Bereichen der Produktherstellung kann die virtuelle Realität benutzt werden. Sie kann zu Planungs-, Simulation- oder Inspektionszwecken verwendet werden, oder als Trainingsumgebung für Mitarbeitern in neuen Aufgabenbereichen [JCL97].
All die oben genannten Punkte reißen die möglichen Anwendungen dieser Technologie nur an, zeigen aber eindrucksvoll, dass virtuelle Realität ein den unterschiedlichsten Bereichen genutzt werden kann und ein immenses Potential bietet.

2.1.3. Probleme

Die technische Entwicklung der virtuellen Realität durch HMDs hat in den letzten Jahren einige Fortschritte gemacht. Doch gibt es einige Limitierungen momentaner VR-Brillen, welche das Erlebnis der virtuellen Realität trüben. Für Smartphone basierte Modelle gilt dies im Besonderen. Zwar bieten sie einen kostengünstigen Einstieg in die virtuelle Realität, doch steht ihnen nicht genügend Rechenleistung zur Verfügung, um an die Werte eines High-End Gerätes heranzukommen. Dafür muss bei der Anschaffung der High-End Geräte neben dem an sich schon teuren Preis von bis zu 899 USD zusätzlich noch ein leistungsstarker Rechner zur Verfügung gestellt werden. Neben der hohen Anschaffungshürde haben High-End Modelle weiter Limitierungen. Die Auflösung pro Auge wurde zwar seit den Anfängen der VR-Brille stets verbessern und liegt nun meist bei 1080x1200 Pixeln. Dennoch ist die Auflösung nicht ausreichend genug um zu verhindern, dass einzelne Pixel zu erkennen oder sehr kleine Details darstellbar sind [Fel16]. Des Weiteren ist die “Simulator Sickness“ immer noch ein Problem bei VR-Anwendungen. Durch Unterschiede in der Wahrnehmung des visuellen und vestibulären Systems kann es zu Schwindel und Übelkeit bei manchen Personen kommen [CMR+16]. Des Weiteren ist der Vergenz-Akkomodation-Konflikt immer noch ein Problem der HMDs. Wird Dreidimensionalität auf einem zweidimensionalen Bildschirm suggeriert kann es zu Unstimmigkeiten zwischen der Akkomodation der Augenlinse auf den Bildschirm und Vergenz der Augenmuskeln auf das dreidimensionale Objekt, welches

sich scheinbar vor oder hinter dem Bildschirm befindet kommen. Auch dies kann zu Schwindel und Übelkeit führen [Kra16]. Zusätzlich gibt es auf Seiten des Komforts Verbesserungspotential für die Head-Mounted Displays. So ist für die Verwendung von High-End Modellen eine Kabelverbindung zum Rechner nötig. Grade bei der HTC Vive, die eine Bewegung im Raum zulässt, kann sich diese Kabel als hinderlich erweisen. Zudem spielt gerade bei längeren Anwendungen der Tragekomfort und das Gewicht der Brille eine entscheidende Rolle.

2.1.4. Google Cardboard

Das Google Cardboard ist eine Halterung für Smartphones, mit der diese als VR-Brillen verwendet werden können. Das Cardboard besteht, wie der Name vermuten lässt, meistens aus Pappe, es gibt mittlerweile aber auch Varianten aus anderen Materialien wie zum Beispiel Plastik. Ein Cardboard aus Pappe, wie es im Google Shop zu finden ist, kann der Abbildung 2.1.1 entnommen werden.
Zur Verwendung des Cardboards wird auf dem Smartphone eine VR-Applikation gestartet und in die Klappe des Cardboards gelegt. Anschließend kann sich das Cardboard vor das Gesicht gehalten werden. Dabei wurde bewusst auf eine Kopfhalterung verzichtet, da der Nutzer das Cardboard automatisch in einem solchem Abstand hält, dass ein scharfes Bild zu sehen ist.

Abbildung 2.1.1.: Ein Google Cardboard der Version 2 [Goo17c]

2.1.5. Unity

Unity ist eine 3D-Engine für Multi-Plattform Spielentwicklung. Als solches ist auch Unity ein Werkzeug, das im Hintergrund vieler Spiele arbeitet und dort viele Aufgaben übernimmt, wie zum Beispiel das Rendering der für den Spieler sichtbaren Objekte und Landschaften. Um dies zu erreichen nutzt Unity zum einen Just-in-time-Kompilierung, wodurch der Programmcode erst direkt vor seiner Ausführung kompiliert wird.
Als Entwicklungsumgebung wurde in Unity MonoDevelop genutzt, welches mittlerweile jedoch von Microsofts VisualStudio abgelöst wurde. Innerhalb der Entwicklungsumgebung werden die von Unity genutzten Skripte geschrieben. Zusätzlich sind einige Bibliotheken in Unity integriert, um den gebotenen Funktionsumfang zu erweitern. Beispiele dafür sind Nvidias PhysX Physics-Engine, OpenGL und DirectX für das Rendering und OpenAL für Audio. [Gol09]
Besonderes wurde bei der Entwicklung von Unity darauf Wert gelegt, dass neuen Entwicklern ein möglichst einfacher Einstieg geboten wird. Dies erreicht Unity durch ihren graphischen Editor, mit dem der eigentliche Programmcode der Engine im Hintergrund bleibt und der Nutzer alles über graphische Oberfläche steuern kann. [Gol09]
Des Weiteren bietet Unity Unterstützung für insgesamt 27 verschiedene Systeme, darunter alle gängigen Desktop-, Mobil-, VR/AR-Umgebungen, sowie alle Spielkonsolen der momentanen Generation, Webbrowser und diverse Smart-TVs [Tec13d].

Grundkonzepte von Unity

Um den Einstieg in Unity relativ einfach zu halten, wurden einige Grundkonzepte bereits vordefiniert, die für die Erstellung von dreidimensionalen Umgebungen benutzt werden können [Gol09].

Szenen In Unity sind Szenen die Grundgerüste in denen virtuelle Objekte angelegt werden. Diese können dafür genutzt werden um Level oder Menüs in Spielen darzustellen [Gol09].

Assets Assets sind in Unity alle möglichen externen Ressourcen, dies können importierte 3D-Modelle, Sound-Dateien, Bilder, etc. sein [Gol09].

GameObjects Jedes virtuelle Objekt innerhalb einer Szene wird in Unity als GameObject bezeichnet. Jedes dieser GameObjects enthält dabei direkt eine Transform-Komponente, über die die Positionierung, Rotation und Skalierung des Objektes festgelegt wird. Diese Komponente des Objektes kann dann in Skripten genutzt

werden, um zum Beispiel die Position des GameObjects zu manipulieren [Gol09] [Tec13a].

Komponenten Komponenten in Unity dienen dazu einem GameObject einfach und direkt um neue Funktionen der Engine zu erweitern. Beispiele dafür sind Lichtquellen und Kameras. Auch eigene erstellte Skripte werden in Unity als Komponente betrachtet [Gol09] [Tec13c].

Skripte Durch Skripte können in Unity eigene Funktionalitäten definiert werden. Skripte selbst können dabei in C# oder JavaScript geschrieben werden. In den neueren Unity Versionen hat sich, auch aufgrund des Wechsels von MonoDevelop zu VisualStudio, C# als Standardsprache durchgesetzt. Durch diese Skripte ist es in Unity möglich, selbst Spiele und somit Szenen zu programmieren, ohne die darunterliegende Programmierung von Unity zu kennen [Gol09].

Interface des Unity Editors

Zum Erstellen von Szenen bringt Unity einen eigenen Editor mit. Das Hauptfenster des Editors ist in Abbildung 2.1.2 zu sehen und lässt sich in vier Hauptbestandteile einteilen. Diese sind [Tec13b]:

- **Scene View:** In dieser View kann man durch die erstellte Szene navigieren und diese auch bearbeiten.
- **Hierachy Window:** In diesem Bestandteil werden alle GameObjects samt ihrer Komponenten aufgelistet.
- **Inspector Window:** Erlaubt es die Eigenschaften der Komponenten des ausgewählten GameObjects einzusehen und diese zu ändern.
- **Project Window:** Zeigt die zur Verfügung stehenden, importierten Assets an.

2.1.6. Google VR SDK

Da es zum Anspruch dieses Projektes gehört dem Nutzer einen möglichst leichten Einstieg bei der Implementierung eigener Applikationen zu ermöglichen, liegt es nahe auf bestehende Software zurück zu greifen, die dem Nutzer möglichst viel Arbeit abnimmt. So soll sich der Nutzer beispielsweise im Kontext von virtueller Realität nicht selber um ein Tracking der Kopfbewegungen oder die Darstellung eines stereoskopischen Bildes kümmern müssen. Die Software sollte es zudem

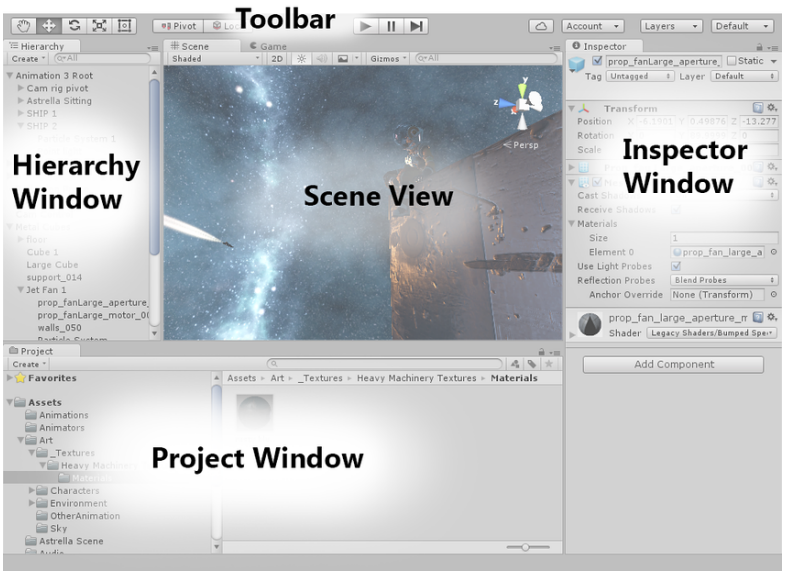

Abbildung 2.1.2.: Das Interface des Unity Editors [Tec13b]

ermöglichen, Applikationen für den Betrieb auf einem Smartphone zu erstellen. Wie bereits erwähnt sind Smartphones für den schnellen und kostengünstigen Einstieg in die virtuelle Realität ideal.
Google stellt Entwicklern mit dem Google VR SDK für Unity Werkzeuge zur Erstellung eigener VR Applikationen zur Verfügung. Das Software Development Kit (SDK) erlaubt es in Kombination mit der Unity Engine Anwendungen der virtuellen Realität für Smartphones mit Android Betriebssystem oder iOS zu erstellen. Zudem nimmt die Software dem Nutzer einige Probleme ab, die es bei der Erstellung von VR Applikationen zu beachten gibt. Durch das im Smartphone integrierte Gyroskop wird ein Tracking der Kopfbewegungen ermöglicht und dem Nutzer eine Orientierung im Raum gegeben. Zusätzlich kann der Nutzer durch Controller und andere Trigger mit der Software interagieren. Für die Darstellung der virtuellen Realität wird die in der Unity Engine erstellte virtuelle Umgebung in Form von zwei Bildern auf dem Smartphone Display dargestellt. Die Bilder werden aus leicht unterschiedlichen Winkeln aufgenommen und erlauben dem Nutzer ein stereoskopisches Wahrnehmen der Szene. Die Krümmung des Bildes durch eine Linse, wie sie üblicherweise in Head-Mounted Displays verwendet werden, wird dabei bereits von Googles Software herausgerechnet. Auch eventuelle Verzerrungen der Gyroskopdaten werden nach eigenen Angaben bereits berücksichtigt. [goo17a]
Aufgrund dieser Vorzüge und einer breiten Unterstützung der Entwickler in Form von Anleitungen und Foren, fiel unsere Wahl auf dieses SDK als Grundbaustein für die Entwicklung von Anwendungen der virtuellen Realität.

2.2. Mixed Reality

Neben Virtual Reality und Augmented Reality gibt es weitere Varianten, die die Aspekte der virtuellen und realen Realitäten vermischen. Das Spektrum dieser Realität bis hin zur Virtualität ist in Abbildung 2.2.1 dargestellt. Durch das Kombinieren ergeben sich weitere Möglichkeiten für den Einsatz in verschiedenen Anwendungsgebieten. Diese hybride Realität, auch Mixed Reality (MR) genannt, wird in diesem Abschnitt beschrieben.

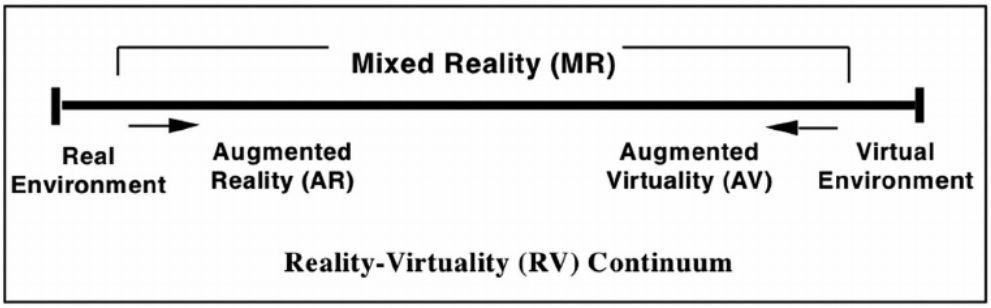

Abbildung 2.2.1.: Milgram Reality-Virtuality Continuum [mil17]

2.2.1. Der Begriff Mixed Reality

Wie der Begriff Mixed Reality vermuten lässt, ist es eine Mischung aus echter Realität und virtueller Realität. Jedoch existiert keine klare Definition von Mixed Reality und oft werden AR und MR einheitlich verwendet. Eine verbreitete Definition stammt von Milgram et al. aus dem Jahr 1995 und besagt, dass der Begriff alles in den Bereich zwischen den beiden Realitäten umfasst und somit auch verschiedene Abstufungen besitzt. So ist Augmented Reality eine Abstufung der Mixed Reality, und die sogenannte Augmented Virtuality (dt. erweiterte Virtualität) ist eine weitere Abstufung. Bei der Augmented Virtuality überwiegt der virtuelle Anteil den reellen Anteil, d.h. es sind z.B. reale Objekte in einer virtuellen Welt eingebunden. Das Hauptaugenmerk liegt hierbei mehr auf den virtuellen Objekten als auf den reellen Objekten (vgl. [DBGJ13a], [Bru03]).
Die Idee von Mixed Reality ist unter anderem die Interaktion zwischen reellen Objekten oder Personen mit virtuellen Objekten oder Personen. Für die Realisierung muss wie bei Virtual Reality nicht nur der Nutzer, sondern auch seine Bewegungen und Gesten erkannt werden, die entsprechend auf das virtuelle Objekt übertragen werden müssen. Hierbei kommen ebenfalls Tracking-Techniken wie z.B. mittels eines Datenhandschuhs zum Einsatz. Für die Visualisierungen können die im Kapitel

2.3 genannten AR-Techniken angewandt werden, wobei die See-Through Techniken am geeignetsten sind.

2.2.2. Anwendungsbereiche

Mixed Reality besitzt ein großes Potential für verschiedene Anwendungsbereiche, die in Zukunft den Alltag stark prägen werden. Dennoch ist Mixed Reality noch eine recht junge Wissenschaft, bei der noch Durchbrüche bevorstehen. Es existieren bereits mehrere Projekte, die die Forschung vorantreiben, um die Zukunftsvision wahr werden zu lassen. Im Folgenden wird die mögliche Nutzung von Mixed Reality in verschiedene Anwendungsbereiche kurz vorgestellt.

Mixed Reality in der Industrie

In der Industrie wird Mixed Reality bereits unter anderem in der Produktion und Logistik genutzt. Da oftmals große Produktionsanlagen, Maschinen als auch die Ausfallzeiten bei einem Umbau sehr teuer sind, ist es umso ärgerlicher, wenn die Produktionsanlagen nicht wie geplant arbeiten. Mit Mixed Reality Programmen kann im Vorfeld eine Anlage übersichtlicher geplant und simuliert werden, sodass Probleme und Komplikationen im Vorfeld erkannt und die Anlage angepasst werden kann. Dabei liegt der Schwerpunkt in der Handhabung bei der Planung, dies kann durch MR-Techniken und z.B. Modellblöcken intuitiv gestaltet sein. (vgl. [Bru03])
Allgemein lässt sich durch MR Daten visuell gut darstellen und so Analysen von Prozessen oder Produkten einfacher gestalten, was dem Betrieb Zeit und Kosten spart.

Mixed Reality in der Medizin

In der Medizin kann Mixed Reality Ärzte und Chirurgen bei der Diagnostik und Chirurgie helfen. Gesammelte Daten von Patienten von z.B. Röntgen, Computertomographie (CT) oder Magnetresonanztomographie (MRT) können in einem 3D-Modell anschaulich dargestellt und analysiert werden. Chirurgen können bei Eingriffen durch zusätzliche Informationen sowie visuelles Feedback unterstützt werden. Es könnten Organe, Knochen und Blutbahnen auf den Körper projektiert werden, sodass der Chirurg nicht mehr ganz blind operieren muss.
Auch bei Therapien kann Mixed Reality angewendet werden. Durch MR-Spiele können Patienten, die z.B. einen Schlaganfall erlitten haben, bei ihrem Training zur Rehabilitation motiviert werden. Der Vorteil bei Mixed Reality basierenden Spielen ist die intuitive Steuerung, da alltägliche Bewegungen zur Steuerung genügt. Auch

psychologisch erkrankte Patienten können mit z.B. gegen ihre Angststörungen durch Mixed Reality behandelt werden, indem sie gefahrlos ihre Ängste begegnen können. (vgl. [Shu11])

Mixed Reality in der Bildung

Mixed Reality eignet sich ebenfalls im Bereich der Bildung. In der Schule könnten Themen für die Schüler in den verschiedenen Fächer interessanter und interaktiver gestaltet werden. In Geschichte könnten vergangene Ereignisse im Klassenraum abgespielt werden, in Chemie können verschiedene, auch gefährliche Experimente problemlos und ungefährlich durchgeführt werden, in Physik können Schaltkreise mit verschiedenen Maschinen oder verschiedene physikalische Gesetze visuell simuliert werden und in Biologie kann die Anatomie von Menschen und Tieren in 3D-Modellen dargestellt werden. Dabei können Schüler bei Interesse und Fragen bestimmte Regionen oder Abschnitte genauer betrachten. (vgl. [Bru03])
Dieses Konzept eignet sich auch für Museen. Informationen zu Exponaten können direkt eingeblendet werden, die Exponate könnten selbst zum Leben erwachen und ihre Geschichte erzählen, oder es können verschiedene Zeitperioden in Hallen simuliert werden und so die Neugier der Besucher fördern, sodass diese mehr über das Thema wissen möchten.

Mixed Reality in der Ausbildung und im Beruf

Auch in der Ausbildung können Auszubildende mit Mixed Reality reelle Arbeitssituationen in einer Simulation üben und sich so auf schwierige sowie kritische Situationen vorbereiten. Mit z.B. Datenhandschuhen und Modellbauteilen können Auszubildende an Schalttafeln Schaltungen erstellen und testen ohne Gefahr zu laufen Geräte im Fall einer Fehlverknüpfung zu zerstören. (vgl. [Bru03])
Auch in einigen Berufen kann MR-Anwendung Unterstützung leisten, z.B. in der Architektur. Der Architekt kann seinem Kunden das entworfene Haus als 3D-Modell darstellen und bei Bedarf einzelne Räume vergrößern und so den Kunden nachvollziehbarer vorführen. Hier kann der Kunde auch Änderungswünsche anmerken, die gleich im 3D-Modell per Hand umgesetzt werden können. Das gleiche Prinzip kann auch beim Möbelkauf oder anderen handwerklichen Bereichen angewendet werden.

Mixed Reality in der Unterhaltung

Im Bereich der Unterhaltung bietet Mixed Reality ebenfalls viele Möglichkeiten zur Anwendung. Es können neuartige Spiele entwickelt werden, bei denen der Spieler nicht mehr nur einen Controller oder Maus und Tastatur bedienen muss, sondern

sich aktiv bewegen muss. Auch Filme und Bücher können interaktiver gestaltet werden. So kann der Zuschauer mitten im Film sein und gar die Handlung durch seine Interaktion beeinflussen.
Eine andere Möglichkeit ist die Internettelefonie. Diese kann nicht mehr nur durch Webcams und Mikrofone realisiert werden, sondern durch virtuelle Räume und Hologramme, sodass die Nutzer das Gefühl haben mit der anderen Person im gleichen Raum zu sitzen und sich mit ihr zu unterhalten. Eine weitere Möglichkeit mit ähnlichem Prinzip ist das Erkunden entfernter Länder im eigenen Wohnzimmer.

2.3. Augmented Reality

AR ist eine Variation von VR. Im Gegensatz hierzu, wird keine künstliche Umgebung, in der der Anwender agieren kann, geschaffen, sondern die reale Welt wird mit digitalen oder computergenerierten Informationen ergänzt. Dabei kann es sich um Bilder, Audio- oder Videomaterial oder haptische Eindrücke handeln, die in die reale Welt integriert sind oder sie überlagern. Aus technischer Sicht kann AR alle fünf Sinne beeinflussen, jedoch wird sich heutzutage meist auf den visuellen Eindruck beschränkt.
Ein oft genanntes Beispiel ist ein Head-up-Display (HUD), wie ihn Kampfjetpiloten während ihres Flugs tragen. In Abbildung 2.3.1 ist ersichtlich, dass viele Informa-

Abbildung 2.3.1.: Head-up-Display eines Kampfjetpiloten (vgl. [KR12], S.2)

tionen, die ein „normaler“ Pilot von analogen Geräten ablesen würde, direkt im Sichtfeld eingeblendet werden und somit dauerhaft verfügbar sind. Dies spart im

Einsatz zwar nur Sekunden, aber diese könnten über Leben und Tod entscheiden. (vgl. [KR12], S. 1-2)
Zusammenfassend müssen drei wesentliche Eigenschaften vorliegen, damit es sich um Augmented Reality handelt.

- Augmented Reality schafft eine Verbindung zwischen virtuellen und realen Inhalten.
- In Augmented Reality wird in Echtzeit interagiert.
- Augmented Reality hat eine Kopplung im dreidimensionalen Raum.

Als Kombination von verschiedensten Technologien bringt AR digitale Informationen jeglicher Art in Echtzeit ins Sichtfeld des Anwenders. Aus diesem Grund ist AR, je nach Gesichtspunkt, als Technologie, Forschungsfeld, eine Vision zukunftsweiser Datenverarbeitungsmöglichkeiten, eine aufstrebende kommerzielle Industrie und ein neues Medium kreativen Ausdrucks zu verstehen. Genau diese Punkte wurden interessanterweise in den 1980er auch im Zusammenhang mit der grafischen 2D-Benutzeroberfläche genannt. (vgl. [KR12], S. 4-5)

2.3.1. Video-see-through

Ein Augmented Reality Display, welches auf Video-Input basiert, verarbeitet ein digitales Signal um virtuelle Bilder mit der aufgenommenen Realität zu verbinden. Diese Art von Displays digitalisiert zuerst die aufgenommene Szene um danach die digitalen Bilder einfügen zu können. Abbildung 2.3.2 zeigt die übliche Struktur eines solchen Systems.
Oftmals ist die Kamera auf der Rückseite des Displays angebracht, sodass das Sichtfeld des Nutzers aufgenommen wird. Dies erzeugt die Illusion, durch den Bildschirm, in die reale Welt zu schauen (daher auch video-see-through) (vgl. [BCL15], S. 128-129).

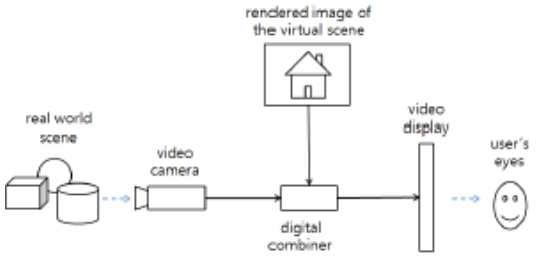

Abbildung 2.3.2.: Struktur video-basierter AR-Displays (vgl. [BCL15], S.129)

2.3.2. Optical-see-through

Optical see-through AR-Displays nutzen ein optisches System um die reale Welt mit virtuellen Bildern zu ergänzen. Diese Art von Systemen verwendet dabei meist Strahlteiler (wie z.B. halbdurchlässige Spiegel oder Prismen). Die durch den Strahlteiler gesehene reale Welt wird über die Spiegelung eines Videos mit den virtuellen Informationen ergänzt. Abbildung 2.3.3 stellt die Struktur dar. Ein Beispiel ist ein HUD, das oft in Flugzeugcockpits und modernen Autos verarbeitet sind.
Im Gegensatz zu video-basierten AR-Displays, bietet die optische Lösung eine direkte Sicht auf die reale Welt, die nicht durch z.B. Auflösung oder zeitliche Verzögerung gestört ist. Trotzdem ist eines der Hauptprobleme von optical-see-through Anwendungen die fehlende Genauigkeit der platzierten virtuellen Bilder. Außerdem ist die korrekte Tiefenwahrnehmung der virtuellen Bilder und der realen Welt schwer miteinander zu kombinieren. (vgl. [BCL15], S. 133-135)

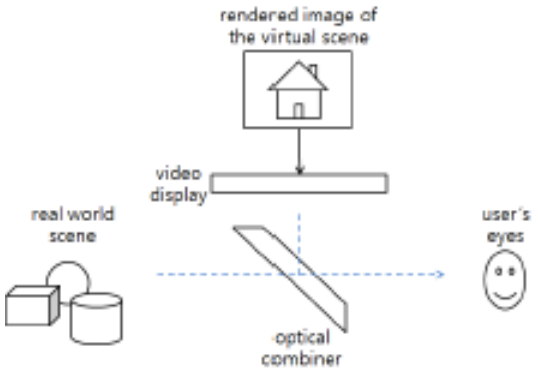

Abbildung 2.3.3.: Struktur eines optischen AR-Displays (vgl. [BCL15], S.134)

2.3.3. Vuforia

Vuforia ist eines der am weitesten verbreitetsten AR-Toolsets mit weltweit über 300.000 Entwicklern, vertreten in über 35.000 Applikationen mit über 400 Millionen Installationen und gehört zu der Firma Parametric Technology Company.
Vuforia ermöglicht es, 3D Inhalte in physische Umgebungen zu integrieren. Der Kern ist hierbei die Vuforia-Engine, die mithilfe von Funktionen aus dem Bereich der Computervision, die Möglichkeit bietet, Objekte zu erkennen und Umgebungen zu rekonstruieren. Über das Vuforia SDK können auch VR-Applikationen ermöglicht werden. Außerdem werden verschiedene Targets unterstützt (Image Targets, Cylinder Targets, Multi Targets). Die Parametric Technology Company hat sich zudem 2016 mit Unity darüber geeinigt, dass Vuforia zukünftig in Unity integriert ist citepeddie2017augmented.

2.3.4. ARToolkit

Das ARToolkit ist ein Werkzeug für die Entwicklung von Augmented Reality Anwendungen und wurde von Dr. Hirokazu Kato von der Universität Osake entwickelt. Das ARToolkit steht unter einer Open-Source-Lizenz und ist dementsprechend frei zugänglich. Zu den zentralen Bestandteilen gehören das Verfolgen des Zuschauerblickpunktes mittels eines speziellen Algorithmus, der in Echtzeit die Kameraposition in Relation zu einem optischen Marker berechnet. Zusätzlich werden Kalibrierung von Kameras und das Erkennen von physischen Marker vom ARToolkit bereitgestellt. [Taz07] In der Version 5.x ermöglicht das ARToolkit, unter anderem, das simultane Verfolgen von multiplen Videoquellen, sowie die generelle Unterstützung für Verarbeitung und Kalibrierung von Stereo-Kameras. Des Weiteren können mit dem Toolkit eine Vielzahl an Video-Input-Systemen, von der Windows Plattform über Android bis hin zum Apple-Ökosystem, genutzt werden. U.a. kann das Toolkit in Kombination mit den Programmiersprachen und APIs von C++, CSharp, Objective C und Java verwendet werden. Für Grafik und Rendering werden die Unity Engine sowie OpenSceneGraph unterstützt. Als Entwicklungsumgebungen können viele aktuelle Umgebungen, wie Eclipse, Visual Studio 2013 oder Xcode 6.x. eingesetzt werden. [DAQ17]

2.4. Eye-Tracking

Beim Eye-Tracking handelt sich um eine Art Augen- bzw. Blickverfolgung sowie gleichzeitiger Messung der Aktivität der Augen. Die Aktivität kann durch die folgenden Punkte evaluiert werden: [Koe09]:

- Was wird angesehen und wie lange?
- Was wird dabei ignoriert?
- Wann wird geblinzelt?
- Wie reagiert die Pupille auf verschiedene Reize?

Beim Eye-Tracking gibt es natürliche Interaktionen, Fixationen und Sakkaden. Bei der Fixation wird ein beliebiger aber bestimmter Punkt fokussiert und visuell fixiert [Koe09]. Die Fixation wird in vielen Techniken verwendet. Dabei kann der verwendete Eye-Tracker ein stationäres System (remote system) oder ein am Kopf getragenes System (head-mounted) darstellen. Eine Sakkade ist der Wechsel von einer Fixation zur nächsten.
Das Einsatzgebiet des Eye-Tracking ist breit gefächert. Von Webseiten, die mittels Eye-Tracking auf ihre Benutzerfreundlichkeit untersucht werden, bis zur Forschung in der Medizin oder dem Kaufverhalten von Kunden. Der große Vorteil des

Eye-Tracking ist die Echtzeit Messung der Augenaktivität sowie die gute Auswertbarkeit dieser Ergebnisse. Bei der Anwendung von Eye-Tracking Systemen sind folgende drei Schlüsselfaktoren wichtig [BG10]:

- Genauigkeit des Eye-Trackings
- Kalibrierungsverlauf
- Vermeidung des "Midas Touch" (vgl. 2.4.2)

2.4.1. Monokular vs. Binokular

Eye-Tracking kann sowohl monokular wie auch binokular durchgeführt werden. Diese beiden Verfahren werden in den folgenden beiden Abschnitte voneinander abgegrenzt.

Monokulares Eye-Tracking

Beim monokularem Eye-Tracking wird ein Auge für das Tracken der Augendaten verwendet. Dafür wird eine Eye-Tracking Kamera für das Tracken des Auges benötigt. Diese wird in dem Head-Mounted System oder einem Remote Systeme befestigt. Das zweite Augen hat somit keine Auswirkungen auf das Tracking, wird aber auch nicht verdeckt oder ähnliches. Die Szenen sowie das Kalibrieren wird auf das jeweilige Auge angepasst und eingestellt. Vorteil des monokularen Eye-Trackings ist zum einen der geringere Modifikationsaufwand beim Platzieren der Kamera und zum anderen ist der Einbau schneller. Die Ergebnisse die durch das Tracking geliefert werden, sind genauso repräsentativ wie die des binokularem Eye-Trackings.

Binokulares Eye-Tracking

Beim binokularem Eye-Tracking werden beide Augen gleichzeitig für das Tracken der Augendaten verwendet. Dementsprechend werden hierfür zwei Eye-Tracking Kameras benötigt und in das ausgewählte System montiert. Durch das Tracken beiden Augen sind die Ergebnisse des Trackings genauer als bei monokularem Eye-Tracking. Zudem ist der Befestigungsaufwand der Kameras etwas aufwendiger als der des monokularem Eye-Trackings. Für aufwendigere Forschungsansätze bietet das binokulare Eye-Tracking einen besseren Standard, da wenn beiden Augen zuverlässig getrackt werden alle Extrempunkte abgedeckt werden können.

2.4.2. Interaktionstechniken

Interaktion

Eine Interaktion kann zwischen verschiedensten Dingen stattfinden. In diesem Fall handelt es sich um eine "Human-Computer-Interaction". Diese wird auch häufig mit Kommunikation gleichgesetzt, denn der Mensch kommuniziert mit einem System. Interaktion zwischen Mensch und Maschine kann auf verschiedene Art und Weise stattfinden. Der Mensch bietet mit seinen Sinnen viele Möglichkeiten an mit Maschinen zu kommunizieren. Zusätzlich besteht die Möglichkeit die unterschiedlichen Sinne miteinander zu kombinieren, um weitere Interaktions- bzw. Kommunikationsmöglichkeiten zu erhalten. Die Interaktionstechniken, die im Folgenden näher beschrieben werden, befassen sich mit dem "visuellen Interagieren" zwischen Menschen und Systemen.

Gaze Horizontal

Bei Gaze Horizontal handelt es sich um spontanes blickgesteuertes Interagieren mit öffentlichen Anzeigen. Das System erkennt die Anwesenheit des Passanten und bietet ihm die Möglichkeit die Inhalte der Anzeige zu navigieren. Das System gibt den Passanten Instruktionen wie dieser mit dem System interagieren soll. Der Fokus dieser Technik liegt auf dem horizontalen Scrollen der Inhalte durch Augenbewegungen und Fokus.

- Das Gesicht wird verfolgt, wenn es in der richtigen Region positioniert ist und wenn der Gesichtsbereich größer als 200x200 Pixel ist.
- Augenbildmerkmale werden extrahieren und mit Bildverarbeitungstechnologie von Zhang angewendet (vgl. [ZMC+14])
- Daraufhin wird dann der Pupil-Canthi-Ratio berechnet (Blick geradeaus) (vgl. [ZMC+14])

Smart Scrolling/Vertikales Scrollen

Die Methode beschäftigt sich mit der Verfolgung der Augenbewegungen und das daraus resultierende Scrollen. Dafür müssen zunächst die Augen lokalisiert werden und die Bewegung dieser erfasst werden [LPA13].

- Für die Lokalisierung der Augen wird die Technik Six-Segmented Rectangular (SSR) für die Gesichtserkennung angewandt. Dafür wird die Partie um die Augen in sechs gleichgroße Sektoren aufgeteilt. Daraus resultiert ebenfalls die Lokalisation der Pupille.

- Die Between-the-eyes Methodik verwendet auf jeder Seite des Gesichts eine Region von 14x12 Pixel, auf der nach dem dunkelsten Punkt gesucht wird.
- Ist das Auge bzw. die Pupille erkannt kann ein Clustering Algorithmus angewendet werden.
- Dafür wird der Blick gerade gehalten und virtuell eine Linie durch das Auge gelegt, woraufhin der Algorithmus berechnet auf welcher Seite sich mehr Pixel befinden um die Scrollrichtung zu bestimmen.

Smooth Pursuit

Bei dieser Technik folgen die Augen den Objektbewegungen auf einem Bildschirm. Ein Benutzer agiert hier instinktiv ohne weitere Instruktionen [VBG13].

- Dafür werden sowohl horizontalen und vertikalen Augendaten erhoben und mit horizontalen sowie vertikalen Positionen der Objekte kombiniert.
- Die Berechnung der Begegnung von Blick und Objekt werden mit dem Korrelationskoeffizienten von Pearson berechnet.

Orbits

Hierbei handelt es sich um eine Technik in der sich der Benutzer für einen Inhalt einer Anzeige entscheidet. Die Objekte haben als Auswahleingabe einen Cursor, der sich kreisförmig um das Objekt herumbewegt (Orbit). Der Nutzer entscheidet sich für ein Objekt indem er den Orbit des gewählten Objektes mit seinen Augen verfolgt. Während der Nutzer den Orbit mit den Augen verfolgt kann über zusätzliches visuelles Feedback der Auswahlprozess unterstützt werden. So kann der Orbit beispielsweise Farbe oder Form ändern. Vorteil dieser Methode ist zudem der geringe Kalibrierungsaufwand, da nur der relative Vergleich zwischen Orbit und Augenbewegungen nötig ist. Zudem könnte diese Technik auch zu Kalibrierungszwecken verwendet werden [GKR15].

Gesten

Interagieren über Gesten der Augen durch Muster, die genutzt werden können um bestimmte Aussagen mit diesen zu treffen. Dafür gibt es verschiedenste Muster, die ein Proband nutzen kann, um zu interagieren. Die Muster können für Ausführungen wie zum Beispiel “zurück“, “öffnen“ oder “schließen“ stehen [VB12].

Midas Touch

Wäre es nicht einfach den Mauscursor gegen die Augenposition auszutauschen? Sehen wir nicht sowieso immer dorthin wo der Cursor der Maus ist. Dies ist laut Robert J.K. Jacob der wohl naivste Ansatz, die Augenposition zu verwenden [Jac90]. Die Navigation mittels der Augen ist für die meisten Menschen nicht alltäglich. Daher muss sich der Nutzer bewusst werden, dass er mit seinem Blick ununterbrochen Informationen an das System übermittelt. Das heißt, schweift der Blick hin und her, werden mal hier und mal dort Befehle ausgeführt. Dies führt nicht nur bei der bedienenden Person zu Unklarheiten, auch das System mit dem interagiert wird, kann nicht alles ausführen, was mit dem Blick fixiert wird. Das Problem wurde von Robert J.K. Jacob als das "Midas Touch Problem" benannt. Deswegen ist es auch so wichtig bei der Auswahl und Erstellung bzw. Anwendung von Interaktionstechniken darauf zu achten, das "Midas Touch Problem" zu vermeiden. Unter Berücksichtigung des Problems kann es erst gar nicht zu einer solchen Überforderung aller Beteiligten kommen. Im Best Practice Fall ist der Bediener der Chef und entscheidet wann die Eingabe und Nutzung der Augen "aktiv" ist und wann nicht.

2.4.3. Pupil Capture

Bei Pupil Capture handelt es sich um eine open source Eye-Tracking Software, die frei zugänglich und unter Windows, Linux sowie MacOS läuft. Durch den Faktor der Open-Source-Lizenz, kann diese jederzeit erweitert werden und individuell auf die Wünsche des Nutzers durch diesen angepasst werden. Pupil Labs bietet neben Capture zusätzlich noch Pupil Player, Pupil Service und Pupil Mobile an. Auf diese wird in dieser Ausarbeitung aber nicht weiter eingegangen.
Pupil Capture arbeitet mit zwei Ansichten, zum einen mit der „World View" (vgl. 2.4.1) und zum anderen mit der „Eye View" (vgl. 2.4.2). Dabei werden die jeweiligen Video-Streams abgegriffen. Durch diesen Video-Stream wird die Pupille getrackt, der Blick verfolgt und Marker erkannt sowie die Szenen der virtuellen oder erweiterten Realität übertragen. Des Weiteren arbeitet Pupil Capture mit Echtzeitdaten und bietet die Möglichkeit die „World View" aufzunehmen. Pupil Capture unterstützt sowohl monokulares als auch binokulares Eye-Tracking.
Pupil Capture bietet mit seiner guten Übersichtlichkeit und der einfachen Bedienbarkeit optimale Möglichkeiten individuelle Nutzeranpassungen vorzunehmen. Unter den verschiedenen Abschnitten der Einstellung können zum Beispiel die Kamera(s) ausgewählt oder die Wahl der Kalibrierungsmethode bestätigt werden. Pupil Capture bietet eine Vielzahl von Kalibrierungsmöglichkeiten, zum Beispiel Manual Marker oder Natural Feature Calibration.
Über die „Eye View" kann das ausgewählte Auge durch diverse Einstellungen

optimal für das Tracken bereitgestellt werden. Dafür wird zum Beispiel die Eingrenzung der Pupille und die Verwendung des Algorithmus definiert. Zudem kann die Pupillengröße und die Intensität angepasst werden.

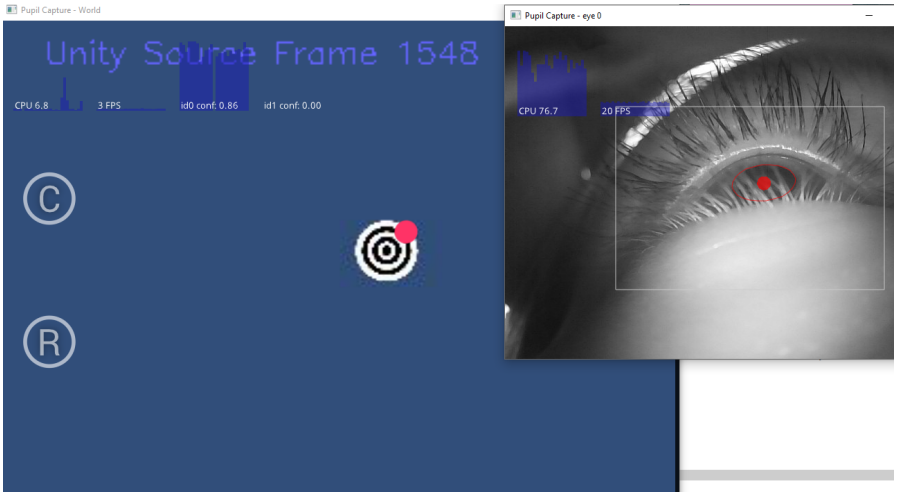

Abbildung 2.4.1.: Pupil Capture World View und Eye View

2.5. Optimierung von Objekten

Damit die Echtzeitbedingungen im VR/AR-System eingehalten werden können ist ein effizientes Rendering ein ausschlaggebender Faktor. Dieser kann durch die Reduktion bzw. Vereinfachung von komplexeren Objektgeometrien in großem Maße verbessert werden [DBGJ13b]. Deshalb behandelt dieser Abschnitt Optimierungsmöglichkeiten für 3D-Assets, wie die Vereinfachung von Polygonnetzen, Detailreduktion bei größerer Entfernung und Texture Baking.

2.5.1. Vereinfachung von Polygonnetzen

Eine wichtige Maßnahme, um die Echtzeitfähigkeit der 3D-Modelle zu unterstützen, ist die Reduktion der Polygonanzahl [DBGJ13b]. Hierfür kann z.B. das open source 3D-Modellierungs- und Animationswerkzeug Blender genutzt werden. Dieses steht kostenlos zur Verfügung und wird folglich den Anforderungen an dieses Projekt gerecht. In Blender ermöglicht der „Decimate Modifier“ die Anzahl an Oberflächen und Eckpunkten zu reduzieren, ohne die Form und Struktur des ausgewählten Objektes stark zu beeinflussen. Folglich kann die Polygonanzahl auf nicht-destruktive Art verringert werden. Anders als die meisten Modifier, ist es mit dem „Decimate

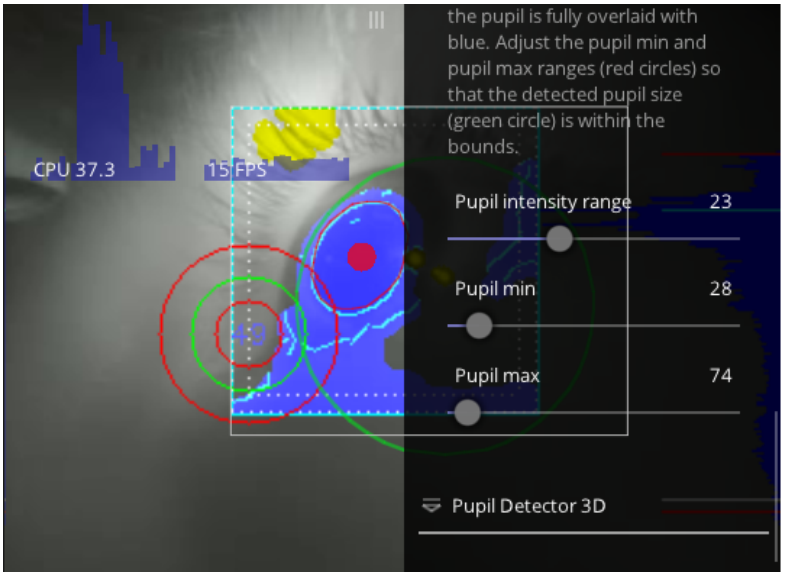

Abbildung 2.4.2.: Pupil Capture Eye View im Algorithmus Modus

Modifier“ jedoch nicht möglich, die gemachten Änderungen direkt im Edit-Mode darzustellen [Tea].

2.5.2. Darstellung unterschiedlicher Detailgrade

Je weiter sich ein 3D-Objekt vom Betrachter entfernt, desto weniger Details sind wahrnehmbar. Diese Gegebenheit lässt sich nutzen, um das Rendering des Objektes effizienter zu gestalten. Dabei wird ein 3D-Objekt in mehreren Detaillierungsgraden (Level of Detail) abgelegt [DBGJ13b]. Der Begriff Level of Detail bezieht sich also darauf, die Komplexität eines 3D-Assets zu variieren - je nachdem wie viel Platz dieses auf dem Bildschirm einnimmt [Hes12]. Die unterschiedlich detaillierten Objekte können z. B. durch die schrittweise Vereinfachung eines Polygonnetzes – wie zuvor beschrieben – entstehen [DBGJ13b]. Eine weitere Möglichkeit besteht in der Verwendung von unterschiedlich großen Texturen [KR06]. Zur Laufzeit wird dann - abhängig von der Distanz zum Betrachter - durch das VR/AR-System ein geeigneter Detaillierungsgrad ausgewählt [DBGJ13b].

2.5.3. Texture Baking

Wie bereits erwähnt ist es häufig notwendig, die Polygonanzahl eines hochauflösenden 3D-Assets zu reduzieren, um die bei einer interaktiven VR/AR-Anwendung obligatorischen Echtzeitanforderungen gerecht zu werden. Um dennoch den Eindruck

einer detaillierteren Darstellung zu erhalten, wird auf die Technik des Texture Baking zurückgegriffen. Beim Texture Baking wird die Farbinformation der beleuchteten Oberfläche eines hochauflösenden 3D-Modells in eine Textur gespeichert. Die so „gebackene“ Textur wird dann auf die niedrig aufgelöste, polygonreduzierte Modellversion übertragen [Vur13], [DBGJ13b].

3. Projektmanagement

In diesem Kapitel wird zunächst auf die Projektgruppe und die Rollenverteilung eingegangen. Anschließend werden die angewendeten Vorgehen Scrum und Human-Centered Design näher erläutert, woraufhin ein Überblick über den Verlauf der vier Quartale der Projektphase präsentiert wird, wobei Besonderheiten und Entscheidungen innerhalb der Projektgruppe genauer beschrieben werden.

3.1. Projektgruppe

Die Projektgruppe „Bull's Eye“ der Carl von Ossietzky Universität Oldenburg beschäftigt sich in ihrem Projekt mit der Kombination von Eye-Tracking, Augmented Reality und Virtual Reality. Das Ziel der Projektgruppe ist es, Eye-Tracking-Daten in AR- und VR-Applikationen nutzbar zu machen, sodass Interaktionstechniken auf einfacher und kostengünstiger Basis mit den Augen möglich sind. Die Projektgruppe bestand aus ursprünglich neun Masterstudenten, sank jedoch im Verlauf des ersten Quartal auf sechs. Ihre Mitglieder stammen aus den Studiengängen Informatik, Wirtschaftsinformatik und Eingebettete Systeme und Mikrorobotik. Von dieser Mischung verschiedener Studiengänge konnte die Projektgruppe im Verlauf des Projekts profitieren, da die Erfahrungen und das Wissen aus diesen Bereichen vorteilhaft für das Projekt genutzt werden konnte.

3.2. Rollen innerhalb der Projektgruppe

Im Folgenden werden die verschiedenen Rollen innerhalb der Projektgruppe aufgegliedert und näher beschrieben. Die Rollen wurden zu Beginn der Projektgruppe ermittelt und verteilt; weitere kamen im Verlauf der Projektgruppe zustande.
Der nachfolgenden Tabelle können die einzelnen Rollen, sowie die Person die diese ausgeführt hat, entnommen werden.

3.2.1. Rollen

Nachdem zunächst die Verteilung der Rollen dargestellt wurde, werden sie im nachfolgenden Abschnitt näher erläutert.

Rolle	Gruppenmitglied
Dokumentationsbeauftragte	Daniela
Webseitenbeauftragter	Henrik
Qualitätsmanagement	Aljoscha
Redmine-Beauftragter	Tim
Projektleitung	Rieke/Stefan
Git-Beauftragter	Stefan
ARBI-Beauftragter	Stefan
Social-Event-Beauftragte	Rieke/Aljoscha
Wissensmanagementbeauftragte	Rieke
Product Owner	Tim
Scrum Master	Henrik
HCD Master	Aljoscha
Hardwarebeauftragter	Henrik
Usability-Beauftragte	Daniela
Scrum Team	Alle
Stakeholder	Tim und Uwe

Tabelle 3.1.: Rollenverteilung

Dokumentationsbeauftragter

Jedes Projekt muss aus informativen sowie rechtlichen Gründen gut dokumentiert sein. Die Projektdokumentation muss Informationen über die Projektorganisation, Anforderungen, Konzeptentwürfe, Projektablauf und erreichte Ziele beinhalten bzw. wiedergeben. Der Dokumentationsbeauftragter ist für die Projektdokumentation verantwortlich. Er muss dafür sorgen, dass alle Dokumente klar strukturiert, einheitlich und fehlerfrei sind.

Webseitenbeauftragter

Die Aufgaben des Webseitenbeauftragten sind die Erstellung und Wartung der Projektwebseite im World Wide Web. Der Webseitenbeauftragter beschäftigt sich also mit dem Aufbau einer modernen und ansprechenden Webpräsenz. Weitere Aufgaben sind die Nutzerführung, das Interfacedesign und die Präsentation des Projektes nach außen.

Qualitätsmanagement

Qualitätsmanagement ist die systematische Ausrichtung, Steuerung und Kontrolle aller Aktivitäten der Projektgruppe in Bezug auf die Qualität. Qualitätsmanager

beschäftigen sich mit der Verbesserung der Prozessqualität/Leistungen, wodurch es zur Einhaltung von Standards und Normen kommt. Ziele sind u.a. die Arbeit zu verbessern und diese durch Effektivität und Effizienz zu stützen.

Redmine-Beauftragter

Verwaltung und Einrichtung des Redmine an die Gegebenheiten der Projektgruppe. Hierbei kümmert sich der Redmine-Beauftragte, neben der Einrichtung, auch um das Verwalten von Rollen und Rechten. Zudem muss dieser bei Bedarf geeignete Plugins finden und diese einbinden.

Projektleitung

Die Projektplanung bzw. das Projektmanagement beschäftigt sich mit der Organisation und Umsetzung eines Projektes. Der/Die Projektplaner/in ist für die organisatorische Leitung des Projekts zuständig und übernimmt das gesamte Projektmanagement. Er/Sie muss dafür sorgen, dass das Projekt einem konkreten Ablauf folgt.

Git-Beauftragter

Die Aufgaben des Git-Beauftragten ist die Erstellung und Wartung des Projekt-Repository, dazu agiert dieser als Ansprechperson bei Problemen mit Git. Dieser soll also durch sein Aufgabenfeld ein funktionales Git-Repository zur Verfügung stellen und allen Projektteilnehmern die Verwendung vereinfachen.

ARBI-Beauftragter

Die Aufgaben des ARBI-Beauftragten ist die Verwaltung des Gruppennutzers der Projektgruppe auf den Servern der Abteilung Rechner- und Netzbetrieb Informatik (ARBI), sowie die Wartung und Installation der verwendeten Software. Typische Aufgaben sind ein Gruppennutzer für die Projektgruppe einrichten, Software auf dem ARBI-Server installieren (z.B. Apache, MySQL, Redmine) und bei Bedarf Probleme mit installierter Software beheben.

Social-Event-Beauftragter

Social-Event-Management beschäftigt sich im Rahmen der PG Bull's Eye mit der Konzeption und Planung von Events außerhalb der Arbeitszeiten, mit dem Ziel den Zusammenhalt und die Motivation innerhalb der Projektgruppe zu fördern.

Wissensmanagementbeauftragter

Das Wissensmanagement beschäftigt sich mit der Instandhaltung einer Ansammlung (im Wiki) oder Datenbank der Artikel, Paper, Links etc., die in unseren Seminararbeiten verwendet wurden, sowie sämtliches, was von unseren Betreuern zur Verfügung gestellt wurde, als auch von anderen Gruppenmitgliedern als wichtig erachtet wurde.

Product Owner

Vertritt die Seite des Auftraggebers, überblickt Anforderungen und erstellt ein Backlog. Steht in Kontakt mit den Stakeholder, um Wünsche im Hinblick auf zusätzliche Funktionalitäten zu ermitteln. Er ist vereantwortlich für ein aktuelles und priorisiertes Backlog, an welches sich bei der Entwicklung orientiert werden kann. Interessen des Auftraggebers sollen vertreten werden und somit sollen aller Stakeholder Interessen in Einklang gebracht werden.

Scrum Master

Der Scrum Master ist verantwortlich für das Gelingen von Scrum. Der Scrum Master arbeitet dazu mit dem Entwicklerteam zusammen, führt Scrum-Regeln ein und Überprüft jener Einhaltung. Des Weiteren gehört auch das Beheben von Störungen und Hindernissen zu seinen Aufgaben. Im Normalfall ist der Scrum Master nicht Teil des Entwicklerteams, was sich aber im Rahmen der Projektgruppe nicht umsetzen lässt, sodass der Scrum Master bei uns auch Teil des Entwicklerteams ist. Der Scrum Master besitzt ein stetiges Wissen über den Scrum-Prozess. Hiermit kann der Scrum-Master überprüfen, ob das Vorgehen konform zum Scrum-Prozess ist. Wenn Aufgaben und Arbeitsweisen nicht der aktuellen Scrum-Phase entsprechen, liegt es in seiner Verantwortung einzugreifen und die Aufgaben der aktuellen Phase anzupassen.

HCD-Master

Der HCD-Master ist für die korrekte Umsetzung des Human-Centered-Design-Prozesses verantwortlich. Der Human-Centered-Design (HCD) Master besitzt ein stetiges Wissen über die aktuelle, sowie die vor- und nachgelagerte, Phase des HCD -Prozesses. Hiermit kann der HCD -Master überprüfen, ob das Projektvorgehen konform zum HCD -Prozess ist. Wenn Aufgaben und Arbeitsweisen nicht der aktuellen HCD -Phase entsprechen, liegt es in seiner Verantwortung einzugreifen und die Aufgaben der aktuellen Phase anzupassen.

Hardwarebeauftragter

Die Aufgaben des Hardware-Beauftragten sind die Bestellung, Abholung und Verwaltung, der für das Projekt benötigten Hardware. Der Hardware-Beauftragter ist für die Verwaltung und Überwachung des Hardwarebestands (im Projektschrank), sowie dem Abholen der bestellten Hardware und Bereitstellen relevanter Informationen zum Aufenthaltsort der Hardware verantwortlich.

Usability-Beauftragter

Usability ist ein häufig genutzter Begriff für die Gebrauchstauglichkeit eines Produkts und ist heutzutage Voraussetzung für den Erfolg eines Produkts. Um Usability sicherzustellen gibt es verschiedene Methoden und Verfahren, deren Evaluationen meist mit Personen aus der Zielgruppe durchgeführt werden. Der Usability-Beauftragter ist für die Usability-Tests des Produkts verantwortlich. Er muss dafür sorgen, dass alle Usability-Tests korrekt durchgeführt werden. Er muss darauf achten, dass die entwickelten Prototypen den Anforderungen erfüllen.

Stakeholder

Die Stakeholder sind in diesem Fall unsere Kunden und späteren Anwender (Wissenschaftler). Ziel für das Entwicklungsteam ist es, die Bedürfnisse und Ansprüche der Stakeholder genügsam umzusetzen.

3.3. Prozess-Human-Centered-Design

Dieser Abschnitt beschäftigt sich mit dem Prozess-Human-Centered Design. Dafür wird zum einen die Entscheidung für dieses Vorgehen erläutert, zum anderen wird es eine allgemeine Beschreibung des Prozesses geben, sowie der Aufbau der vier Phasen erläutert. Als Abschluss wird darauf eingegangen, wie wir den Prozess angewandt haben und welche Besonderheiten dabei auf uns zu kamen. Zudem wird das Zusammenspiel zwischen HCD und Scrum erörtert.

3.3.1. Entscheidung

Es wurde sich für den HCD-Prozess entschieden, da dieser eine nutzerorientierte Gestaltung zulässt. Im folgenden Kapitel werden die Vorteile des Prozesses beschrieben (vgl. 3.3.2). Der iterative Aufbau und die Möglichkeit, Stakeholder in den Prozess zu integrieren, waren Argumente für die Nutzung von HCD. Des Weiteren haben wir die Möglichkeit genutzt, den Human-Centered Design Prozess mit Scrum zu verbinden (vgl. 3.4). Die vier Phasen sind klar definiert, wodurch wir

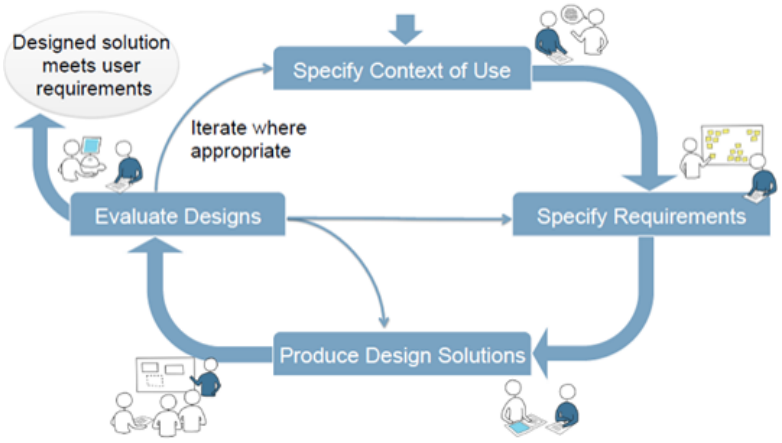

Abbildung 3.3.1.: HCD Prozess

eine gute Übersicht unseres aktuellen Standes zu jedem gewünschten Zeitpunkt erhalten konnten. Wie wir den HCD Prozess in unserem Projekt genutzt haben, wird im Kapitel 3.3.3 detailliert beschrieben.

3.3.2. Beschreibung

Der gesamte HCD Prozess (vgl. Abbildung 3.3.1) kann minimal einmal und maximal, so oft wie Defizite im System vorhanden, durchlaufen werden. Vorteile des HCDs sind [Gei10]:

- Iteratives Verfahren
- Potenzieller Nutzer steht im Mittelpunkt, wodurch Anforderungen besser umgesetzt werden können
- Mehrere Durchläufe möglich
- Nutzer agiert als Kontrolleur, ob seine Anforderungen entsprechend seinen Vorstellungen umgesetzt wurden

In den vier Phasen können verschiedene Methoden als Hilfe dienen, um bessere Ergebnisse zu erhalten; beispielsweise Interviews, Usability Tests, Use Cases, Personas, Online-Fragebögen, Paper-Prototyping oder Heuristiken [Elm].
Die vier Phasen werden in den nachfolgenden Abschnitten erläutert [Gei10] [Mag01].

Nutzungskontext

In dieser Phase geht es zunächst darum, die potentiellen Nutzer zu ermitteln. Hat man die potenziellen Nutzer erfasst, gilt es ihre Bedürfnisse in Bezug auf das zu erarbeitende System aufzustellen. Dazu gehören auch die Ziele, die ein zukünftiger Nutzer mit dem erarbeiteten Konzept hat, zu berücksichtigen. Des Weiteren sollten in der ersten Phase die Aufgaben, welche mit der Anwendung durchgeführt werden sollen, dokumentiert, die Umgebung für das System ermittelt und die Rahmenbedingungen festgelegt werden. Hat man sich mit all diesen Fragen beschäftigt, kann die nächste Phase erreicht werden.

Benutzeranforderungen

Die zweite Phase arbeitet mit den Ergebnissen der ersten Phase weiter und baut darauf auf. Hier gilt es Benutzeranforderungen zu ermitteln und zu definieren. Durch die Anforderungen der Nutzer kann der Prototyp, welchen es gilt in der nächsten Phase zu konzipieren, eingegrenzt werden.

Prototyping

Sind nun alle Informationen aus den ersten beiden Phasen zusammen, gilt es jetzt diese richtig umzusetzen und, gemäß den Anforderungen der potenziellen Nutzer, einen geeigneten Prototyp zu entwickeln. Dafür werden die aufgestellten Anforderungen in ein geeignetes Hardware- bzw. Softwarekonzept umgesetzt.

Evaluation

Die vierte Phase des HCDs beschäftigt sich nun damit, ob das, was in den vorherigen Phasen erarbeitet wurde, den Ansprüchen des potenziellen Nutzers gerecht wird. Dafür gibt es verschiedene Evaluationsmöglichkeiten. Der implementierte Prototyp wird den potentiellen Nutzer vorgeführt und es wird ermittelt, ob die Anforderungen erfüllt worden sind. Ist dies der Fall kann das System bzw. die Anwendung abgenommen werden. Ist dies nicht der Fall, gilt es die problematische Phase zu ermitteln, in diese zurück zu springen, und eine weitere Iteration durchzuführen.

3.3.3. Anwendung

In unserem Projekt wurde der HCD-Prozess insgesamt in zwei Iterationen, mit jeweils den nachfolgend beschriebenen Phasen, durchgeführt. Dabei wurde der HCD-Zyklus beim ersten Mal vollständig durchlaufen, jedoch in der zweiten Iteration

die erste Phase übersprungen, da am Nutzungskontext keine weiteren Änderungen vorgenommen werden mussten.

Erste Iteration

In der ersten Iteration wurden alle vier Phasen des HCD-Zyklus durchlaufen. Im Folgenden werden diese näher erläutert und auf den genauen Ablauf eingegangen:

1. Phase Der grundlegende Nutzungskontext wurde uns vorgegeben und wird daher nachfolgend nur beschrieben:
Die primären Nutzer des zu entwickelnden Produktes sind Wissenschaftler, die im Bereich der Mensch-Maschinen-Interaktion tätig sind. Das Produkt soll die Zielgruppe bei Rapid-Prototyping unterstützen und muss somit ein geeignetes Framework, das quelloffen und erweiterbar ist, besitzen, sowie schnell und kostengünstig reproduzierbar sein. Im Wesentlichen wird das Produkt unter Laborbedingungen Anwendung finden.
Die Motivation zum Einsatz des Produkts ist für den Anwender das einfache und schnelle Testen und Umsetzen von (selbst-)entwickelten Lösungen, die Augmented oder Virtual Reality mit Eye-Tracking kombinieren. Darüber hinaus kann davon ausgegangen werden, dass der Anwender über technische Kenntnisse in Programmiersprachen und Betriebssystemen, sowie über grundlegende technische Erfahrung verfügt.
Außerdem wurden User Stories geschrieben und auf deren Basis Anwendungsbeispiele erstellt.

2. Phase In der zweiten Phase lag der Fokus auf der Erhebung der Benutzeranforderungen.
Zu diesem Zweck wurden Experteninterviews mit unseren zwei Betreuern, als potentielle Nutzer des Produktes, durchgeführt. Die dabei gestellten Fragen waren offengehalten und das gesamte Interview war semi-strukturiert (vgl. 7). So hatten die Interviews eine generelle Struktur, und es konnte sichergestellt werden, dass alle wichtigen Aspekte durch Fragen abgedeckt wurden. Durch die offenen Fragen gab es aber auch die Möglichkeit, das Interview in andere vorher nicht absehbare Bereiche auszudehnen und neue Aspekte und Anforderungen aufzunehmen.
Da das Augenmerk des ersten Quartales vor allem auf der Erstellung eines Hardware-Prototypen lag, waren auch die Fragen primär auf die Hardware fokussiert.
Die Interviews wurden während der Durchführung sowohl durch einen Protokollanten dokumentiert, als auch mit Einverständnis der Interviewten auditiv aufgenommen.
Nach den Interviews wurde mit Hilfe der Aufzeichnungen ein Anforderungskatalog 4) aus funktionalen und nicht-funktionalen Anforderungen erstellt. Diese sollen die

Anforderungen der im ersten HCD-Teil ermittelten potentiellen Nutzer möglichst gut wiederspiegeln und die Erstellung des Produktes anleiten, um so zu einem möglichst benutzerfreundlichen Endergebnis zu gelangen. Die Erfüllung der Anforderungen wurde in der vierten Phase des HCD-Zyklus verifiziert und anschließend in der nächsten Iteration verfeinert und erweitert.

3. Phase In der dritten Phase Prototyping wurde ein Software- und Hardwareprototyp implementiert bzw. entwickelt.
Im Fokus stand die Entwicklung eines ersten „Quick and Dirty“ Hardware- Prototypen und die entwickelten Software-Auszüge dienten primär zum Testen der einzelnen Hardware-Komponenten.
Hardwareseitig wurde zunächst versucht möglichst schnell einen funktionsfähigen Prototyp zu erstellen, der alle nötigen, grundlegenden Funktionalitäten bietet. Dabei wurde nicht explizit festgelegt, ob es sich um einen „Wegwerfprototypen“ handelte, der nur exemplarisch Funktionalitäten des späteren Systems abbildete, aber nicht im weiteren Verlauf wiederverwendet oder ob dieser in einer Weiterentwicklung den Kern des finalen Systems darstellen würde. Für die Erstellung wurden aus den zuvor erhobenen Anforderungen Design- und Kaufentscheidungen an die Hardware abgeleitet. Dabei wurden ebenfalls die übergeordneten Anforderungen an das Projekt berücksichtigt. Hierzu zählen kostengünstige und einfach zu beschaffende Teile, da das fertige Produkt unter niedrigen Kosten einfach zu reproduzieren sein soll. Die Funktionalität wurde auf grundlegende Elemente beschränkt. Es sollte möglich sein, die Augenbewegung des Nutzers beim Tragen des HMDs mono- oder binokular zu verfolgen. Parallel sollte die Integration des Eye-Tracking in das HMD die Wahrnehmung der AR-/VR-Applikation nicht negativ beeinflussen (z.B. durch einen größeren Abstand zwischen Augen und Cardboard-Linsen, was zu einer unscharfen Wahrnehmung des Smartphone-Bildschirms führen würde). Die AR-Applikation wurde in diesem HCD-Zyklus über die integrierte Smartphone-Kamera realisiert, um auch hier dem Grundsatz zu folgen, möglichst schnell eine funktionierende Lösung zu haben.
Die Software-Entwicklung diente primär dem Testen der Hardware. So konnte bspw. durch die AR-/VR-Applikation verifiziert werden, dass diese trotz des veränderten Augenabstands zu den Linsen, unverzerrt wahrgenommen werden kann. Des Weiteren wurden Softwaremodule für die Kommunikation der Hardwarekomponenten entwickelt. Die Smartphone-Applikation wird in Echtzeit in das Eye-Tracking-Tool Pupil Capture übertragen, in dem ebenfalls eine Verbindung zu den Eye-Tracking-Komponenten des HMDs besteht, sodass auch das Bild der Eye-Tracking-Kamera sichtbar wird. Es wurden ausschließlich Open-Source-Frameworks verwendet, da der Fokus darauf liegt das fertige Projekt als quelloffen (unter Massachusetts Institute of Technology (MIT)-Lizenz) zu veröffentlichen.

Näheres zur Software als auch Hardware ist in den Kapiteln 5 und 6 nachzulesen.

4. Phase In dieser Phase wurde sich dazu entschieden eine Evaluation mittels Nutzerinterviews durchzuführen, da so direktes Feedback zum Prototypen und der verwendeten Hardware erhalten wird. Andere Verfahren hätten diese Informationen nicht geliefert, beziehungsweise nur in Kombination mit einem kurzen anschließenden Interview. Dabei nahmen an den Nutzertests fünf Probanden teil. Die erste Evaluation wird umfangreicher im Kapitel 7 geschildert.

Zweite Iteration

Die zweite Iteration wurde erneut mit der 2. Phase des HCD-Zyklus begonnen, um die Anforderungen auf Basis der ersten Nutzertests zu überarbeiten.

2. Phase Das genaue Vorgehen, der zur erneuten Anforderungserhebung durchgeführten Nutzertests, wird im Kapitel Erste Studie näher beschrieben und die daraus resultierenden Anforderungen wurden im Kapitel Anforderungen dargelegt.

3. Phase Nachdem die Anforderungen aktualisiert wurden, musste auf dessen Basis weiter implementiert werden. Dabei lag der Fokus zwar auf der Software, jedoch musste auch weiterhin die Hardware weiterentwickelt werden, um sich den Softwareanforderungen anzupassen. Besonders wurde dabei auf die Verbesserung der Benutzerfreundlichkeit wert gelegt; einerseits durch indirekt softwareseitige Maßnahmen, wie z.B. die Erstellung eines VR-/AR-Prefabs oder des Verfassens von Anleitungen, andererseits auch durch das Schreiben von Interfaces als Vorlagen für Interaktionstechniken oder das Implementieren eines Pupil Capture Streaming-Plug-Ins.

4. Phase Abschließend wurden weitere Nutzertests mit sechs HCD Forschern zum Testen des Frameworks in Kombination mit der Hardware durchgeführt. Ein wichtiger Aspekt war dabei z.B. die Position der Augenkamera, die weiter verbessert werden konnte. Näheres hierzu ist im Kapitel 7 zu finden.

3.4. Vorgehensmodell - Scrum

Als Projektmanagement-Tool wurde in unserem Projekt Scrum, ein agiles Vorgehensmodell zur Softwareentwicklung, ausgewählt und angewandt. Im Folgenden wird näher darauf eingegangen, wie es zu dieser Wahl bzw. Entscheidung kam

und was das Vorgehensmodell ausmacht und wie es letztendlich genau in unserem Projekt umgesetzt wurde.

3.4.1. Entscheidung

Aus der großen Menge an Konzepten für das Projektmanagement, baten sich zur Umsetzung dieses Projektes vor allem agile Konzepte an. Aufgrund der Kombination aus Hard- und Softwareentwicklung war abzusehen, dass im Verlauf der Projektgruppe stetig neue Anforderungen und damit Veränderungen am Konzept auftreten konnten. In einem agilen, iterativen Vorgehensmodell lassen sich neue Anforderungen in späteren Iteration neu einplanen und umsetzen (vgl. [Tre12]). Eine Wasserfall ähnliche Methode würde ein erneutes Einplanen nicht vorsehen. Von den iterativen Modellen fiel die Wahl auf Scrum, da es sich gut mit dem von uns angestrebten Vorgehen nach den Prinzipien des HCD kombinieren lässt. Zusätzlich versprachen wir uns von diesem Vorgehensmodell eine Unterstützung bei der Kommunikation durch regelmäßige Stand-Up Meetings. Da einige Projektgruppenteilnehmer bereits Erfahrung im Umgang mit Scrum hatten und die Methode gut dokumentiert ist, erhofften wir uns einen einfachen Einstieg in die Methode, ohne den Administrationsaufwand unnötig zu erhöhen.

3.4.2. Beschreibung

Wie bereits erwähnt, wurde für die Durchführung und Planung des Projekts unter anderem das Vorgehensmodell Scrum genutzt. Scrum wurde in der Softwaretechnik entwickelt und dient zur agilen Softwareentwicklung, kann aber auch unabhängig davon in anderen Bereichen genutzt werden. Grundsätzlich besteht Scrum aus wenigen Regeln, die fünf Aktivitäten, drei Artefakte und drei Akteure beschreiben. In den folgenden Abschnitten werden diese näher beschrieben.

Akteure

Die drei Akteure bzw. Rollen in einem Scrum Modell sind *Product Owner*, *Scrum Master* und das *Entwicklungsteam*. Alle drei Rollen zusammen werden als das Scrum Team bezeichnet. Der Product Owner ist für den wirtschaftlichen Erfolg des Produktes verantwortlich. Dieser gibt ebenfalls die Anforderungen an das zu erstellende Produkt an.
Der Scrum Master besitzt die Verantwortung, dass Scrum und dessen Regeln umgesetzt werden. Er beschäftigt sich unter anderem mit dem Einführen von Regeln oder der Überprüfung von Regeleinhaltungen. Der Scrum Master arbeitet mit dem Entwicklungsteam zusammen und bringt dem Team den Umgang mit Scrum bei.

Das Entwicklerteam ist für die Umsetzung von Backlogeinträgen verantwortlich. Dabei organisiert das Entwicklerteam sich selbst. (vgl. [ABP17], [Kri])

Artefakte

Die drei Artefakte, die in Scrum angewandt werden, sind *Product Backlog*, *Sprint Backlog* und *Product Increment*. Der Product Backlog ist ein langfristiger Plan mit einer Auflistung aller Anforderungen an ein Produkt. Dieser Plan unterzieht sich einer kontinuierlichen Weiterentwicklung. Der Product Owner ist hierbei für die Wartung und Pflege des Product Backlogs verantwortlich. Um diesen Backlog fachlich und anwendungsorientiert zu halten, werden für die Verdeutlichung am besten Eigenschaften eines Produktes in Form von User Stories formuliert.
Der Sprint Backlog ist ein Detailplan mit allen Aufgaben, die in diesem Sprint zu erledigen sind. Diese Aufgaben werden aus dem Product Backlog entnommen und in den jeweiligen Sprint Backlog übernommen. Für die gute Übersichtlichkeit werden die Teammitglieder dazu aufgefordert, ihre Bearbeitungsfortschritte einzutragen. Der Sprint Backlog ist für alle Mitglieder des Teams einsehbar. Das Product Increment ist die Summe aller Einträge im Product Backlog, also die Aufgaben, die erledigt wurden. (vgl. [ABP17], [Kri])

Aktivitäten

Die fünf unterschiedlichen Aktivitäten sind *Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospektive* und *Product Backlog Refinement*. Zur Sprint Planung stellen sich hierbei zunächst zwei Fragen: Was kann entwickelt werden und wie wird dies erledigt? Dafür wird die Planung meist in zwei Teile geteilt. Der erste Teil beschäftigt sich mit dem Was, der zweite Teil mit dem Wie. Das Daily Scrum ist der sogenannte Informationsaustausch zwischen den Teammitgliedern des Entwicklerteams. Somit erhalten alle Beteiligte einen Überblick über den aktuellen Stand der Aufgaben.
Die dritte Aktivität, die Sprint Review, findet am Ende eines Sprints statt. Es wird überprüft, ob die Ziele des Sprints erfüllt und der Projektfortschritt stimmt. Ebenso wie die Sprint Review steht am Ende eines Sprints auch die Sprint Retrospektive an. Diese hilft dem kompletten Team ihre bisherige Arbeitsweise gegenüberzustellen und zu reflektieren, wie effektiv und effizient ihre Arbeitsmoral ist. Die Sprint Retrospektive besteht aus fünf Phasen, die es gilt einzuteilen. Als fünfte Aktivität wird die Weiterentwicklung und Anpassung des Product Backlogs genannt, das so genannte Product Backlog Refinement. Dazu gehören Aufgaben wie: ordnen, löschen, hinzufügen, zusammenfassen und planen von Einträgen. (vgl. [ABP17], [Kri])

3.4.3. Anwendung

Die grundsätzliche Struktur des Projektverlaufes haben wir mittels Scrum organisiert. Nachdem im Laufe des ersten Quartals Scrum als Vorgehensmodell bestimmt wurde, konnte das Vorgehen in den einzelnen Sprints organisiert werden. Diese Sprints waren in der Regel zwei Wochen lang. Ausnahmen sind hier jedoch der erste Sprint, der fünf Wochen lang war und die Einarbeitung und die Seminarphase umfasste. Zudem wurden die Sprints direkt vor den jeweiligen Quartalsenden auf drei Wochen verlängert.
Jeder dieser Sprints begann mit einer Sprint Planung und endete mit einer Sprint Review und einer Sprint Retrospektive. Im Rahmen der Retrospektive wurde der jeweilige Sprint von den Gruppenmitgliedern reflektiert. Hierbei wurde Verhalten in den fünf Kategorien gesammelt. Die Kategorien sind: „More of“, „Less of“, „Keep doing“, „Start doing“ und „Stop doing“.
Zusätzlich dazu wurden erledigte Aufgaben auf Feedback gestellt und von einer anderen, nicht an der Bearbeitung beteiligten Person evaluiert, bevor diese endgültig auf erledigt gestellt wurde. Sollte eine Aufgabe nicht rechtzeitig vor Sprintende beendet werden können, wurde diese mit in den nächsten Sprint übernommen und dort vor den neuen Aufgaben bearbeitet.
Eine weitere Änderung ab dem dritten Quartal war eine konkrete Meilensteinplanung für jeden Sprint und eine teilweise dynamische Verteilung von Aufgaben. Bei dieser wurde Tickets für einen Aufgabenbereich erstellt, jedoch nicht zugewiesen. Stattdessen konnte sich jeder aus der Projektgruppe selbstständig eines der Tickets aussuchen, das dann bearbeitet wurde.

3.5. Projektphasen

Die Dauer des Projekts betrug ein Jahr, und bestand somit aus vier Quartalen. Für jedes Quartal waren Meilensteine gesetzt worden, die es zu erreichen galt. Zusätzlich stand beim Abschluss eines Quartals eine Präsentation an, die den aktuellen Stand des Projekts und Änderungen zum vorherigen Quartal aufzeigt.
Im Folgenden werden der Projektverlauf und die zu erreichenden Meilensteine im Form von Artefakten der einzelnen Quartale näher beschrieben und erläutert.

3.5.1. Quartal 1

Der Fokus des ersten Quartals lag auf der Erstellung eines ersten Hardware-Prototypen. Zusätzlich beinhaltete das Quartal die Einarbeitungs- sowie Seminarphase der Projektgruppe. Zudem fand im ersten Quartal eine Projektorganisation statt, im Rahmen derer das Vorgehensmodell der Projektgruppe festgelegt und die Infrastruktur, wie Git und Redmine, erstellt wurde.

Verlauf/Artefakte

Angefangen hat das Quartal mit der Einarbeitungs- und Seminarphase. In dieser wurden Seminarthemen in der Projektgruppe verteilt und jeder erstellte zu einem dieser Themen eine Ausarbeitung. Diese diente als inhaltliche Grundlage der Projektgruppe. Zusätzlich wurden in dieser Phase ein Großteil der Rollen innerhalb des Projektes verteilt, auf diese näher in Kapitel 3.1 eingegangen werden.
Außerdem wurde schon zu Beginn des Quartals als erste Idee festgesetzt, dass unser Prototyp auf Basis des Google Cardboards erstellt werden sollte und die verwendete Software (vgl. 6.2.2) selektiert. Zum Ende der Anfangsphase haben wir eine Kombination aus HCD (vgl. 3.3.2) und Scrum (vgl. 3.4) als Vorgehensmodell festgelegt.
Nach dieser ersten Phase der Einarbeitung in die Thematiken Mixed Reality und Eye-Tracking begann im ersten Sprint die strukturierte Arbeit an der Erstellung des Prototyps. Im Rahmen des HCD-Prozesses begannen wir damit Anforderungen durch zwei Experteninterviews zu erheben. Das Quartal endete mit der Erstellung des ersten Prototyps (vgl. 5.3.1) und den dazugehörigen Hardware-Tests und der Dokumentation für die Einrichtung der Software- und Hardware-Komponenten.

Aufgaben und Meilensteine

Die Aufgaben innerhalb des ersten Quartals lassen sich in mehrere Kategorien aufteilen:

- Organisatorisches:
  - Festlegen eines Vorgehensmodells
  - Erstellen eines Projektplans
  - Aufstellen eines Basis-Anwendungsfalls
  - Rollenverteilung und -beschreibung
  - Erstellen von Dokument- und Präsentationsvorlagen
  - Einrichten von Git, Mailverteiler und Redmine
- Konzeptuelles:
  - Einblicke sammeln in unterschiedliche Komponenten, wie Pupil Capture, Unity und ARToolkit
  - Erstellen einer grundlegenden Architektur
  - Erstellen eines Konzepts für den Hardware-Prototypen
  - Durchführen der Anforderungserhebung

  - Erstellen eines ersten Backends für Pupil Capture
  - Erstellen erster AR/VR Szenen in Unity

Entscheidungen/Besonderheiten

Innerhalb des Quartals wurde neben vielen organisatorischen Entscheidungen, wie Vorgehensmodell und Rollenverteilung, auch die Komponenten des Hardware-Prototypen bestimmt.

3.5.2. Quartal 2

Nachdem der Schwerpunkt im ersten Quartal auf der Erstellung des Hardware-Prototypen lag, legte das zweite Quartal den Fokus primär auf die Softwareentwicklung. Parallel wurde am erstellten Prototypen aus Quartal 1 weitergearbeitet und Verbesserungen vorgenommen. Nachdem zunächst die ersten drei Wochen des Quartals zunächst auf die Evaluationsphase des HCD-Zyklus konzentriert waren, ist ab Sprint Nr. 5 die tatsächliche Softwareentwicklung in den Vordergrund gerückt.

Verlauf/Artefakte

Die Zielsetzung des zweiten Quartals war ein Rapid-Prototyping förderndes Open-Source-Framework unter MIT-Lizenz zu implementieren. Es sollte zudem die Entscheidungen gefällt werden, wie das video-see-through realisiert wird und wie das Eye-Tracking in Software umgesetzt wird. Für das video-see-through wurde sich letztendlich für die Smartphone-Kamera als Medium entschieden und das Eye-Tracking wurde, wie schon in Q1 entschieden, mit Pupil Capture umgesetzt, womit die errechneten Blickdaten letztendlich an die mobile Applikation übermittelt werden. Neben dem Software-Framework war ein weiteres Artefakt des Quartals die aufgesetzte Website, auf der das Projekt und die Arbeitsergebnisse auch nach Abschluss der Projektgruppe, dargestellt werden.

Aufgaben und Meilensteine

Die Aufgaben des zweiten Quartals lassen sich in folgende Kategorien einteilen:

- Allgemein
  - Recherche zu Evaluationsverfahren
  - Durchführen von Nutzerinterviews
  - Aufstellen eines Entwurfskonzeptes
  - Recherche zu alternativen Hardware-Maßnahmen

- Software
  - Datenexport und -import von Pupil zu Unity als Grundlage für Interaktionstechniken
  - Kalibrierung von Pupil Capture
  - ARToolkit aufgrund der Inkompatibilität zu dem erstellten Software-Framework als AR-Plattform verlassen und eine eigene AR-Lösung implementieren (vgl. 6.2.3)
  - Manual Marker Calibration als Kalibrierungsmethodik für Pupil Capture
  - Mapping von der Stream-Kamera auf Split Screen
  - Unit-Tests
  - Integrationstests
  - Framework konzipieren und implementieren
  - Pupil-Capture-Backend verbessern
  - Webseite aufsetzen
- Hardware
  - Optimierung des Prototyps (AR, VR und Eye-Tracking)
  - HCD Zusammenfassung der vier Phasen
  - Alternatives Hardwarekonzept aufstellen
  - Version Zwei des Prototyps anfertigen
  - Auslagerung von Pupil Capture auf den Pi3
  - AR Kameraentscheidungen (integrierte Smartphone Kamera oder externe Universal Serial Bus (USB)-Kameras)

Entscheidungen/Besonderheiten

Eine Besonderheit dieses Quartals war die Powerwoche vom 06.03 bis 10.03, in der es zum einen um die Erstellung des Software-Frameworks und zum anderen um die Weiterentwicklung des Hardwareprototypen ging. Hierfür wurde zu Beginn der Woche zunächst das Framework auf konzeptioneller Ebene entwickelt und daraufhin implementiert. Eine weitere Zielsetzung der Powerwoche war es, durch die enge Zusammenarbeit Teamgeist und Motivation zu steigern. Außerdem bekamen alle Mitglieder der Projektgruppe in alle Teilbereiche Einblicke, welche die Transparenz für die verschiedenen Aufgabenbereiche erhöhen konnte.

3.5.3. Quartal 3

Im dritten Quartal lag, anders als beim Projektkickoff festgelegt, nicht nur der Fokus auf der einfachen Reproduktion unseres Gesamtsystems und der damit verbundenen Dokumentation wie z.B. der Erstellung von Anleitungen, sondern auch auf der Erstellung erster Proof-of-Concept-Szenen. Der Grund dafür war, dass aufgrund des zeitlichen Verlaufs uns angeraten wurde, die Ziele und Artefakte des 3. und des 4. Quartals in Teilen zu tauschen, um einen effektiveren Projektverlauf zu gewährleisten.

Verlauf/Artefakte

Das dritte Quartal hatte die (angepasste) Zielsetzung, das Eye-Tracking-Frameworks zu erweitern und für die unterschiedlichen Interaktionstechniken Proof-of-Concept-Szenen zu erstellen. Weiterhin sollten Anleitungen verfasst werden, die es schnell und einfach ermöglichen, eigene Interaktionstechniken mit unserem Framework zu implementieren.
Im Verlauf des Quartals wurde daher der Schwerpunkt erst auf die Softwareentwicklung gesetzt, um anschließend auf Basis des weiterentwickelten Frameworks Anleitungen, sowie auch einfache Proof-of-Concept-Szenen erstellen zu können. Abschließend folgte die Präsentation unseres Fortschritts im Oberseminar.

Aufgaben und Meilensteine

Die Aufgaben des dritten Quartals lassen sich in die Kategorien Allgemeines, Software und Hardware einteilen.

- Allgemein:
  - Anleitung SDK/JDK
  - Anleitung Einrichtung Unity
  - Anleitung AR-Szene
  - Anleitung VR-Szene
  - Webseite wurde auf Englisch übersetzt, erweitert und alle fertiggestellten Anleitungen hochgeladen
  - Konzept für Nutzertests zur weiteren Anforderungserhebung erarbeitet und Tests durchführt
  - Q3 Präsentation

- Software:
  - Dynamische Kalibrierung in die Szene integriert und Input über die Fernbedienung realisiert
  - Nutzung und Unterstützung von Vuforia bzw. Marker-Tracking in Zusammenspiel mit unserem Framework
  - Bildübertragung von Unity nach Pupil Capture in ein Plug-In umgeschrieben
  - Interfaces als Vorlagen für Interaktionstechniken
  - Services zum Erkennen von Fixationen, Gesten und Verfolgung
  - Die Bereitstellung des Software-Frameworks als Unity-Package
  - Sakkadenvisualisierung
  - Testen der Services durch Unit-Tests
  - Proof-of-Concept-Szene für die Gesten, Fixation sowie die Verfolgung von Objekten
- Hardware:
  - Geätzte LED-Platinen für die Eye-Tracking-Kameras mit Komponenten bestückt
  - Cardboard-Bauanleitung überarbeitet

Entscheidungen/Besonderheiten

Eine Besonderheit dieses Quartals war das QM-Treffen. Es wurde von unserem Qualitätsmanagement organisiert, um z.B. eine neue Git Struktur mit dem gesamten Projektteam zu erarbeiten und umzusetzen. Es wurde weiterhin dazu genutzt, die Code-Dokumentation auf Fehlerfreiheit zu überprüfen, ggf. anzupassen und fehlende Dokumentation hinzuzufügen. Außerdem wurde die Webseite auf eine eigene Domain übertragen.

3.5.4. Quartal 4

Die Zielsetzung des vierten Quartals war zum einen das Testen des Frameworks in Kombination mit der Hardware durch Nutzertests, zum anderen soll auf Basis dessen Feedback eingearbeitet werden. Dafür wurden die Nutzertests geplant und durchgeführt. Des Weiteren wurde Fokus auf die Planung des Abschlussberichtes gesetzt. Zudem gab es intern diverse weitere Tests des Frameworks und des Hardware-Prototypen. Insbesondere wurde dort noch einmal die Kamera-Position überdacht und verändert. Außerdem wurde die Webseite aktualisiert und mit neuen Inhalten gefüllt.

Verlauf/Artefakte

Durch den Tausch der Quartale drei und vier wurden einige Artefakte geändert. So standen in diesem Quartal diese Artefakte auf unserer Liste: Do-It-Yourself (DIY)-Charakter, niedrige Kosten, einfache Reproduktion. Diese Artefakte wurden durch die durchgeführten Nutzertests erfüllt. Durch die zuvor erstellten Anleitungen konnte eine einfache Reproduktion erforscht werden. Der DIY- Charakter wurde durch das Zusammenbauen des Cardboards wiedergespiegelt und die niedrigen Kosten durch die verhältnismäßig günstigen Komponenten der Hardware. Zudem wurde die Frage: „Welche Informationen braucht jemand, um das System nachzubauen und für Forschungsprojekte einzusetzen?“ gestellt. Diese Informationen kann ein potenzieller Nutzer aus unserer Webseite erhalten. Dort werden alle Rahmenbedingungen aufgeführt und die Anleitungen bereitgestellt. Zudem kann auf das Software-Framework über unser Git-Repository zugegriffen werden. Die Einsatzgebiete für das Framework sind breit gefächert. Bei den potenziellen Nutzern handelt es sich um HCI-Wissenschaftler, die dieses Framework für ihre Studien einsetzen können.

Aufgaben und Meilensteine

Die Aufgaben des vierten Quartals lassen sich in folgende Kategorien einteilen:

- Allgemein:
  - Anpassung der Anleitungen
  - Reviews der vergangenen Sprints erstellen
  - Webseiteninhalte aktualisieren und mit neuen Informationen befüllen
  - Präsentationvorbereitungen für Herr Gellersen
  - Alle Skripte dokumentieren
  - Endpräsentationskonzept
  - Nutzertest durchführen
- Software:
  - Auslagerung der Streaming-Kamera in ein Skript
  - Toleranzbereich des PursuitDetectionServices an den Kamerawinkel anpassen
  - Koordinaten-Berechnung von Bildschirm auf Kameraauflösung umstellen
  - Drei Szenen für die Nutzertests implementieren und durch Anleitungen unterstützen, sodass diese mit Probanden getestet werden können

  - Einheitlicher Zugriff auf Objekte gewährleisten
  - Toolbox (Service Provider) als zentrale Modelklasse ausbauen
  - Streaming-Einstellungen überprüfen
  - Kalibrierungsmarker anpassen, sodass diese korrekt mit dem linken, rechten oder beiden Augen erfasst werden können
  - Erweiterung um Unsubscribe-Methoden in den drei Services
  - Größe des Pursuit-Toleranzbereiches in Abhängigkeit des Kamera-Abstands berechnen
  - Unit-Test für die drei Services schreiben
  - Fehlerbehebung des Software-Frameworks
  - Binokulares Eye-Tracking testen
  - Inspector-Einstellungen vereinheitlichen und mit Beschreibungen versehen
  - Gesture Creator soll über den Inspector-Bereich zu öffnen sein
  - DrawSaccades umstellen, sodass die nur fertigen Gesten bemalt werden oder den Fortschritt einer Geste
  - Aufnahmen und Abspeichern der Eye-Tracking-Daten und der Kopfbewegung, sowie die Möglichkeit zum Abspielen soll implementiert werden
  - Testen und Überarbeiten der Services
- Hardware:
  - Vorlagen des Cardboards anpassen
  - Neuere Kopfhalterung an den Hardware-Prototypen bauen
  - Kamera-Position testen
  - Informieren über ein „Bundle“

Entscheidungen/Besonderheiten

Es wurde eine letzte Meilensteinplanung für das Quartal durchgeführt. Zudem wurden letzte Änderungen an der Software durchgeführt und danach ein „Code Freeze“ gesetzt. Es gab einen Vortrag von Herr Gellersen über Eye-Tracking. Des Weiteren haben wir ihm unsere Lösung präsentiert. Das Framework wurde für die Bereitstellung vorbereitet und auf unser Git geladen. Zudem wurde die Webseite finalisiert.

4. Anforderungen

Im folgenden Kapitel werden die Anforderungen an unser Produkt näher erläutert, die durch das Interviewen von potenziellen Nutzern erhoben wurden und auf deren Basis weitergearbeitet wurde. Die funktionalen und nicht-funktionalen Anforderungen werden im Abschnitt Benutzeranforderungen näher beschrieben. Im zweiten Teil dieses Kapitels wird vertieft auf das Produkt bezüglich grundlegender Vision, Funktion und Einbettung eingegangen. Zudem werden die Anforderungen potenzieller Nutzer betrachtet, anhand eines allgemeinen Anwendungsfalles verdeutlicht und schlussendlich allgemeine Rahmenbedingungen spezifiziert.

4.1. Nutzerinterviews

Nach einer Spezifikation des Nutzerkontextes und Charakterisierung beispielhafter Anwender und Anwendungsfälle des zu erstellenden Produktes, empfiehlt der HCD-Prozess 3.3.2 eine Erhebung der Benutzeranforderungen. Hierbei sollen Erwartungen und Wünsche einer potentiell nutzenden Person an das Produkt ermittelt werden, um ein besseres Verständnis für diese zu erlangen und ein Produkt möglichst nah an den Vorstellungen dieser Person zu entwickeln. Zu diesem Zweck wurden Interviews mit zwei Wissenschaftlern aus dem Bereich der Mensch-Maschine-Interaktion durchgeführt.
Für die Interviews wurde ein Katalog aus Fragen vorbereitet, die dem Interviewer bei der Strukturierung der Befragung helfen sollte. Mithilfe dieses Fragenkataloges wurde das Interview dann semi-strukturiert durchgeführt, sodass zwar die generelle Struktur durch die eher offenen Fragen vorgegeben war, das Interview bei Bedarf aber auch in unvorhergesehene Richtungen vertieft werden konnte. Die Befragten waren angehalten frei und offen über ihre Erwartungen und Wünsche zu sprechen und wurden nach ihrer persönlichen Meinung gefragt. Die Interviews wurden dabei protokolliert und bei vorher erfragtem Einverständnis durch eine Tonaufnahme aufgezeichnet. Aus den Protokollen und Aufzeichnungen wurden dann funktionale und nicht-funktionale Anforderungen erhoben und in Form eines Anforderungskataloges spezifiziert. Das folgende Kapitel 4.2 soll einen Überblick über die erhobenen Anforderungen geben, die eine zentrale Rolle bei dem Entwicklungsprozess der Hard- und Software gespielt haben.

4.2. Benutzeranforderungen

Durch die zuvor durchgeführten Nutzerinterviews konnten die Bedürfnisse und Wünsche der potenziellen Nutzer erfasst werden. Nun galt es die Benutzeranforderungen durch die Ergebnisse der Interviews zu ermitteln und zu definieren. Die erarbeiteten funktionalen und nicht-funktionalen Anforderungen können dem nachfolgenden Kapitel entnommen werden.

4.2.1. Funktionale Anforderungen

Dieser Abschnitt enthält zunächst die funktionalen Anforderungen an das Produkt, die auf Basis der Nutzerinterviews identifiziert werden konnten.

FA01 Das Head-Mounted Display soll über Software virtuelle Realität simulieren können.

FA02 Das Head-Mounted Display soll erweiterte Realität realisieren können.

FA03 Die erweiterte Realität soll über video-see-through realisiert werden.

FA04 Das Head-Mounted Display soll über eine Möglichkeit verfügen, die Bewegung der Augen erfassen zu können.

FA05 Die Eye-Tracking-Komponente soll die erfassten Daten an die anderen Komponenten übertragen können.

FA06 Das Software-Framework soll die Daten des Eye-Trackings den anderen Software-Komponenten zur Verfügung stellen.

FA07 Das System soll die Möglichkeit bieten, Forschungsdaten aus der Software zu extrahieren und zu versenden.

FA08 Das System kann die Möglichkeit eines Video-Streams auf einem externen Rechner zu Beobachtungszwecken bieten.

FA09 Das Software-Framework soll dem Nutzer die Möglichkeit bieten eigene Interaktionstechniken realisieren zu können.

FA10 Das System bietet Möglichkeiten für mono- sowie binokulares Eye-Tracking an.

Schnittstellenanforderungen

Bei Schnittstellenanforderungen handelt es sich um Anforderungen bzgl. der Kommunikation zwischen mehreren Komponenten. Durch die Umsetzung dieser Anforderungen soll die Kommunikation zwischen den einzelnen Komponenten, auch bei möglichem Austausch, gewährleistet werden.

SA01 Die Probandensicht des Smartphones muss per User-Datagram-Protocol (UDP)/Internet Protocol (IP) erreichbar sein und einen Video-Stream liefern.

SA02 Die AR- bzw. VR-Applikationen sollen über Transmission-Control-Protocol (TCP)/IP mit der Eye-Tracking-Software beidseitig Daten austauschen können.

SA03 Die Eye-Tracking-Software soll eine Schnittstelle zur Datenaufnahme in einem spezifizierten Dateiformat aufweisen, die sowohl verarbeitete als auch Rohdaten aufnehmen kann.

SA04 Die Unity-Applikation muss eine Schnittstelle zur Aufnahme von Eye-Tracking-Daten zur Verfügung stellen und eine Funktion zum Einlesen dieser aufweisen, wodurch eine Nachvollziehbarkeit der Szenenabläufe gewährleistet ist.

SA05 Die Eye-Tracking-Software muss per USB Zugriff auf die Eye-Tracking-Kameras haben, um von diesen Kamerabilder einlesen zu können.

Entwurfsanforderungen

„Die Entwurfsanforderungen bestimmen maßgeblich, wie die Architektur zu entwerfen ist.“ ([FK09]), S.17). Sie beschreiben „was, nicht wie“ ([FK09]), S.17) die Software oder Hardware auszusehen hat und bilden somit das Fundament für die spätere Software-Architektur.

EA01 Das Head-Mounted Display soll auf der Basis des Google Cardboards und zusätzlich mit einem Referenz-Smartphone mit einem bzw. zwei Kameramodulen zum Eye-Tracking funktionieren.

EA02 Die Kommunikation zwischen den Komponenten des Head-Mounted Displays muss eine geringe Latenz aufweisen, sodass die Benutzbarkeit des Head-Mounted Displays nicht eingeschränkt ist.

EA03 Die Übertragung des Bildes der Unity-Anwendung, von dem Smartphone zum Rechner, muss über ein Netzwerk mit Wireless Local Area Network (WLAN) funktionieren.

EA04 Es muss eine Übertragung der Eye-Tracking Daten von Pupil Capture zur Unity-Anwendung auf dem Smartphone stattfinden.

EA05 Das Head-Mounted Display darf nicht zu schwer und zu groß sein um den Tragekomfort nicht einzuschränken.

4.2.2. Nicht-funktionale Anforderungen

Der folgende Abschnitt listet alle nicht-funktionalen Anforderungen an das Produkt auf. Diese sind in die Aspekte Leistung, Schnittstellen, Entwurf und Qualität gruppiert.

Leistungsanforderungen

Bei Leistungsanforderungen handelt es sich um Anforderungen, die die Leistungsfähigkeit des Produktes beschreiben. Durch die Definition dieser Anforderungen soll sichergestellt werden, dass das Produkt performant genug für die Verwendung ist.

LA01 Da es sich um ein echtzeitkritisches System handelt, muss die Latenz bei Antworten des Systems bei 95% aller Aktionen zwischen 1-10ms liegen.

LA02 Das Eye-Tracking muss in 95% der Anwendungsfälle präzise genug sein, um den Blickpunkt des Nutzers identifizieren zu können.

LA03 Da das Einsatzgebiet des Produktes Rapid-Prototyping in der Forschung ist, muss es den Nutzern nach maximal einem halben Arbeitstag Implementierungsaufwand zur Verfügung stehen.

LA04 Die Applikation soll performant auf einem Referenz-Smartphone laufen.

Qualitätsanforderungen

Die Software-Qualität wird nach DIN-ISO-NORM 9126, „als die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produktes, die sich auf dessen Eignung beziehen [..und] festgelegte Erfordernisse erfüllen“, definiert.
Dies drückt aus, dass es besonders für Software aber auch Hardware, „nicht das eine [Qualitäts]Kriterium“ gibt. Eine Vielzahl an unterschiedlichen Aspekten muss berücksichtigt werden, um Qualität zu gewährleisten. (vgl. [Hof13], S.6)
Der nächste Abschnitt legt für einige dieser Aspekte Anforderungen dar.

Zuverlässigkeit Zuverlässigkeit ist der Grad, in dem etwas (z.B. ein System oder ein Produkt) erwartungsgemäß bzw. korrekt funktioniert. D.h., dass sowohl Fehlerzustände selten bis nie erreicht bzw., falls möglich, behandelt oder verhindert werden.

QA01 Im Fehlerfall sollten, falls möglich, Fehlermeldungen ausgegeben werden, die dem Benutzer das Problem hinreichend schildern.

QA02 Im Regelfall sollten im laufenden Betrieb nur selten Fehler auftreten.

Sicherheit Ein weiterer Aspekt, der zur Qualität des Produktes beiträgt, ist eine gefahrenfreie Bedienbarkeit. Aus diesem Grund sollte folgende Anforderungen nicht vernachlässigt werden.

QA03 Das Produkt darf kein Sicherheitsrisiko darstellen.

QA04 Die Verbreitung von personenbezogener Daten nach außen soll vermieden werden.

Wartbarkeit Das Produkt sollte wartbar sein d.h. auch für andere Wissenschaftler erweiterbar und veränderbar sein.

QA05 Eine genaue Dokumentation des Entwicklungsprozesses muss während der Entwicklung zeitnah erstellt werden und zum Projektabschluss vollständig vorhanden sein.

QA06 Es müssen Hardware- und Softwaretests durchgeführt werden.

QA07 Der entwickelte Code muss ausreichend kommentiert sein.

Erlernbarkeit Für den Nutzer des Produkts bzw. Systems muss ein möglichst schneller Einstieg gewährleistet werden.

QA08 Das Produkt muss Rapid-Prototyping fördernd sein.

QA09 Das Produkt soll ein Proof-of-Concept durchlaufen.

QA10 Das Produkt soll durch Bauanleitung unterstützt werden.

QA11 Das Framework bietet Beispielanwendungen und bereits implementierte Beispieltechniken an.

QA12 Eine Hardware-CAD-Vorlage muss erstellt werden.

Benutzbarkeit Die Benutzbarkeit des Produktes sollte für den Nutzer keine Einstiegshürde bieten. Daher sind folgende Anforderungen sinnvoll.

QA13 Die Hardware sollte einen möglichst hohen Tragekomfort aufweisen.

QA14 Es soll eine freihändige Nutzung durch den Probanden gewährleistet sein.

QA15 Es sollte außerdem intuitiv anhand der Anleitungen benutzbar sein und einen möglichst geringen Einarbeitungsaufwand erforderlich sein.

4.3. Produkt-Einbettung

Das System besteht aus zwei Hauptkomponenten: dem Cardboard und einem Rechner. Am Cardboard ist ein USB-Hub, eine bzw. zwei Eye-Tracking USB-Kameras und eine bzw. zwei Infrarot (IR)-LED-Platinen angebracht und es wird ein Smartphone in die Smartphone-Halterung des Cardboards eingelegt. Eine IR-LED-Platine ist mit der Eye-Tracking-Kamera verbunden und wird dadurch mit Strom versorgt. Die IR-LEDs sollen das Auge des Nutzers beleuchten während die Eye-Tracking-Kamera das Auge aufzeichnet. Die Kamerabilder der Eye-Tracking-Kamera werden über ein USB-Hub und ein USB-Verlängerungskabel an den Rechner verschickt und in Pupil Capture, das Eye-Tracking Programm des Systems, geladen. Das Eye-Tracking Programm wertet die Bilder aus und schickt die ausgewerteten Daten an den Smartphone am Cardboard. Auf dem Smartphone läuft die zum System zugehörige Applikation (App), die diese Daten empfängt und in eine VR/AR Szene umsetzt. Dabei wird die Kamera des Smartphones für die AR-Szene genutzt. Die App stellt verschiedene Anwendungen zu VR und AR bereit, in der die Eye-Tracking-Daten z.B. zur Steuerung oder Messung dienen. Zusätzlich bietet die App eine Funktion, die die verschiedenen Ausgaben, sprich VR, AR und Eye-Tracking, an den Rechner zur Forschungszwecken sendet (siehe Abbildung 4.3.1).

4.4. Produkt-Vision

In diesem Abschnitt soll, unter Berücksichtigung der erhobenen Anforderungen, beschrieben werden, welche Vision also Zielvorstellung für das finale Produkt verfolgt bzw. angestrebt wird.
Die Augenbewegung wird wahlweise über eine oder zwei USB-Kameras aufgenommen, da sowohl mono- als auch binokulares Eye-Tracking unterstützt wird. Die USB-Kamera(s) werden dann über USB an Pupil Capture weitergeleitet. Hier findet das reguläre Eye-Tracking statt, in dem aus der Augenbewegung mithilfe von Pupil Capture als *.JSON* kodierte Koordinaten/Vektoren erzeugt werden. Diese

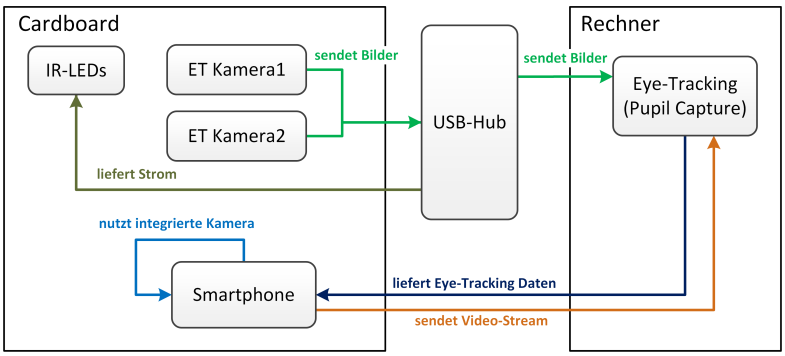

Abbildung 4.3.1.: Darstellung der verschiedenen Komponenten

werden dann über TCP mit ZeroMQ/ZMQ an eine VR/AR-Unity3D-Applikation übertragen und dort als Richtungsvektoren im 3D-Raum oder Positionspunkte auf dem Bildschirm verarbeitet. Der Stream der VR/AR-Applikation wird über UDP zurück an Pupil Capture übertragen. Ebenso können hier relevante Forschungsdaten zurückgegeben werden. Im Anwendungsfall einer AR-Applikation wird die reale Umgebung über die Smartphone-Kamera abgefilmt. Das Bild steht der Applikation also direkt zur Verfügung und musst nicht erst, wie z.B. im Falle von externe AR-Kameras, an das Smartphone übertragen werden, wodurch die Performance nicht geschmälert wird.

4.5. Produkt-Funktionen

Das, im Rahmen der Projektgruppe entwickelte, HMD und das dazugehörige Software-Framework sollen eine Plattform zur Verfügung stellen, mit der einfach und schnell Prototypen für AR/VR Anwendungen in Kombination mit Eye-Tracking entwickelt werden können.
Um dies zu ermöglichen bietet das HMD einige Funktionen:

- Das HMD unterstützt Augmented Reality/Virtual Reality Anwendungen.
- Das HMD beinhaltet eine bzw. zwei Augenkamara(s) für das Eye-Tracking.
- Für bessere Verwendung in Augmented Reality Anwendungen nutzt das HMD, die in das Smartphone integrierte Kamera.

Das dazugehörige Software-Framework bietet Funktionen, durch die eine schnelle Entwicklung von Prototypen möglich ist:

- Das Framework übernimmt die Kommunikation zwischen den verschiedenen Komponenten (HMD, Eye-Tracking Kamera(s) und Eye-Tracking Software auf einem externen Rechner).

- Das Framework ist durch die Open-Source-Lizenz anpassbar und erweiterbar.

4.6. Allgemeiner Anwendungsfall

Um die Anforderungen besser zu spezifizieren, wird ein Anwendungsfall benötigt. Mit Hilfe von Anwendungsfällen können Szenarien und Produkte besser dargestellt werden. In diesem Abschnitt wird ein allgemeiner Anwendungsfall vorgestellt, der den typischen Einsatz des Frameworks visualisiert. Der Anwendungsfall ist in der Abbildung 4.6.1 dargestellt.

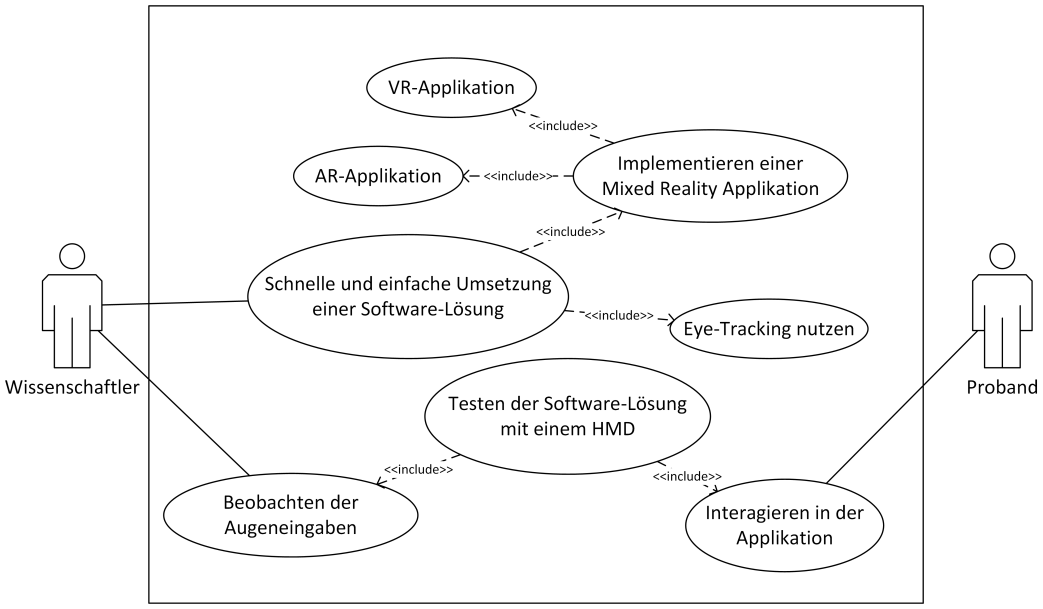

Abbildung 4.6.1.: Darstellung des allgemeinen Anwendungsfalls

Der Akteur ist ein Wissenschaftler, der eine Idee für eine Interaktionstechnik mit Eye-Tracking in einer Mixed Reality Applikation besitzt und diese schnell und kostengünstig umsetzen bzw. testen möchte. Dafür baut er mit Hilfe der Bauanleitung ein HMD und implementiert mit Hilfe des Frameworks eine Mixed Reality Applikation mit seiner Interaktionstechnik. Anschließend setzt sich ein Proband das HMD auf und interagiert mit der Mixed Reality Applikation während der Wissenschaftler das Geschehen in einem Forschungsrechner beobachten kann.

4.7. Allgemeine Anforderungen an den Benutzer

Dieser Abschnitt soll eine allgemeine Beschreibung zu den Anforderungen eines möglichen Benutzers liefern.

Dabei soll nicht auf die konkreten Anforderungen an das System/Produkt eingegangen werden, sondern nur ein Abriss des Benutzerkontextes bzw. der Benutzergruppe dargestellt werden.

- Alter: unabhängig
- Geschlecht: unabhängig
- Bildungshintergrund: Akademiker
- Sprache: deutsch/englisch
- Kultureller Hintergrund: unabhängig
- Technische Expertise: C#
- Physische Fähigkeiten oder Einschränkungen: ein uneingeschränktes (oder korrigiertes) Sehvermögen und eine nicht beeinträchtigte physische Beweglichkeit wird angenommen
- Kompetenzen/Erfahrung: auch für Anfänger ohne Vorkenntnisse von AR/ VR/ Eye-Tracking geeignet. Eine grundlegende technische Erfahrung wird vorausgesetzt
- Motivation: Eye-Tracking Ansätze ausprobieren, umsetzen und erforschen mit möglichst wenig Ressourcen
- Physische Umgebung: Laborbedingungen

4.8. Allgemeine Rahmenbedingungen

Unter den allgemeinen Rahmenbedingungen versteht man die Möglichkeiten der Realisierung, die das Produkt einschränken und welche Komponenten nur geringfügig verändert werden können. Dafür gibt es vier Kategorien, die durch Rahmenbedingungen eingeschränkt werden können. Zudem ist die Rahmenbedingung der Kosten auf einen Rapid-Prototyp zurückzuführen. Des Weiteren haben wurde der Extrapunkt Hardware beigefügt.

- Voraussetzungen an die verwendete Hardware:
  - Rechner: Windows/Mac, Pupil Capture, Unity Engine, Android Studio (SDK 7.0 API 24)

  - Smartphone: Android 7.0, Unity-Applikation, leistungsstark

  - Head-Mounted Display: Google Cardboard(v2), USB-Kamera(s), Platine(n) mit Infrarot LEDs und Widerstand, Kabel und Befestigungsmaterial

- Technologisch:
  Technologisch wird die IT-Infrastruktur wiedergegeben, die für das Produkt gilt. In erster Linie ist die Umgebung dieses Produktes das Labor. Aber auch hier könnte es für in-field Studien angepasst werden. Für die technische Nutzung der Lösung dient eine „Pupil-Unity Kombination“. Als technische Voraussetzung ist es wichtig, dass die benötigte Hardware international verfügbar ist.

- Organisatorisch:
  Organisatorisch gesehen gehören alle Nutzer und Entwickler des Produktes dazu. Bei der Zielgruppe handelt es sich um Forscher, die dieses Produkt für die Entwicklung von Szenarien nutzen, um diese mit Probanden zu testen. Dennoch ist das Framework erweiterbar und auf andere Nutzergruppen reproduzierbar. Für auftretende Fragen steht ein Benutzerhandbuch zur Verfügung.

- Rechtlich:
  Rechtlich gesehen sollte es für das Produkt keine Einschränkungen geben, da mit lizenzfreien und Open-Source-Lösungen gearbeitet wurde.

- Ethisch:
  Ebenso wie bei den rechtlichen Einschränkung wird es keine ethischen Einschränkungen geben.

4.9. Annahmen und Abhängigkeiten

In diesem Abschnitt werden zusätzliche Annahmen sowie Abhängigkeiten für den Einsatz und Aufbau des Produkts aufgelistet.
Es wird angenommen, dass

- die späteren Anwender hauptsächlich Wissenschaftler sind.

- die späteren Anwender die Platine für die Beleuchtung des Auges löten können.
- die späteren Benutzer Programmierkenntnisse in C# besitzt.
- das Produkt hauptsächlich in geschlossenen Räumen genutzt wird.
- das Produkt hauptsächlich in der Forschung genutzt wird.
- das Produkt in Rahmen von Rapid-Prototyping in ein bis zwei Tagen aufgebaut werden kann(inkl. Bestellungen)
- den späteren Anwendern einen Lasercutter zur Verfügung steht.
- den späteren Anwendern aktuelle Smartphones und Rechner zur Verfügung stehen.
- das Framework von den späteren Anwendern erweitert wird.

Daraus ergibt sich, dass

- das Produkt eine Funktion für die Analyse von durchgeführten Studien bzw. Experimenten auf einen Rechner anbieten muss.
- die Materialien des Prototyps leicht zu beschaffen sein müssen.
- der Prototyp leicht aufzusetzen ist.
- das Framework des Produkts leicht erweiterbar sein muss.

5. Hardware

In diesem Kapitel wird das Konzept und die Entwicklung des HMD näher beschrieben.
Zuerst werden die Zielsetzung und die Anforderungen an die Hardware und an den Prototypen erläutert. Anschließend wird das grundlegende Konzept für das HMD vorgestellt. Daraufhin folgt der Hauptteil des Kapitels: Die Umsetzung des Konzepts. In diesem Unterkapitel werden die Versionen der einzelnen Prototypen sowie derer Entwicklung, begleitet mit Bildmaterialien, beschrieben und erläutert.

5.1. Zielsetzung

Die Voraussetzungen an die ausgewählte Hardware soll genauso wie die Software unter dem Stichwort Rapid-Prototyping stehen. Dafür werden bestimmte Anforderungen an die Hardware gestellt.
Da es sich bei dem Framework und der Hardware um ein Open-Source-Projekt handelt, ist es außerdem wichtig die nachfolgenden Faktoren zu beachten.
Entsprechend unserer Zielsetzung des Rapid-Prototyping ist es essentiell, dass die ausgewählten Komponenten schnell und international verfügbar sind und eine schnelle Realisierung des HMD gewährleistet werden kann. Da es sich bei der Hardware um ein modifiziertes Cardboard der Version 2 handelt, also um ein HMD, dürfen ausgewählten Komponenten nicht zu schwer sein. Zudem müssen die Eye-Tracking-Kameras in das Cardboard passen und befestigt werden können. Die ausgewählten Komponenten der einzelnen Iterationen der Hardware-Entwicklung können dem nachfolgenden Kapitel 5.3 entnommen werden. Weiterhin sollen die Bestandteile auch im vorgegebenen Rahmen der Kosten liegen, weil dies auch eine Anforderung des Rapid-Prototyping ist. Bei der Vorlage des zu cuttenden Cardboards muss darauf geachtet werden, dass es einfach zu reproduzieren ist. Es wird darauf verzichtet, schwierige und dadurch komplizierte „Knickungen“ zu wählen, um die Einfachheit der Vorlage widerzuspiegeln. Somit ist eine weitere Anforderung, die als Zielsetzung der Hardware gesetzt wurden ist, die einfache Handhabung und Umsetzung der Hardware. Der Kontext des Rapid-Prototyping wird verwendet, um ein relativ schnelles Ergebnis in Bezug auf die Machbarkeit zu erhalten.

5.2. Konzept

Zu Beginn des Projekts wurde ein grundlegendes Konzept für die Realisierung des HMDs entwickelt, welches in Abbildung 5.2.1 schematisch dargestellt wird. Als Basis für das HMD wird das Google Cardboard Version 2 genutzt und für die Nutzung mit Eye-Tracking modifiziert. Hierfür sollen eine bzw. zwei Kamera(s) in das Cardboard eingebaut werden, die auf jeweils ein Auge ausgerichtet sind. Das Kamerabild wird an die Eye-Tracking Software Pupil Capture übertragen, in der das Auge von der Software verfolgt werden kann. Anschließend werden die Eye-Tracking-Daten, die von Pupil Capture berechnet werden, von der VR/AR-Applikation auf dem Smartphone im Cardboard ausgelesen und für die Interaktionstechniken genutzt. Zusätzlich zu den Kamerabildern der Eye-Tracking-Kameras wird ein Video-Stream von der VR/AR-Szene der Applikation benötigt. Dieser Video-Stream wird ebenfalls in Pupil Capture eingebunden, sodass die Eye-Tracking Kalibrierung durchgeführt werden kann und der Nutzer einen Einblick hat, was der Proband sehen kann. Für AR-Anwendungen nutzt die Applikation die integrierte Smartphone-Kamera, um Video-see-through zu realisieren.

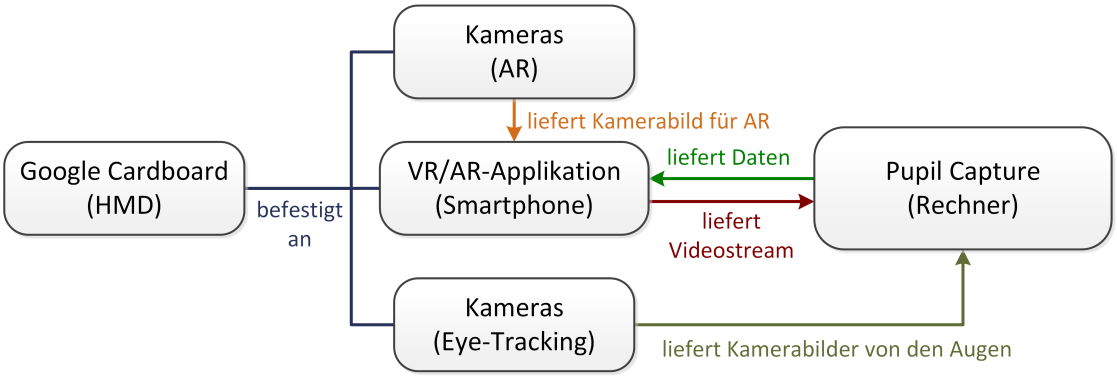

Abbildung 5.2.1.: Darstellung des grundlegenden Konzepts

Neben den Überlegungen zum Aufbau des Prototypen gab es ebenfalls Überlegungen für die Wahl des Übertragungsmedium für die jeweiligen Komponenten. Hier stehen hauptsächlich zwei Medien zur Wahl: USB und WLAN. Welches Übertragungsmedium für welche Komponente gewählt werden sollte, hängt vom Aufbau des Prototypen sowie den Anschlussmöglichkeiten der einzelnen Komponenten ab. Dabei gilt USB als stabil und schnell, jedoch ist der Freiraum für Bewegung wesentlich beschränkter als beim WLAN. Dafür ist die Bandbreite beim WLAN geringer als bei USB, was zu Latenzen führen kann. Welche Übertragungsmedien für die jeweiligen Prototypen verwendet wurden, werden im folgenden Unterkapitel 5.3 erläutert.

5.3. Umsetzung

Der folgende Abschnitt zeigt die vier Phasen der Entwicklung des Hardware-Prototypen auf. Dabei wird in jedem Kapitel auf den Aufbau, die Komponenten, die Kamera-Position, die Übertragung des Kamerabilds sowie die durchgeführten Hardwaretests eingegangen. Diese Veränderungen werden durch Bilder verdeutlicht.

5.3.1. Erster Prototyp - Standard Cardboard V2 mit Raspberry Pi Zero und Pi NoIR Kamera

Im ersten Quartal des Projekts lag der Fokus auf die Entwicklung einer ersten Version des Prototypen, mit der es möglich sein sollte, ein Kamerabild vom Auge in einer Eye-Tracking-Software einzubinden. In den folgenden Unterabschnitten werden die Bestandteile und der Aufbau des ersten Prototypen beschrieben und die Entscheidungen bzgl. dieser erläutert.

Aufbau

Der grundsätzliche Aufbau des ersten Prototypen ist in der Abbildung 5.3.1 dargestellt: Als Basis diente das Google Cardboard, an dem ein Raspberry Pi Zero, ein Raspberry Pi NoIR Kameramodul und eine IR-LED angebracht wurden.
Der Raspberry Pi ist ein kleiner Einplatinencomputer, der für den Einstieg in Linux, Programmierung und Hardware für Schüler entwickelt wurde. Er eignet sich für Projekte, bei denen ein kleines Rechnersystem benötigt wird und keine hohe Rechenleistung erforderlich ist. Hierbei ist der Raspberry Pi Zero die kleinste Version des Raspberry Pi.
Der Raspberry Pi Zero ist zudem mit einem Rechner verbunden, auf dem die Eye-Tracking-Software läuft und der Stream bzw. die Ausgabe stattfindet. Zusätzlich wird das Smartphone mit der AR/VR-Applikation in die entsprechende Halterung des Cardboards gelegt, sodass der Nutzer auf das Display blicken kann.
Abbildung 5.3.2 zeigt den entwickelten Prototypen von der Außen- und Innenseite. Der Raspberry Pi Zero ist an der Außenseite des Cardboards angebracht, während das Kameramodul und die IR-LED an der Innenseite angebracht sind. In der Klappe auf der Außenseite befindet sich außerdem ein Loch für die Kamera des Smartphones, sodass diese nicht abgedeckt wird. Auf der Innenseite des Cardboards ist ein Abstandshalter aus Papier angebracht, um den richtigen Abstand zwischen Augen und Kamera zu bewahren. Dies ist notwendig, da das Cardboard mit der Hand gehalten wird und somit die Position der Kamera zum Auge nicht konstant ist.

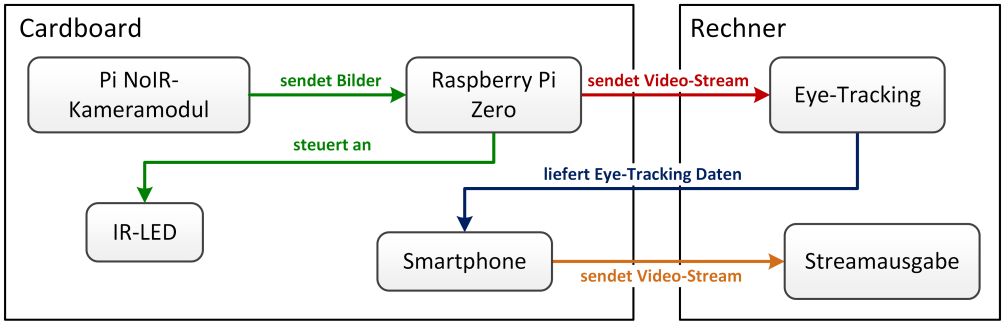

Abbildung 5.3.1.: Komponentendiagramm des ersten Prototypen

Abbildung 5.3.2.: Innenseite (links) und Außenseite (rechts) des ersten Prototypen

Kamera-Position

Eine gute Kamera-Position ist wichtig, da das Auge für die Eye-Tracking-Software gut erkennbar und die Pupille bei jeder Blickrichtung sichtbar sein muss. Hierbei wird für die erste Version des Prototypen monokulares Eye-Tracking verwendet, um es in der nächsten Version auf binokulares Eye-Tracking zu erweitern. Zudem stand die Frage im Raum, ob direktes oder indirektes Eye-Tracking verwendet werden soll. Beim direkten Eye-Tracking wird die Kamera direkt auf das Auge gerichtet, während beim indirekten Eye-Tracking ein Spiegel zur Umlenkung des Bilds genutzt wird. Dabei ist bei indirektem Eye-Tracking durch den Einbau eines Spiegels ein deutlich höherer Modifikationsaufwand an der Hardware zu erwarten. Zwar erkauft man sich dadurch theoretisch Platzvorteile bei der Positionierung der Kameras, doch dürfte der nötige Platz auch bei direktem Eye-Tracking vorhanden sein. Die Wahl fällt demnach auf das direkte Tracking. Hinzu kommt, dass dieses Verfahren kostengünstiger und somit mehr dem Rapid-Prototyping gerecht wird.

Wie es links in der Abbildung 5.3.2 zu sehen ist, ist das Raspberry Pi NoIR-

Kameramodul an der linken unteren Ecke der Innenseite des Cardboards angebracht. Hierfür wurde ein Loch in das Cardboard geschnitten, sodass das Kameramodul hineinpasst. Das Kameramodul ist klein und flach, wodurch es in einen relativ frontalen Winkel zum Auge ausgerichtet werden kann. Zusätzlich ist oberhalb des Kameramoduls eine einzelne IR-LED angebracht, um das Auge auszuleuchten. Das resultierende Kamerabild des Auges ist in der Abbildung 5.3.3 zu sehen: Das Auge wird von der LED beleuchtet, sodass das Licht der LED im Auge reflektiert wird und die Pupille selbst klar erkennbar ist. Durch den Abstandshalter wird sichergestellt, dass sich das Auge nicht zu weit am Rand befindet.

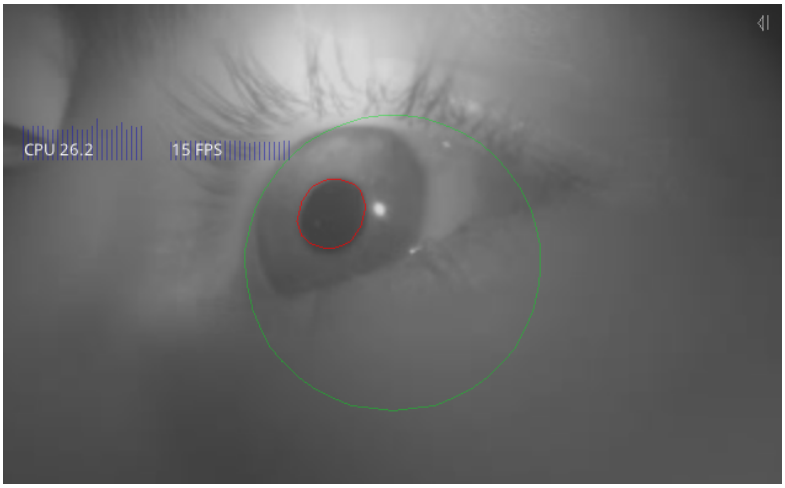

Abbildung 5.3.3.: Bild des Auges aufgenommen durch das NoIR-Kameramodul

Übertragung des Kamerabilds

Für die Übertragung des Kamerabildes zur Eye-Tracking Software Pupil Capture wird der Raspberry Pi Zero genutzt. Das Raspberry Pi NoIR-Kameramodul ist am Raspberry Pi Zero angeschlossen, sodass der Raspberry Pi Zero auf die Kamerabilder zugreifen kann. Um diese Kamerabilder auch für einen externen Zugriff zur Verfügung zu stellen, wird das Skript Pi Video-Streaming genutzt. Dadurch werden die Kamerabilder an einen festgelegten Port vom Raspberry Pi Zero zur Verfügung gestellt. Da der Raspberry Pi Zero kein Wifi-Modul besitzt, muss es per Micro-USB Kabel mit dem Rechner verbunden werden, damit die Eye-Tracking-Software über die IP des Pi Zero auf die Kamerabilder zugreifen kann.

Tests

Zur Sicherstellung der grundlegenden Funktionalitäten des ersten Prototypen wurden Hardware-Tests durchgeführt. Diese werden sowohl durch eine Anforderung, einem Zustand und Fotos dokumentiert. Die einzelnen Hardware-Tests werden in folgenden Tabellen aufgeführt.

Testbezeichnung	Funktion der Infrarot-LED
Anforderung	Die IR-LED soll bei eingeschaltetem Raspberry Pi Zero leuchten.
Zustand	Die IR-LED beginnt zu leuchten, kurze Zeit nachdem der Raspberry Pi Zero mit Strom versorgt wird. Die leuchtende IR-LED ist in der Abbildung 5.3.4 zu sehen.

Tabelle 5.1.: Prototyp Version 1, Test 1: Funktion der Infrarot-LED

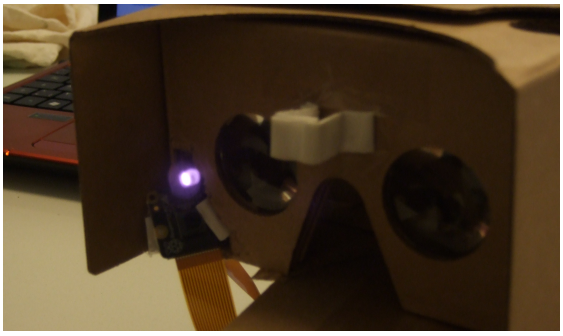

Abbildung 5.3.4.: Die leuchtende Infrarot-LED

Testbezeichnung	Kamera-Funktion
Anforderung	Das NoIR-Kameramodul soll ein Bild aufnehmen können.
Zustand	Das Kameramodul ist aktiv und nimmt Bilder auf, die mit Hilfe eines Testskripts im Browser betrachtet werden können. In der Abbildung 5.3.5 ist die Ansicht der Kamerabilder im Browser dargestellt.

Tabelle 5.2.: Prototyp Version 1, Test 2: Kamera-Funktion

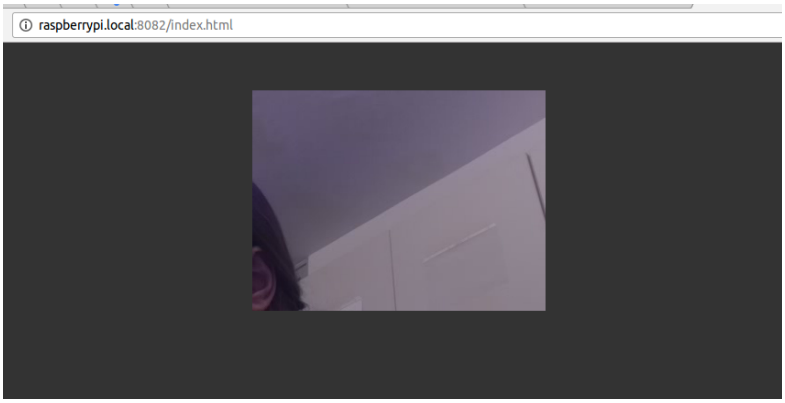

Abbildung 5.3.5.: Ein Bild des Raspberry Pi Zero Kameramoduls, das über das Testskript im Browser angezeigt wird

Testbezeichnung	Bildübertragung an Pupil Capture
Anforderung	Das Kamerabild soll an Pupil Capture gesendet werden.
Zustand	In Pupil Capture kann das Kameramodul als Quelle ausgewählt und die Bilder betrachtet werden, so wie es in der Abbildung 5.3.6 zu sehen ist.

Tabelle 5.3.: Prototyp Version 1, Test 3: Bildübertragung an Pupil Capture

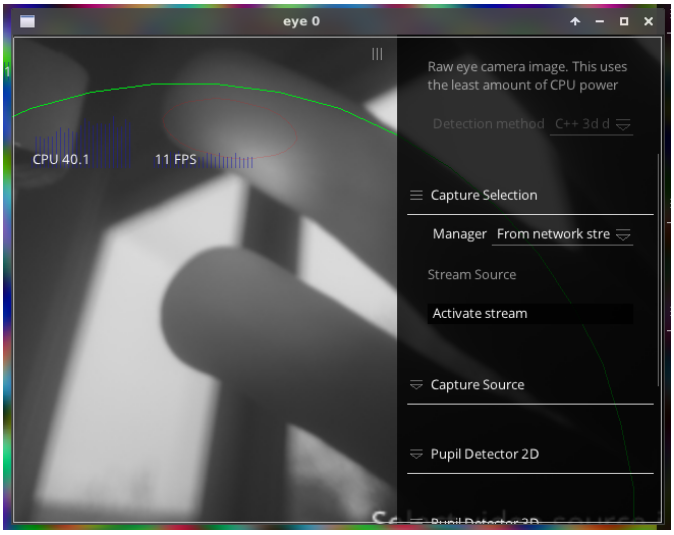

Abbildung 5.3.6.: Ein Bild des NoIR-Kameramoduls in Pupil Capture

Testbezeichnung	Kamera-Ausrichtung
Anforderung	Die Kamera soll auf das Auge ausgerichtet sein, sodass das Auge in Pupil Capture erkannt wird.
Zustand	Bei aufgesetztem HMD ist das Auge auf dem Kamerabild gut zu erkennen. Das Auge ist zentral im Bild und wird in Pupil Capture als solches erkannt. In Abbildung 5.3.7 ist die Betrachtung des Kamerabild in Pupil Capture dargestellt.

Tabelle 5.4.: Prototyp Version 1, Test 4: Kamera-Funktion

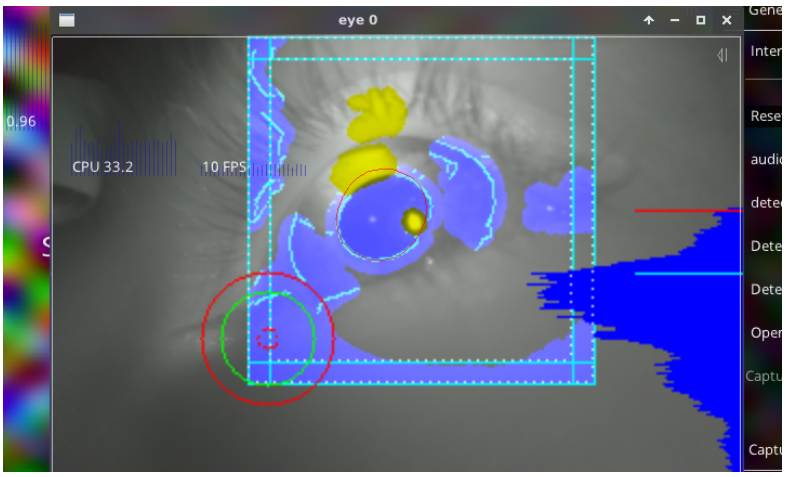

Abbildung 5.3.7.: Ein Bild des erkannten Auges in Pupil Capture

Testbezeichnung	Kamera-Funktion
Anforderung	Das Smartphone (Nexus 5) soll sicher in die dafür vorgesehene Halterung passen.
Zustand	Das Smartphone passt in die Halterung und verrutscht dort nicht.

Tabelle 5.5.: Prototyp Version 1, Test 5: Smartphone-Halterung

5.3.2. Zweiter Prototyp - Experimente mit unterschiedlicher Hardware

Der zweite Prototyp wurde im zweiten Quartal unseres Projekts entwickelt. Aufgrund der Kameraposition des ersten Prototypen wurde hierauf besonderes Augenmerk gelegt, um die Augenerkennung zu optimieren. Daher wurde versucht das Raspberry Pi Modell auszutauschen und dieses sinnvoller zu nutzen.

Aufbau

Die zweite Version des Prototypen besteht aus mehreren Unterversionen. Die erste Idee, die aufkam, war es den Pi Zero durch einen Pi 3 auszutauschen.
Da der Pi Zero eine schlechte Verfügbarkeit aufweist und zudem nur die Aufgabe hat, das Kamerabild weiter zu leiten wurde nach Alternativen gesucht. Dabei bot sich der Pi 3 an, da dort eine Auslagerung der Eye-Tracking-Software stattfinden könnte. Des Weiteren ist die Beschaffung deutlich leichter. Durch die Auslagerung könnte das Nutzen eines externen Rechners umgangen werden, wodurch sich der Proband flexibler und mobiler bewegen könnte.
Der grundlegende Aufbau kann dem Komponentendiagramm in Abbildung 5.3.8 entnommen werden. Hierbei diente das Google Cardboard ebenfalls als Grundlage. Die Pi NoIR Kameramodule wurden durch USB-Eye-Tracking Kameras ersetzt. Diese senden Bilder an den Pi 3. Weiterhin steuert der Pi 3 die IR-LEDs. Des Weiteren sollte die Eye-Tracking Software Pupil Capture durch den Pi 3 ausgeführt werden und die IR-LEDs betreiben, die wiederum Eye-Tracking-Daten für das Smartphone liefert. Das Smartphone sendet den erhaltenen Video-Stream an Pupil Capture. Zudem kommuniziert der Raspberry Pi 3 über eine Remoteübertragung mit dem Rechner um eine Bildausgabe zu erhalten. Außerdem ist in dieser Ansicht zu erkennen, dass mit zwei AR-Kameras gearbeitet werden sollte, um stereoskopisches Sehen zu ermöglichen.
Durch die Verwendung des Pi 3 sollte die Möglichkeit bestehen Pupil Labs bzw. Pupil Capture auszulagern, um den Prototypen mobiler zu gestalten. Letztendlich wurde sich aber gegen eine Auslagerung entschieden, da zu viele Probleme während der Versuche aufgetreten sind.
Beim Testen der Komponenten miteinander sind einige Probleme aufgetaucht, die die Umsetzung dieser Variante erschwerten. Nachfolgend werden einige Probleme aufgelistet:

- Sehr lange Installation der Abhängigkeiten
- Speicherprobleme
- Performanzeinbrüche

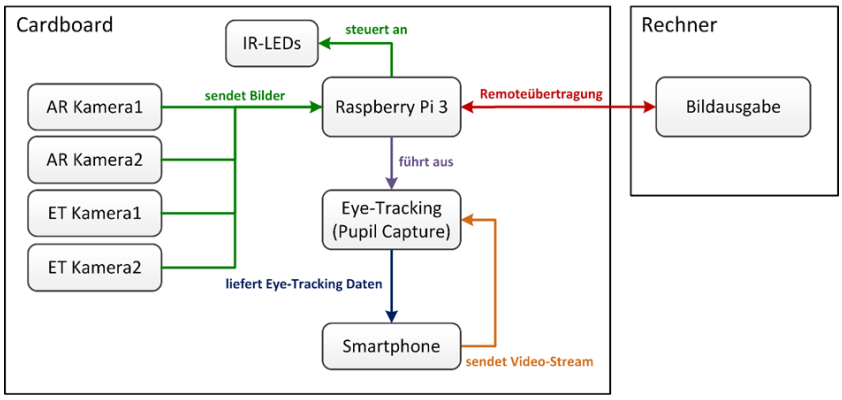

Abbildung 5.3.8.: Komponentendiagramm des zweiten Prototypen, Version 1

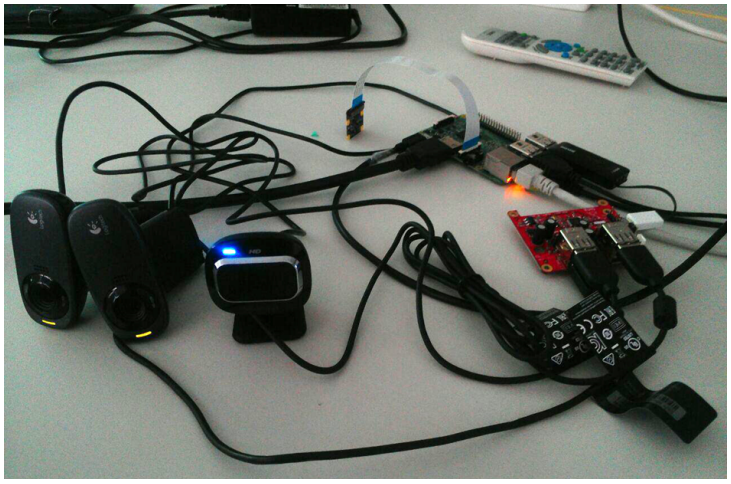

Abbildung 5.3.9.: Versuchsaufbau Pi3 mit drei Webcams und einem Pi Kameramodul

Das Ganze kann zusätzlich aus dem Versuchsaufbau in Abbildung 5.3.9 entnommen werden.
Es gäbe zwar die Möglichkeit an Stelle von Pupil Capture den Pupil Service zu nutzen, der die selben Daten liefert, allerdings ohne Benutzeroberfläche, womit der Pi 3 eine bessere Performance erreichen könnte. Jedoch wurde sich dagegen entschieden, da das vorgesehene Konzept Pupil Capture nicht nur für das Eye-Tracking nutzt, sondern zusätzlich als Methode, die dem Wissenschaftler die Möglichkeit bietet zu sehen was der Proband sieht. Da der Pi 3 hierbei sonst nur als USB-Hub verwendet werden würde, wurde sich dagegen entschieden und stattdessen ein physikalischer USB-Hub eingesetzt.
Des Weiteren wurden in diesem Schritt die Eye-Tracking-Kameras gewechselt, da

der Pi 3 nur einen Anschluss für eine Pi Kamera hat. Zunächst wurde versucht mittels Multiplexer zwei Kameras zu betreiben, das Ergebnis lieferte jedoch keinen zufriedenstellenden Video-Stream. Es wurde festgestellt, dass sich immer nur eine Kamera zur gleichen Zeit ansteuern lässt und die Verzögerung beim Wechseln bzw. der Aufnahme der jeweiligen Bilder zu hoch ist. Die Bilder werden somit immer asynchron gesendet, weil nicht beide Kamera gleichzeitig ein Bild aufnehmen können. Deshalb wurden die Pi Kameramodule durch zwei USB-Kameras für das Eye-Tracking ersetzt.
Deswegen wurde sich ein alternatives Konzept überlegt. Dieses kann aus Abbildung 5.3.10 entnommen werden. Hierbei bestehen wieder die beiden großen Faktoren Google Cardboard und ein externer Rechner. Anstelle des Pi 3 wurde mit einem USB-Hub gearbeitet, der am Rechner angeschlossen ist und somit Strom für die Eye-Tracking Kameras, den IR-LEDs und den AR-Kameras liefert. Die Kameras wiederum liefern Bilder. Die Eye-Tracking-Kameras liefern zudem Strom für die IR-LEDs. Die Eye-Tracking Software Pupil Capture ist auf dem Rechner installiert und liefert die Eye-Tracking-Daten an das Smartphone im Cardboard, welches wiederum den Video-Stream an Pupil Capture sendet.
Im späteren Verlauf der Entwicklung des zweiten Prototypen wurde die zwei AR-Kameras verworfen, da der Aufwand zu groß erschien und sich erst einmal auf ein funktionierendes Gesamtsystem fokussiert wurde.

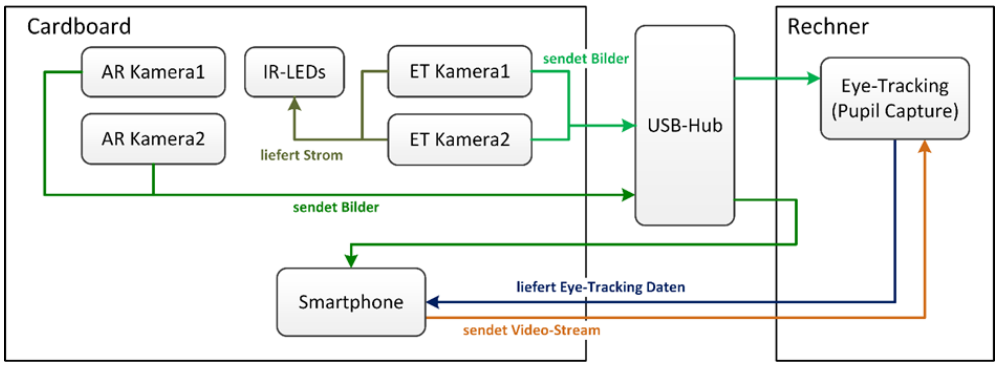

Abbildung 5.3.10.: Komponentendiagramm des zweiten Prototypen, Version 2

Kamera-Position

Um eine Verbesserung der Kamera-Position und somit ein besseres Augenbild zu erhalten, wurden verschiedene Ansätze ausprobiert. Diese werden im Folgenden beschrieben.

Erster Ansatz – Seitliche Kamera-Position

Für den seitlichen Ansatz wurde auf Basis des ersten Prototypen gearbeitet. Hierfür wurde ein Loch in die Seite des Cardboards geschnitten und die USB-Kamera darin platziert, wie es in Abbildung 5.3.11 ersichtlich ist. Da durch diese Position zwar das Auge einigermaßen gut erkannt wurde, allerdings nicht die verschiedenen Richtungen in die das Auge blickte, wurde dahingehend nach einer weiteren alternativen Kamera-Position gesucht. Des Weiteren war oftmals die Linse im Weg der Kamera, wodurch die Eye-Tracking-Daten nicht flüssig waren.

Abbildung 5.3.11.: Seitlicher Ansatz der Kamera-Position

Zweiter Ansatz – Kamera-Position unter dem Auge durch zusätzliche Stützen

Als wir nach einer alternativen Position gesucht haben, fiel dabei auf, dass bei dem Eye-Tracker von Pupil Labs selbst die Bilder so aussahen, als wären sie von weiter unten bzw. unterhalb des Auges aufgenommen worden. So wurde entschieden, diese Position der Kamera auszuprobieren und die Kamera wurde direkt unter der Linse platziert und zum Auge ausgerichtet.
Zunächst wurde als Halterung für die Kamera ein Stück Pappe verwendet, das zu einem Dreieck gefaltet und an der Innenseite des Cardboards befestigt wurde. Durch die dreieckige Form konnte der Winkel angepasst werden. Da allerdings die Pappe allein nicht sehr stabil ist, wurde sie mit Draht verstärkt, um das Gewicht der Kamera besser halten zu können. Eine Skizze zu dieser Halterung ist in Abbildung 5.3.12 sowie auf den Bild 5.3.13 zu sehen. Dabei ist links der Aufbau im Cardboard und rechts die Verstärkung der Pappe durch Draht abgebildet. Allerdings schien die Position noch nicht optimal zu sein, da der Winkel zum Auge für unterschiedliche Probanden noch nicht dynamisch genug war und nur auf ei-

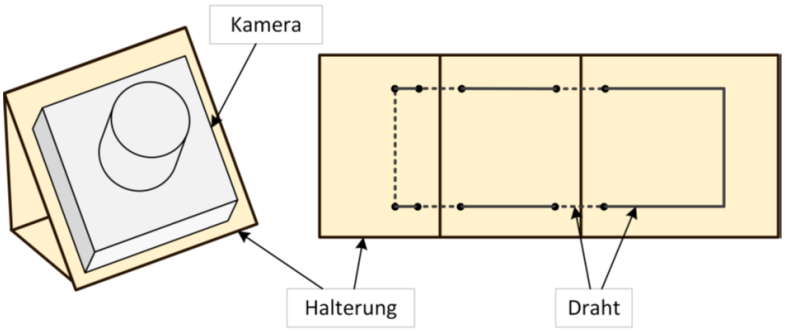

Abbildung 5.3.12.: Skizze des zweiten Ansatzes der Kamera-Position

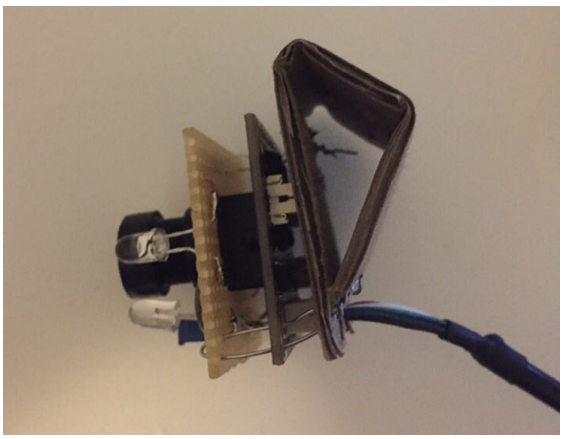

Abbildung 5.3.13.: Bild des zweiten Ansatzes der Kamera-Position

ner Achse geändert werden konnte. Daraufhin wurde die Pappe entfernt und die Kamera nur mit Drähten an das Cardboard befestigt. Die Konstruktion mit den Drähten und zwei Kameras ist in Abbildung 5.3.14 und 5.3.15 dargestellt. Dabei dienen die gezwirbelten Drähte als flexible Halterung für die Kameras. Mit dieser Halterung sollte das Problem der verschiedenen Gesichtsproportionen der Probanden gelöst werden, indem die Kameras individuell auf die Augen des Probanden ausgerichtet werden können. Während des Testens wurde jedoch festgestellt, dass die Drähte allein noch zu instabil sind und die Kameras wurden mithilfe von zwei Drähten und eines Schlitzes an der Seitenwand fixiert (vgl. 5.3.16). Diese Kameraposition war gering verstellbar und passte relativ gut für verschiedene Augen, als jedoch zusätzliches Gewicht durch das Kabel der LED-Platine hinzukam, hielt diese Halterung dem Gewicht nicht stand und es musste nach einer weiteren Lösung gesucht werden.

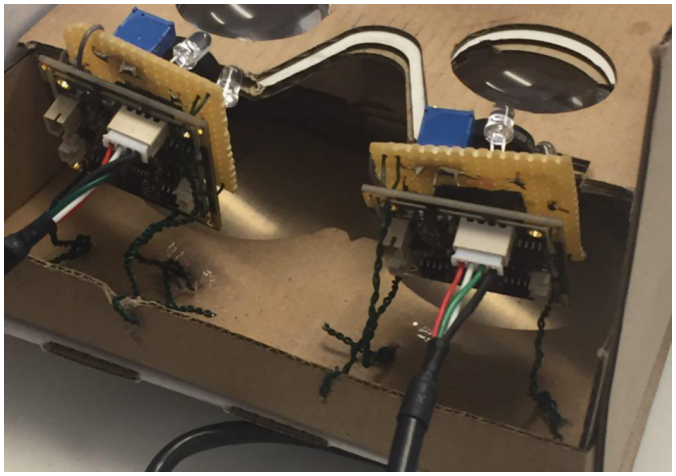

Abbildung 5.3.14.: Kamera-Position im Cardboard(1)

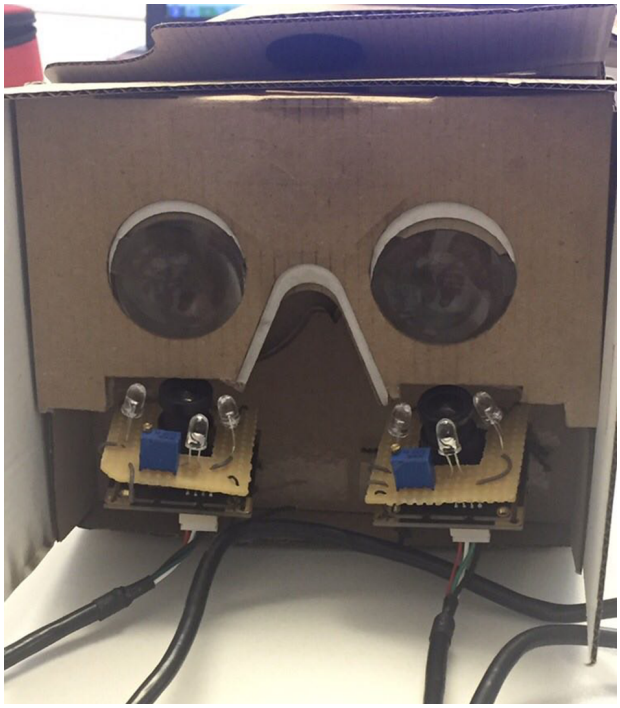

Abbildung 5.3.15.: Kamera-Position im Cardboard(2)

LEDs

Da in dieser Iteration der Entwicklung des Prototypens vorerst die Position der Kamera im Vordergrund stand, wurde die Positionierung der LEDs vernachlässigt. Für die zweite Version des Prototypens wurde zunächst drei IR-LEDs mit einem

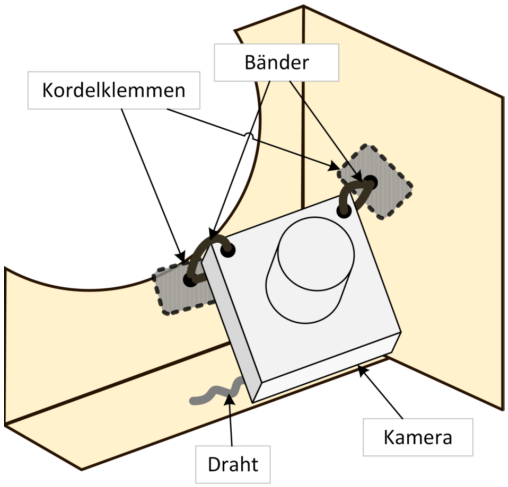

Abbildung 5.3.16.: Alternative Kameraposition mit Befestigung an den Cardboard-Wänden

Abstrahlwinkel von 3 ° verwendet. Diese LEDs sowie ein 18 Ohm Widerstand sind durch Kabel miteinander verbunden (siehe Abbildung 5.3.17). Da der Prototyp

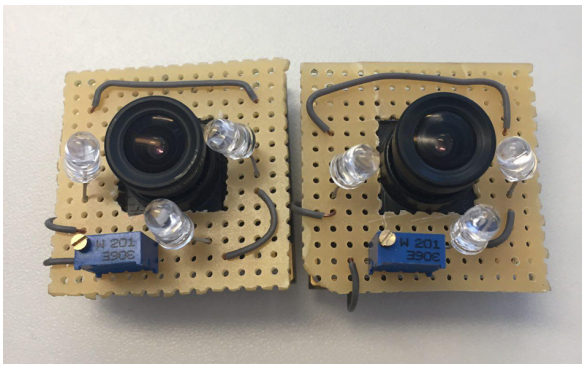

Abbildung 5.3.17.: Platinen mit jeweils drei IR-LEDs

nun USB-Kameras für das Eye-Tracking und kein Raspberry Pi mehr verwendet, erfolgt die Spannungszufuhr von 5V durch die USB-Kamera. Die USB-Kamera besitzt zusätzlich zu ihren Anschlüssen auf der Unterseite insgesamt sechs Pins, auf der Oberseite vier Pins für den USB-Anschluss und zwei Pins als optionale Spannungsquelle mit einer Spannung von 5V. An diese Pins wurde die IR-LEDs angeschlossen. Es wurden Löcher in das Cardboard gebohrt, in denen die LEDs hindurch gesteckt werden, so wie es in der Abbildung 5.3.2 für den ersten Ansatz der Kameraposition aus dem vorherigen Kapitel bereits zu sehen war. Allerdings konnten die IR-LEDs lose nicht auf das Auge ausgerichtet werden, sodass es nicht

korrekt beleuchtet wurde. Die Position, die Anzahl sowie der Abstrahlwinkel der LEDs werden im der dritten Variante genauer untersucht.

Übertragung des Kamerabilds

Die Übertragung des Kamerabilds erfolgt über die USB-Anschlüsse und dem USB-Hub. Die Eye-Tracking-Kameras sind mit dem USB-Hub verbunden, welcher am Rechner angeschlossen ist und auf dem die Eye-Tracking Software Pupil Capture läuft. Die Kameras können dann in Pupil Capture als Quellen ausgewählt werden.

Tests

Zur Sicherstellung der grundlegenden Funktionalitäten des zweiten Prototypen wurden Hardware-Tests durchgeführt. Diese werden sowohl durch eine Anforderung, einem Zustand und Fotos dokumentiert. Die einzelnen Hardware-Tests werden in folgenden Tabellen aufgeführt.

Testbezeichnung	Funktion der Infrarot-LEDs
Anforderung	Die drei IR-LEDs sollen beim Einstecken der USB-Kamera leuchten.
Zustand	Die drei LEDs fangen an zu leuchten sobald die USB-Kamera mit Strom versorgt ist (vgl. Abbildung 5.3.18).

Tabelle 5.6.: Prototyp Version 2, Test 1: Funktion der Infrarot-LEDs

Testbezeichnung	Kamera-Funktion
Anforderung	Es soll durch die USB-Kamera ein Bild aufgenommen werden.
Zustand	Sobald die Kamera angeschlossen ist, nimmt die USB-Kamera ein Bild der Pupille auf.

Tabelle 5.7.: Prototyp Version 2, Test 2: Funktion der Infrarot-LEDs

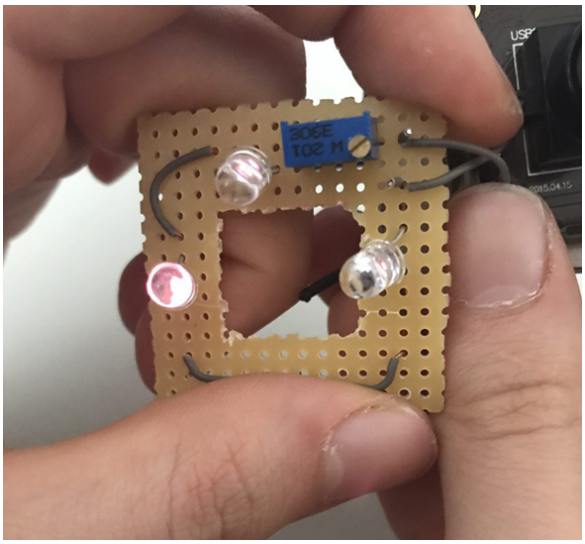

Abbildung 5.3.18.: Drei LEDs leuchten

Testbezeichnung	Bildübertragung an Pupil Capture
Anforderung	Übertragung des Bildes an Pupil Capture.
Zustand	Die USB-Kamera wird von Pupil Capture erkannt und kann ausgewählt werden und somit findet eine Bildübertragung statt.

Tabelle 5.8.: Prototyp Version 2, Test 3: Bildübertragung an Pupil Capture

Testbezeichnung	Kamera-Ausrichtung
Anforderung	Die USB-Kamera soll ein gutes Bild des Auges bzw. der Pupille liefern.
Zustand	Das Bild des Auges wird in der Eye-View von Pupil Capture erkannt (vgl. 5.3.19).

Tabelle 5.9.: Prototyp Version 2, Test 4: Kamera-Ausrichtung

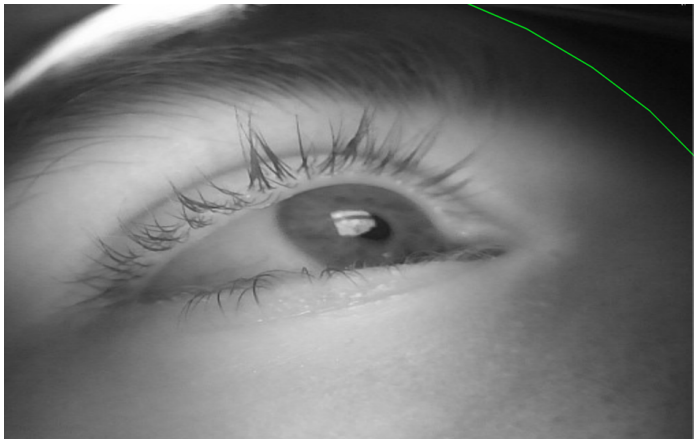

Abbildung 5.3.19.: Position des Auges in der Eye-View von Pupil Capture

Testbezeichnung	Smartphone-Halterung
Anforderung	Das Smartphone soll an einer vorgesehenen Stelle am Cardboard platziert werden.
Zustand	Das Smartphone passt an die vorgesehene Stelle des Cardboards.

Tabelle 5.10.: Prototyp Version 2, Test 5: Smartphone-Halterung

Testbezeichnung	Eye-Tracking
Anforderung	Das Auge soll gut von Pupil Capture erkannt werden.
Zustand	Da die Position noch nicht optimal ist, wird das Auge zwar erkannt, dennoch verliert Pupil Capture des Öfteren den Fokus.

Tabelle 5.11.: Prototyp Version 2, Test 6: Eye-Tracking

Testbezeichnung	LED-Ausrichtung
Anforderung	Die LEDs sollen das Auge gut ausleuchten.
Zustand	Das Auge wird zwar ausgeleuchtet, jedoch nicht optimal. Zudem ist es schwierig die LEDs zu positionieren bzw. auszurichten.

Tabelle 5.12.: Prototyp Version 2, Test 7: LED-Ausrichtung

5.3.3. Dritter Prototyp - Eye-Tracking Kameraposition optimieren

Die dritte Version des Prototyps entstand gegen Ende des zweiten Quartals. Der Fokus lag dabei auf einer stabileren Kamerahalterung, da diese in der zweiten Version zwar flexibel jedoch zu instabil ist sowie auf dem Wegfall der zwei angedachten AR-Kameras.

Aufbau

Wie in der Abbildung 5.3.20 zu sehen ist, hat sich der Aufbau bzgl. der Komponenten zur zweiten Version nicht stark verändert. Der Hauptunterschied liegt hierbei bei der IR-LED Platine: Diese wird nicht mehr über die Kamera, sondern über ein Mirco-USB-Kabel, das an dem Hub angeschlossen ist, mit Strom versorgt. Nach mehreren Tests mit dem zweiten Prototypen musste festgestellt werden, dass

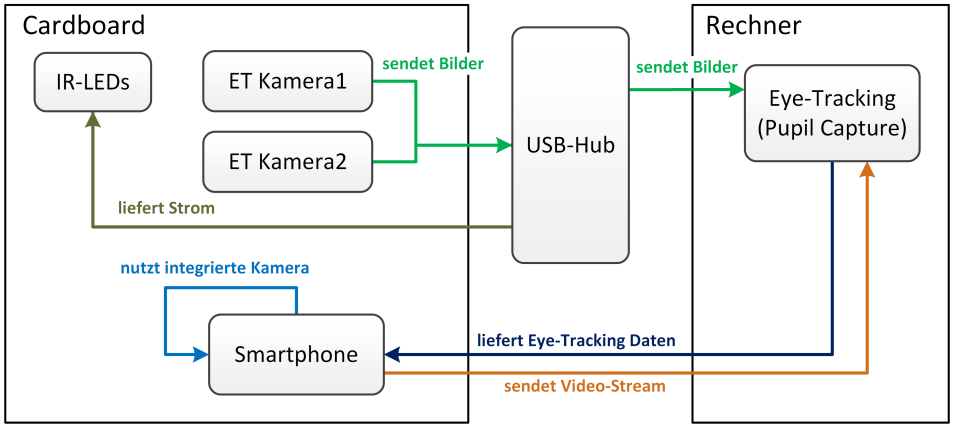

Abbildung 5.3.20.: Komponentendiagramm des dritten Prototypen

der 3° Abstrahlwinkel der verwendeten IR-LEDs zu klein zum Beleuchten des Auges ist. Aus diesem Grund wurden IR-LEDs mit einem Abstrahlwinkel von 140° und 90° beschafft und getestet. Es wurden zwei Platinen mit jeweils zwei dieser IR-LEDs gelötet und ihre Leuchtkraft überprüft. Sowohl 140° als auch 90° beleuchteten den ganzen Bereich im Kamerabild, jedoch viel zu schwach. Abbildung 5.3.21 verdeutlicht den Unterschied des Radius und der Intensität des Lichtpunkts der 3° und 90° IR-LEDs. Daraufhin wurden verschiedene IR-LEDs mit unterschiedlichen Abstrahlwinkel besorgt: 10°, 12°, 15°, 20° und 30°. Mit Hilfe eines Breadboards wurden die IR-LEDs einzeln nacheinander auf ihre Leuchtkraft getestet. Der Versuchsaufbau ist in der Abbildung 5.3.22 und die Leuchtpunkte der einzelnen IR-LEDs sind in Abbildung 5.3.23 zu erkennen. Daraus ergab sich, dass

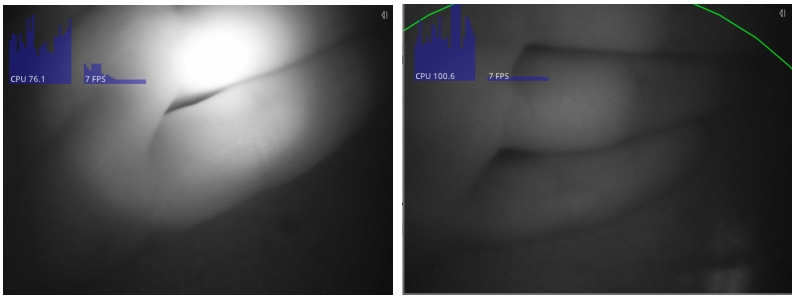

Abbildung 5.3.21.: Vergleich der Abstrahlwinkel. Links: 3 Grad, Rechts: 90 Grad

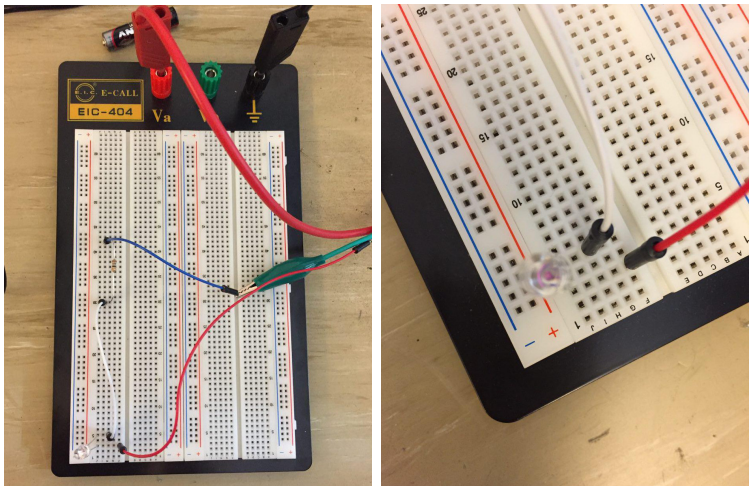

Abbildung 5.3.22.: Erster Versuchsaufbau für die LED-Tests

sechs dieser IR-LEDs das Auge gut genug ausleuchten. Jedoch stellte sich hier die Frage der Spannungs- und Stromversorgung, da die 5V und 120mA von der USB-Kamera für sechs LEDs nicht ausreicht. Es musste also eine externe Spannungsquelle genutzt werden. Dabei fiel die Entscheidung auf einen USB-Anschluss. Ein USB-Anschluss liefert 5V Spannung und 500mA Strom. Für eine Parallelschaltung mit drei Zweigen reicht dies für die IR-LEDs aus. Das Design der Platine wurde hierfür angepasst, sodass die Parallelschaltung mit den sechs IR-LEDs auf die Platine passen. Für den USB-Anschluss ist gedacht eine Micro-USB-Buchse zu verwenden, damit ein normales Micro-USB-Kabel für die Spannungsversorgung genutzt werden kann. Vorerst wurde dies mit den bisherigen Rasterlochplatinen umgesetzt. Die endgültig entwickelten Platinen sind in der Abbildung 5.3.24 zu sehen, wobei nun geätzte Platinen statt Rasterlochplatinen verwendet werden. Dadurch können sie kompakter bestückt werden und sind somit kleiner. Äußerlich hat sich der Prototyp im Ganzen nicht stark verändert. Die Außen- und Innenseite

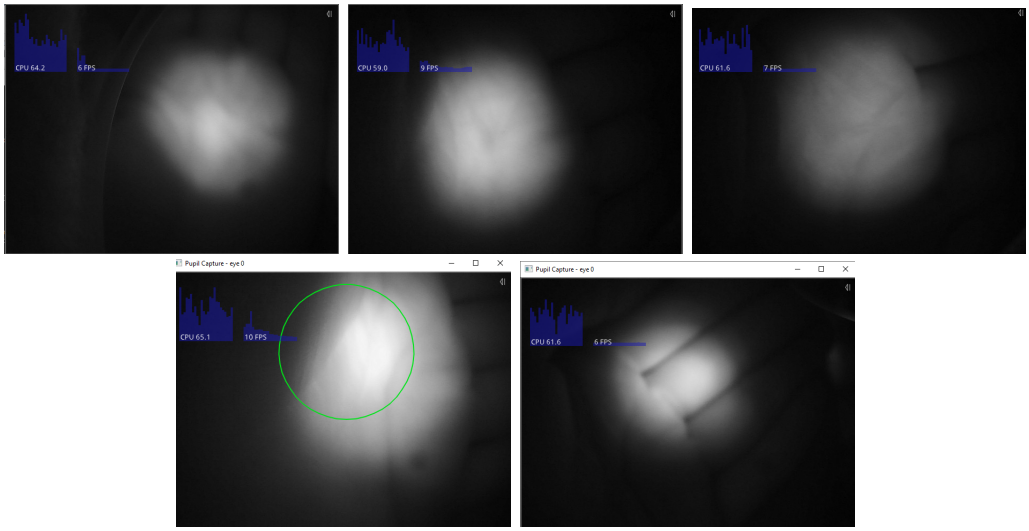

Abbildung 5.3.23.: Vergleich der Leuchtkraft der verschiedenen LEDs. Oben Links: 10 Grad, oben Mitte: 12 Grad, oben Rechts: 15 Grad, unten Links: 20 Grad, unten Rechts: 30 Grad

des dritten Prototypen können in Abbildung 5.3.25 betrachtet werden. Die größten Veränderungen im Design sind hierbei die Kamerahalterung, auf die im nächsten Abschnitt näher eingegangen wird, und die Kopfhalterung. Die Kopfhalterung besteht aus drei gezwirbelten Gummibändern, jeweils ein für links, rechts und oben, die an einem Ende am Cardboard befestigt sind. Am andere Ende sind die Gummibänder mit einem Kordelstopper gebündelt, sodass der Träger des Cardboards die Länge der Gummibänder an seiner Kopfgröße anpassen kann. Durch die Kopfhalterung muss das Cardboard nicht mehr festgehalten werden, wodurch sich auch die Kameraposition während des Tragens nicht ständig verschiebt.

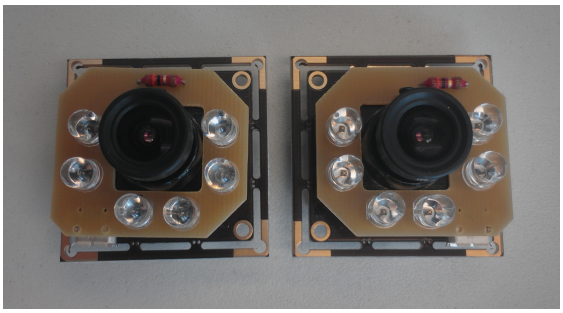

Abbildung 5.3.24.: Die geätzten Platinen des dritten Prototyps

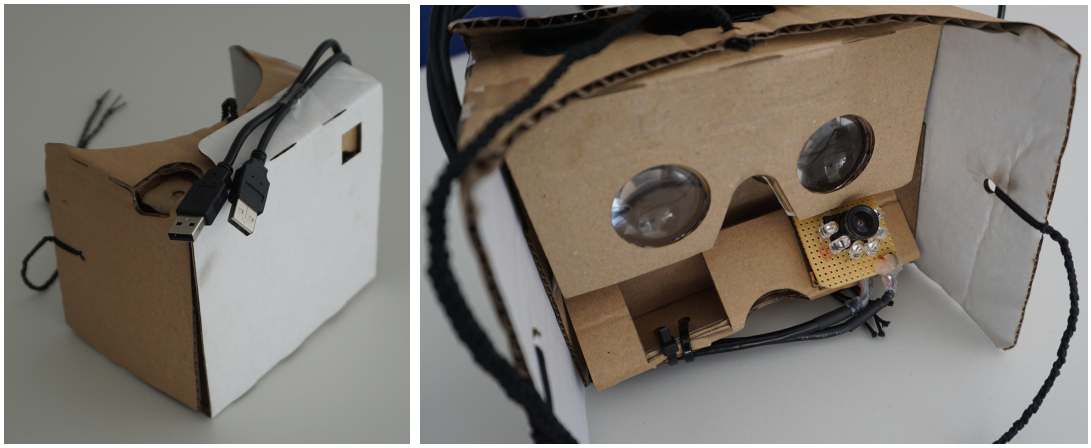

Abbildung 5.3.25.: Innenseite (links) und Außenseite (rechts) des dritten Prototypen

Kamera-Position

Die Kombination von Draht und Kordelklemme erlaubt eine flexible Einstellung der Ausrichtung der Kamera. Allerdings war der Winkel noch nicht optimal und die Kamera wurde wieder direkt unter die Linse gesetzt, diesmal höher und somit näher an der Linse. Als Halterung wurde wieder Pappe als Dreieck gefaltet. Im Gegensatz dazu, ist die Breite der Halterung so breit wie das Cardboard selbst, sodass die Halterung in das Cardboard reingesteckt werden kann. Abbildung 5.3.26 zeigt die Konstruktion, hierbei wird die Kamera mittels Bändern und Kordelstoppern an der Halterung befestigt. Die somit erreichte Kameraposition und der Winkel sind

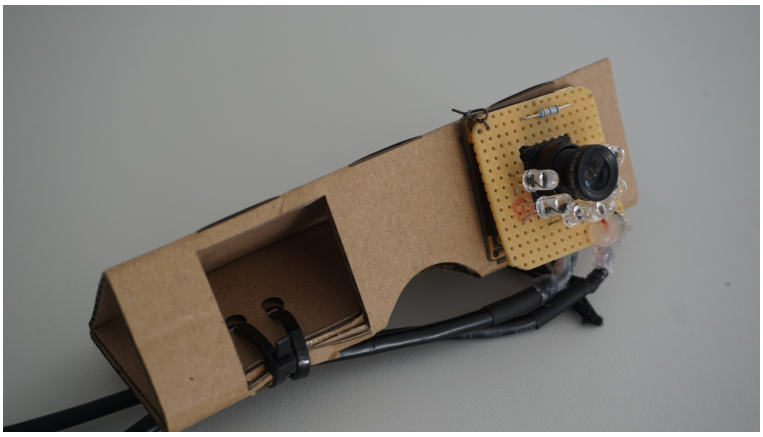

Abbildung 5.3.26.: Kamerahalterung des dritten Prototypen

für viele getestete Augenpositionen geeignet, sodass Pupil Capture das Auge tra-

cken kann. Abbildung 5.3.27 zeigt das resultierende Kamerabild mit verschiedene Blickrichtungen. Zusätzlich ist diese Halterung einfach nachzubauen.

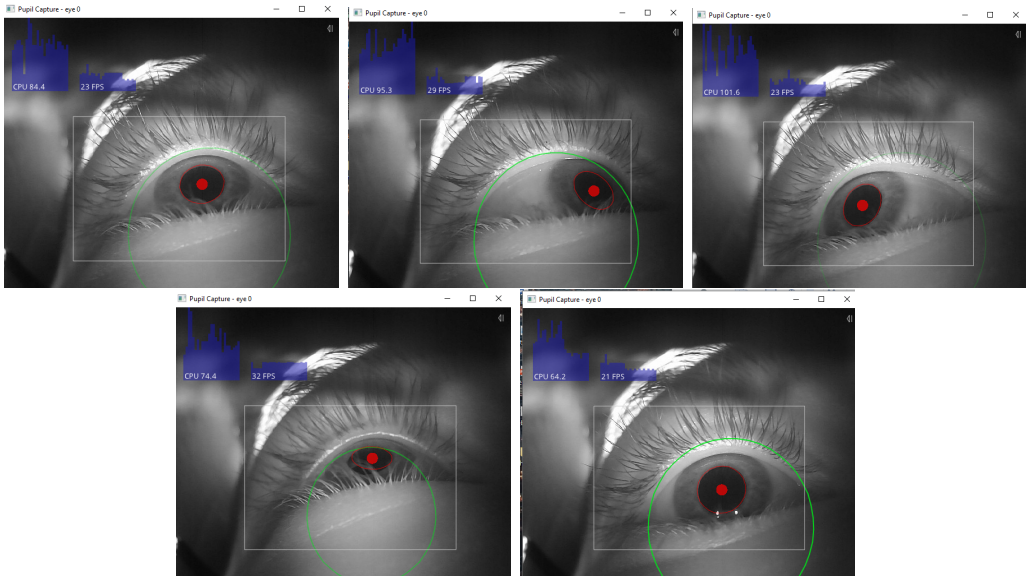

Abbildung 5.3.27.: Position des Auges mit der Kameraposition des dritten Prototypen

Übertragung des Kamerabilds

Die Übertragung des Kamerabilds hat sich zur zweiten Version nicht geändert: Die Eye-Tracking-Kameras sind an einem USB-Hub angeschlossen. Der USB-Hub ist wiederum an einem Rechner angeschlossen, auf den die Eye-Tracking Software Pupil Capture läuft.

Tests

Auch beim dritten Prototypen werden, zur Sicherstellung der grundlegenden Funktionalitäten, Hardware-Tests durchgeführt. Diese werden sowohl durch eine Anforderung, einem Zustand und Fotos dokumentiert. Die einzelnen Hardware-Tests werden in folgenden Tabellen aufgeführt.

Testbezeichnung	Funktion der Infrarot-LEDs
Anforderung	Die sechs Infrarot-LEDs sollen beim Einstecken der USB-Kamera leuchten.
Zustand	Die Infrarot-LEDs beginnen zu leuchten sobald die USB-Kamera mit Strom versorgt wird. In der Abbildung 5.3.28 sind die sechs leuchtenden IR-LEDs auf der Platine zu sehen.

Tabelle 5.13.: Prototyp Version 3, Test 1: Funktion der Infrarot-LEDs

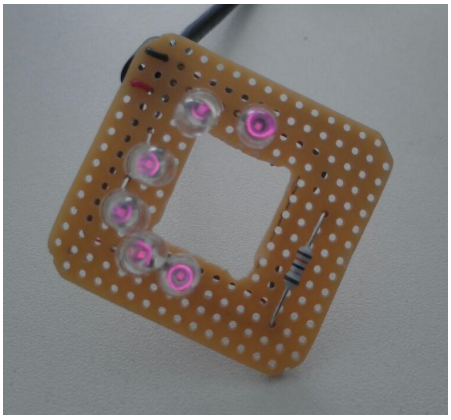

Abbildung 5.3.28.: Die leuchtende Infrarot LEDs

Testbezeichnung	Kamera-Funktion
Anforderung	Die USB-Kamera soll ein Bild aufnehmen können.
Zustand	Die USB-Kamera ist aktiv und nimmt Bilder auf, die über einen Viewer betrachtet werden können. Abbildung 5.3.29 zeigt ein Kamerabild der USB-Kamera.

Tabelle 5.14.: Prototyp Version 3, Test 2: Kamera-Funktion

Abbildung 5.3.29.: Aufgenommenes Bild der Eye-Tracking-Kamera

Testbezeichnung	Bildübertragung an Pupil Capture
Anforderung	Die Kamerabilder sollen an Pupil Capture gesendet werden.
Zustand	In Pupil Capture kann die USB-Kamera als Quelle ausgewählt und die Bilder somit betrachtet werden. Abbildung 5.3.30 zeigt links das Kamerabild in Pupil Capture und rechts die USB-Kamera als Auswahlmöglichkeit für die Videoquelle bei Pupil Capture.

Tabelle 5.15.: Prototyp Version 3, Test 3: Bildübertragung an Pupil Capture

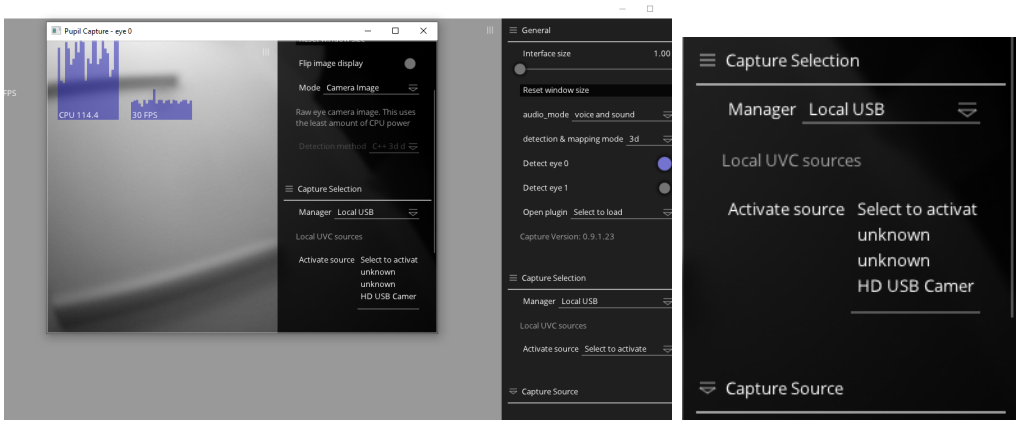

Abbildung 5.3.30.: Kamerabild in Pupil Capture (links) und USB-Kamera als Videoquelle in Pupil Capture (rechts)

Testbezeichnung	Kamera-Ausrichtung
Anforderung	Die Kamera soll auf das Auge ausgerichtet sein, sodass das Auge in Pupil Capture erkannt wird.
Zustand	Bei aufgesetztem HMD ist das Auge auf den Kamerabildern gut zu erkennen und relativ zentral im Bild. In Abbildung 5.3.31 ist die Betrachtung des Kamerabilds in Pupil Capture dargestellt.

Tabelle 5.16.: Prototyp Version 3, Test 4: Kamera-Ausrichtung

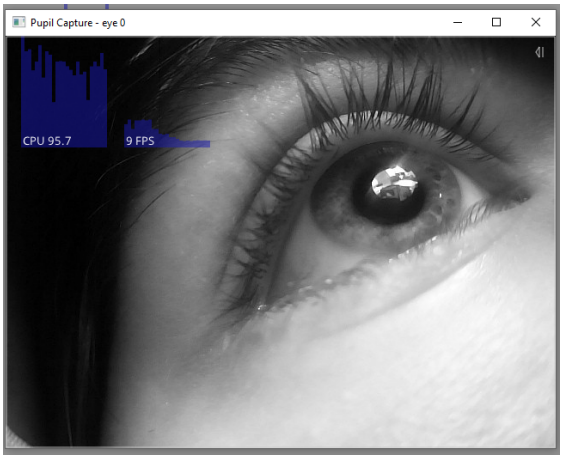

Abbildung 5.3.31.: Ein Bild des erkannten Auges in Pupil Capture

Testbezeichnung	Smartphone-Halterung
Anforderung	Das Smartphone (Nexus 5) soll sicher in die dafür vorgesehene Halterung passen.
Zustand	Das Smartphone passt in die Halterung, verrutscht dort nicht und hat somit die perfekte Position für den Anwender. Die Position des Smartphones und die Halterung ist in der Abbildung 5.3.32 zu sehen.

Tabelle 5.17.: Prototyp Version 3, Test 5: Smartphone-Halterung

Abbildung 5.3.32.: Position des Smartphones auf der Halterung am Cardboard

Testbezeichnung	Eye-Tracking
Anforderung	Die Kamera soll so auf das Auge ausgerichtet sein, sodass dieses in Pupil Capture erkannt wird.
Zustand	Die Pupille des Auges erhält einen roten Kreis in der World View von Pupil Capture. Dies bedeutet, dass Pupil Capture die Pupille erkannt hat und auf Basis dessen eine Kalibrierung gestartet werden kann. In der Abbildung 5.3.33 ist die erkannte Pupille im Kamerabild abgebildet.

Tabelle 5.18.: Prototyp Version 3, Test 6: Eye-Tracking

Testbezeichnung	LED-Ausrichtung
Anforderung	Die jeweils sechs IR-LEDs pro Auge sollen eine gute Ausleuchtung des Auges gewährleisten.
Zustand	Die Position der IR-LEDs ist so ausgerichtet, dass diese das komplette Auge ausleuchten und somit Pupil Capture die Erkennung der Pupille erleichtern. Die Abbildung 5.3.34 zeigt ein durch die IR-LEDs gut ausgeleuchtetes Auge.

Tabelle 5.19.: Prototyp Version 3, Test 7: LED-Ausrichtung

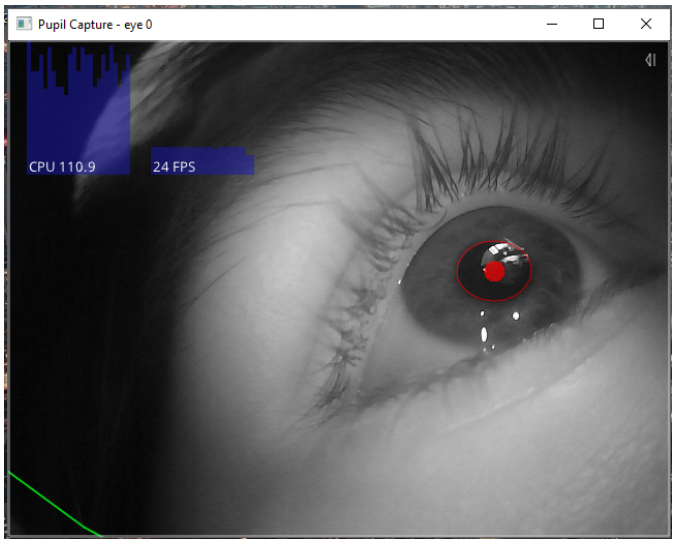

Abbildung 5.3.33.: Erkennung der Pupille in Pupil Capture

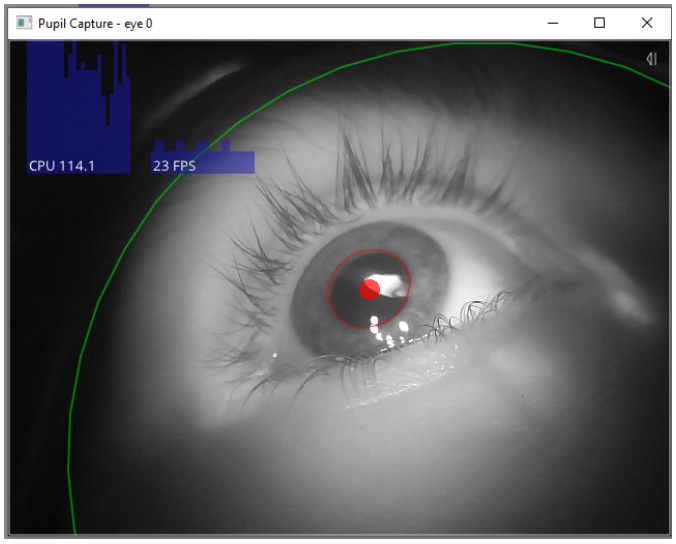

Abbildung 5.3.34.: Kamerabild von einem gut ausgeleuchteten Auge

5.3.4. Vierter Prototyp - Verbesserung der Reproduzier- und Modifizierbarkeit

Die Entwicklung der vierten Version des Prototyps ist sowohl im dritten als auch im vierten Quartal durchgeführt worden. Der Fokus lag hierbei auf der einfachen Reproduzierbarkeit des modifizierten Cardboards.

Aufbau

Am Aufbau direkt hat sich zur dritten Version nichts verändert, somit kann der Aufbau sowie die Beschreibung dazu der Abbildung 5.3.20 aus dem Kapitel des dritten Prototype 5.3.3 entnommen werden.
Da der Fokus eher auf der einfachen Reproduzierbarkeit lag, wurden dementsprechend die Vorlagen für das Cardboard und der Schaltplan angepasst.
Bei der Lasercutter-Vorlage wurden die Teile vereinfacht, die durch erhaltenes Feedback als „schwer“ nachbaubar galten. Da der Magnetknopf nicht mehr benutzt wird, wurde dieser Teil in der Vorlage angepasst. Des Weiteren wurden Gravierungen auf der Vorlage hinzugefügt um die Positionierung zu vereinfachen.
Zudem ist die Kamerahalterung angepasst wurden. Diese besteht immer noch aus einem Modul für beide Kameras (vgl. Abbildung 5.3.35).
Außerdem wurde das Cardboard mit Polsterungen an der Stirn sowie an der Nase ausgestattet (vgl. Abbildung 5.3.35). Bei den Bändern der Kopfhalterung wurde auf breitere Bänder zurückgegriffen. Zudem wurde der Hub mit in die Vorlage des Cardboards integriert, sodass nicht mehr so viele Kabel aus dem Cardboard rausgucken.

Kamera-Position

Die Kamera-Position wurde ein wenig verändert. Dafür wurde die Pappe direkt unterhalb der Linse entfernt und die Kamera dort platziert. Wie bereits im Abschnitt 5.3.4 erklärt, wurde die Halterung der Kamera modifiziert. Es wurden hierbei zwei Seiten entfernt und eine Lasche hinzugefügt, dadurch ist die Halterung einfacher zu falten. Zudem wurde Ausbuchtungen für die Kabel der Kamera hinzugefügt. Des Weiteren sind an der Cardboard-Vorlage Gravierungen hinzugefügt worden um den Nutzer eine einfache Handhabung zu gewährleisten.
Die beiden Bilder 5.3.36 und 5.3.37 zeigen die beiden kritischen Blickrichtungen. Durch den roten Kreis ist zu erkennen, dass diese nun gut erkannt werden.

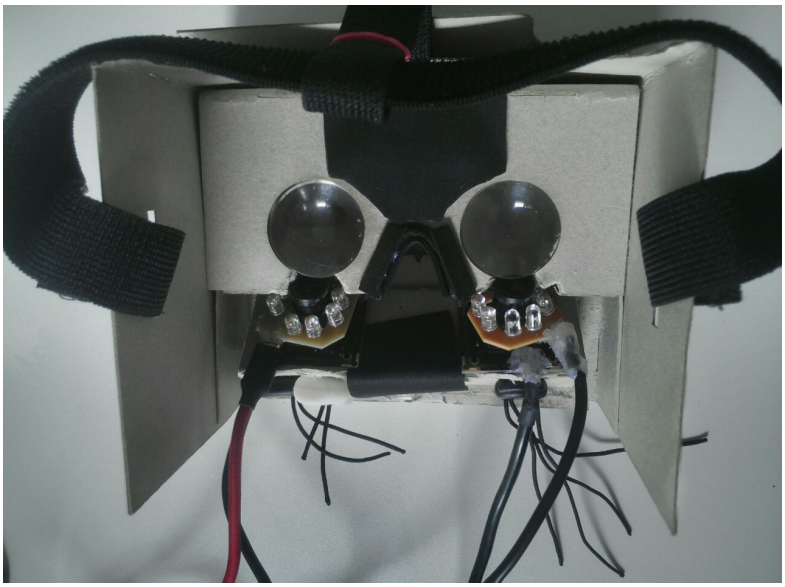

Abbildung 5.3.35.: Polsterung innerhalb des Cardboards

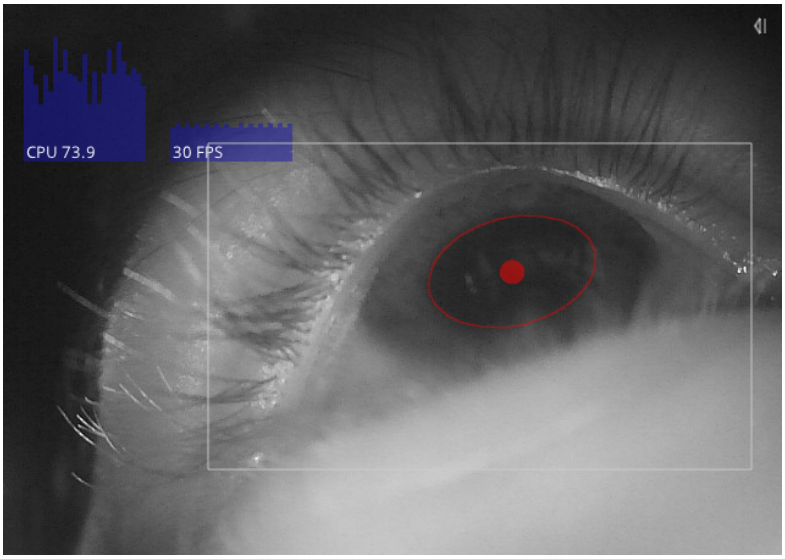

Abbildung 5.3.36.: Blickrichtung oben in der Mitte

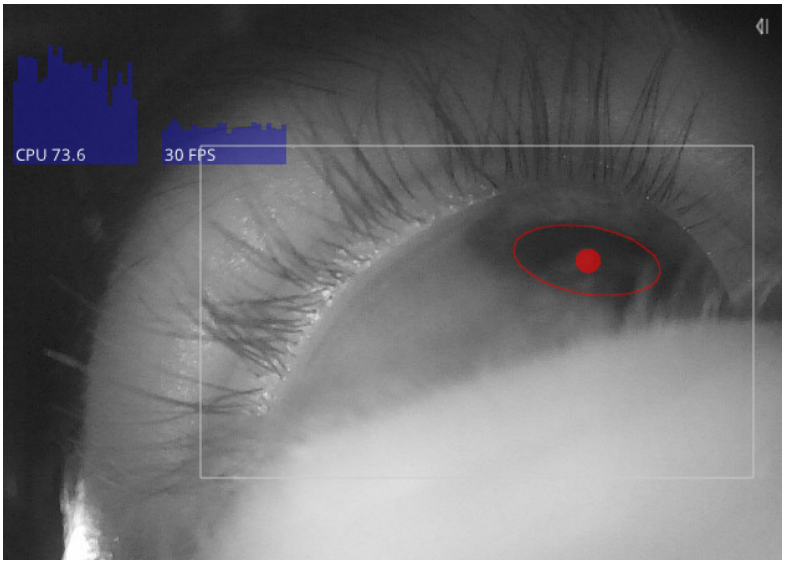

Abbildung 5.3.37.: Blickrichtung oben links

Übertragung des Kamerabilds

Die Übertragung des Kamerabilds hat sich zur dritten Version des Prototypen nicht geändert: Die Eye-Tracking-Kameras sind an einem USB-Hub angeschlossen. Der USB-Hub ist wiederum mit einem Rechner verbunden, auf den die Eye-Tracking Software Pupil Capture läuft.

Tests

Auch bei der vierten Version des Prototypen werden ebenfalls, zur Sicherstellung der grundlegenden Funktionalitäten, Hardware-Tests durchgeführt. Dabei wurde sich an die Gliederung der sechs Hardwaretests des vorherigen Prototyps orientiert (vgl. Kapitel 5.3.3). Auch hier wurde mit sechs Infrarot-LEDs getestet.

6. Software

In diesem Kapitel wird das Software-Framework näher beschrieben, dass im Rahmen der Projektgruppe entwickelt wurde. Dazu werden zunächst die verschiedenen Aspekte unserer Zielsetzung vorgestellt und erläutert wie sie im Verlauf des Projektes umgesetzt wurden.
Der nächste Abschnitt geht auf die Konzeptionierung des Frameworks ein, hier wird zuerst die Architektur vorgestellt. Anschließend wird erklärt warum wir uns für die verwendete externe Software entschieden haben. Zum Abschluss des Abschnittes wird die Entwicklung des Konzeptes und die von uns getroffenen Designentscheidungen aufgezeigt.
Anschließend wird auf die eigentliche Implementierung eingegangen und die einzelnen Bestandteile des Frameworks vorgestellt. Zum Ende des Kapitels folgt eine Vorstellung unserer Tests.

6.1. Zielsetzung

Aus den vorgegebenen und durch Nutzerinterviews erhobenen Anforderungen leiten sich auch für das Software-Framework wichtige Zielsetzungen ab. Im Zentrum steht ein Nutzer, der schnell und einfach, mit geringen Kosten, seine Ideen umsetzen kann. Diese nutzerzentrierte Sichtweise spiegelt sich auch in dem Entwurf des Software-Frameworks wider. Drei zentrale Zielsetzungen für dieses Framework sollen nun im Folgenden genauer erörtert werden, da sie eine wichtige Rolle im Entwicklungsprozess gespielt haben.

6.1.1. Einfachheit

In unserem Framework wird dem Prinzip der Einfachheit eine hohe Relevanz beigemessen. Diese Einfachheit spiegelt sich in für den Anwender geringen Lernaufwand wieder. Auch Anwender mit weniger ausgeprägten Programmierkenntnissen sollen in der Lage sein, mithilfe des Frameworks schnell und einfach AR- und VR-Anwendungen mit Eye-Tracking zu kombinieren. Die Einfachheit unseres Frameworks wird auf unterschiedliche Art und Weise sichergestellt. Die für eine VR- bzw. AR-Anwendung relevanten Bausteine sind bereits als gebündelte Prefabs vorhanden, welche intuitiv per Drag-and-Drop in eine Szene eingefügt werden können.

Die Prefabs erleichtern dabei das Erstellen von AR- und VR-Anwendungen soweit, dass sich nur noch um die virtuellen Objekte der Szene und die Implementierung von eigenen Interaktionstechniken gekümmert werden muss, da alle, für die Funktionsweise des Frameworks relevanten Services, durch die Prefabs zur Verfügung stehen. Das Prinzip der Einfachheit spiegelt sich ebenfalls bei der Implementierung von eigenen Interaktionstechniken wider. Da die jeweiligen Interaktionstechniken über die Implementierung von vorgefertigten Interfaces realisiert werden, bekommt der Anwender die Möglichkeit schon durch das Schreiben von wenigen Zeilen Code die ersten eigenen Interaktionstechniken zu erstellen.

6.1.2. Bedienbarkeit

Als wichtiger Kernaspekt unseres Frameworks wurde in hohen Maße auf die Bedienbarkeit wert gelegt. Bedienbarkeit unterteilt sich in dabei drei Teilaspekte: Bequemlichkeit, Geschwindigkeit und Fehlervermeidung. (vgl. [Pre11], S.3)
Prefabs für unterschiedliche Anwendungszwecke und Interfaces, um eigene Interaktionstechniken zu implementieren, fördern dabei alle drei dieser Aspekte. Es steigert die Geschwindigkeit, da sich so der Nutzer intuitiv einzelnen Modulen zu einer Applikation zusammensetzen kann, ohne sich lange mit der Materie beschäftigt zu haben. Anwendungsfehler werden so auch minimiert und dadurch erlauben die Interfaces dem Nutzer schnell und effektiv an das gewünschte Ziel zu kommen. Auch die benötigte Hardware wurde auf Basis dieser Aspekte konzipiert. Vorlagen für das von uns modifizierte Cardboard, als auch unserer LED-Platinen und eine Einkaufsliste werden für den Benutzer bereit gestellt. Unterstützt wird all dies durch Anleitungen, die den Nutzer optimal auf die Entwicklung seiner eigenen Interaktionstechnik als auch des benötigten HMD vorbereiten und ihn ohne große Mühe das Wissen vermitteln, mit unserem Framework erfolgreich arbeiten zu können.

6.1.3. Erweiterbarkeit

Im Software-Kontext heißt Erweiterbarkeit, dass eine Software mit geringem Aufwand an Änderungen der Anforderungen angepasst werden kann. Um diese Erweiterbarkeit zu unterstützen sollte eine Software modular aufgebaut sein, so dass einzelne Bestandteile ausgetauscht werden können, ohne dass der Rest ebenfalls ausgetauscht oder neu entwickelt werden muss.
Da das Framework unter einer Open-Source-Lizenz veröffentlicht wird, ist es sehr gut möglich, dass Nutzer unseres Frameworks andere Anforderungen an die Software haben, als zum Anfang des Projektes erhoben wurden. Demnach ist eine Zielsetzung der Entwicklung des Frameworks, dass nach Abschluss auch durch externe Programmierer das Framework schnell erweitert werden kann.
Eine Möglichkeit das Framework zu erweitern, ist das Hinzufügen neuer Arten

von Interaktionsmöglichkeiten, da sich bei der Entwicklung für eine begrenzte Anzahl von Techniken entschieden wurde. Diese Erweiterbarkeit ist dadurch gegeben, dass wir unsere Services modular entworfen haben. Durch diesen Aufbau, auf den näher in Kapitel 6.2.1 eingegangen wird, ist es einfach möglich das Framework um neue Services zu erweitern. Ein anderer begünstigender Faktor für die einfache Erweiterbarkeit ist die Überschaubarkeit des Frameworks. Diese ergibt sich die geringe Anzahl an Skripten, die für die Funktionsweise der Services benötigt werden. Weiterhin ist es für die Erweiterbarkeit förderlich, dass wir die Unity Engine mit C# als Skriptsprache verwenden. Die Unity Engine wird bereits in der Forschung zur Entwicklung von zum Beispiel AR-Anwendungen verwendet, da gängige AR-Frameworks, wie zum Beispiel Vuforia, Unity als Grundlage benutzen. Des Weiteren ist C# eine weit verbreitete Programmiersprache, mit sehr großer Ähnlichkeit zu Java. Durch diese Verbreitung bzw. Ähnlichkeit ist es wahrscheinlich, dass ein potentieller Anwender des Frameworks bereits mit der Sprache vertraut ist oder sich schnell in diese einfindet.

6.2. Konzept

In diesem Kapitel wird das Konzept des Frameworks erläutert. Dafür wird zunächst die Architektur beschrieben, bevor auf die verwendete, externe Software und die eigentliche Entwicklung des Konzeptes eingegangen wird.

6.2.1. Architektur

Die Architektur besteht hauptsächlich aus den Services und die dazugehörigen Interfaces. Die drei vorimplementierten Services haben die Aufgaben, zu erkennen, wann eine bestimmte Interaktionstechnik ausgelöst werden soll und leisten somit die Hauptarbeit des Frameworks.

Kommunikation mit der Eye-Tracking-Software

Damit die Services diese Arbeit leisten können, gibt es den *PupilListener*, der die Verbindung zur externen Eye-Tracking-Software Pupil Capture herstellt und so Informationen für die Services zur Verfügung stellt. Der *PupilListener* stellt neben den Informationen, die von den Services benötigt werden, auch alle anderen Informationen, die von Pupil Capture gesendet werden, zur Verfügung. Die Bereitstellung vieler Informationen bestätigt die Anforderung der Erweiterbarkeit (vgl. 6.1.3). Diese Kommunikation durch den Datenfluss von Pupil Capture zum *PupilListener* in Abbildung 6.2.1 dargestellt. Näheres zu der Kommunikation zu Pupil Capture kann dem Kapitel 6.3.5 entnommen werden.

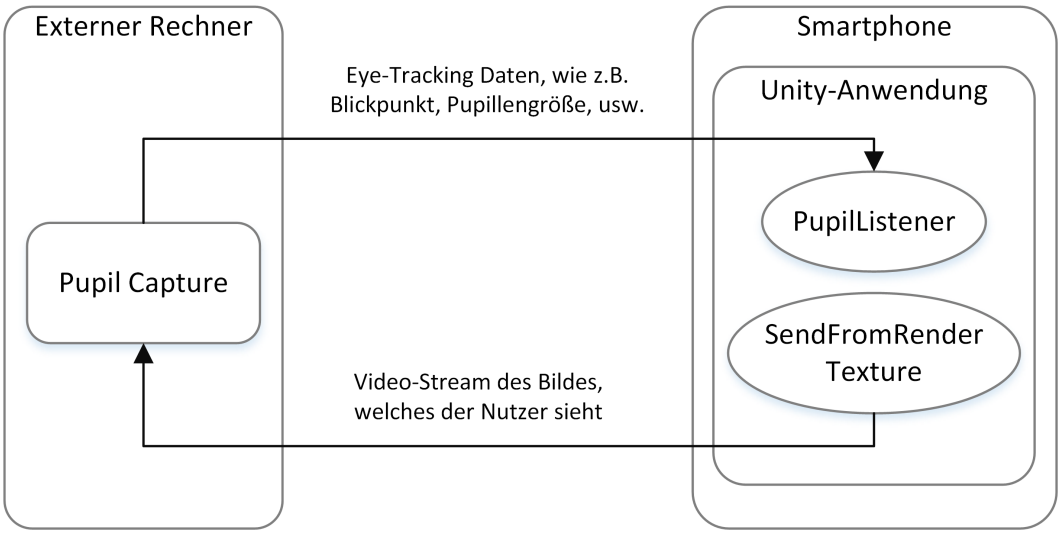

Abbildung 6.2.1.: Die Kommunikation der Eye-Tracking-Software mit dem Framework

Die Services des Frameworks

Die Services wurden so entworfen, dass jeder Service eine Art von Interaktionen erkennen kann. Es gibt jeweils einen Service für Fixationen, Gesten und das Verfolgen von Objekten. Der *FixationDetectionService* ist zuständig für die Erkennung von Fixationen, der *PursuitDetectionService* erkennt Verfolgungen. Beide Services arbeiten auf den aktuellen Blickdaten, die diese sich über den *PupilListener* holen. Der *GestureDetectionService* ist zuständig für die Erkennung von Gesten aus Fixationen und Sakkaden. Der Service arbeitet nicht direkt auf den Blickdaten, sondern nutzt die schon vorhandene Fixationserkennung des *FixationDetectionService*.
Zu jedem der Services wurde ein Interface erstellt, über das Methoden implementiert werden, mit denen der Service mit einer Interaktionstechnik kommunizieren kann. Dadurch können Interaktionstechniken einfach erstellt werden, da sie nur das jeweilige Interface implementieren müssen. Das Interface für den *FixationDetectionService* liefert Methoden für den Start und das Ende einer Fixation. Zusätzlich bietet es noch eine Methode, die stetig aufgerufen wird, solange die Fixation andauert. Das Interface des *GestureDetectionService* hingegen bietet nur eine Methode, wenn die Geste vervollständigt wurde. Das Interface des *PursuitDetectionService* bietet eine Methode, die stetig aufgerufen wird und über den Parameter Informationen zum momentanen Status der Verfolgung liefert. Des Weiteren wird noch eine Methode definiert, die aufgerufen wird, sobald die anfangs definierten Bedingungen der Verfolgung eintreffen. Diese Bedingungen bestehen aus einer Zeit, die

die Verfolgung mindestens bestehen muss und einer unteren Schranke der Korrelation zwischen der Position des Objektes und den Blickdaten des Nutzers.
Außerdem müssen sich Instanzen der Interfaces beim jeweiligen Service anmelden, damit die Interaktionstechnik ausgelöst werden kann. Bei dieser Anmeldung können zusätzlich noch Werte für die Toleranz angegeben werden. Auf die genauen Parameter und deren Bedeutung wird im Abschnitt 6.3.2 eingegangen.
Eine einfache, schematische Darstellung des Aufbaus der Services ist in Abbildung 6.2.2 zu sehen. Hierbei ist in der Darstellung zu sehen, dass der *GestureDetectionService* das Interface des *FixationDetectionServices* nutzt, um sich über neue Fixationen zu informieren.

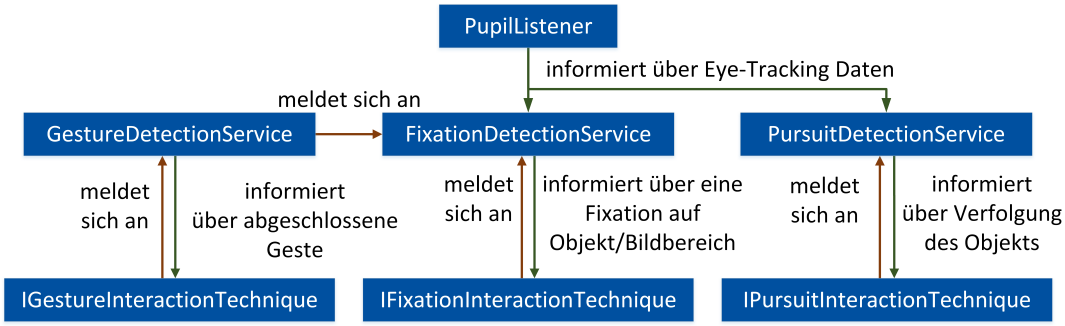

Abbildung 6.2.2.: Der grundlegende Aufbau der Services

Video-Streaming

Neben den Services läuft noch ein weiteres Skript, welches dafür zuständig ist das Bild, welches der Nutzer sieht, an die Eye-Tracking-Software zu senden. Dies geschieht allerdings nicht in der doppelten Sicht für das stereoskopische Sehen, wie es auf dem Smartphone im HMD dargestellt wird, sondern als einzelnes Bild, so dass dieses Bild auch zur Beobachtung bzw. Kontrolle verwendet werden kann. Der Video-Stream ist durch den Datenfluss vom *SendFromRenderTexture* zu Pupil Capture in Abbildung 6.2.1 dargestellt. Nähere Informationen können dem nachfolgenden Kapitel 6.3.4 entnommen werden.

6.2.2. Verwendete Software

In diesem Abschnitt wird erläutert warum sich für die jeweils genutzten externen Software entschieden wurde.

Pupil Capture

Bei der Auswahl der Eye-Tracking-Software musste eine geeignete Open-Source-Lösung gefunden werden. Dies war von Bedeutung damit auch die Eye-Tracking-Software im Sinne der übergeordneten Projektanforderungen auch nach Projektabschluss durch die Community bei Bedarf erweitert werden kann. Mit Pupil Capture konnte diese Anforderung erfüllt werden. Da Pupil Capture die Möglichkeit bietet Plug-Ins einzubinden, konnte das Programm um eine Schnittstelle zu Unity erweitert werden. Mithilfe unseres Plug-Ins wurde ermöglicht, die VR- oder AR-Anwendung vom Smartphone in Echtzeit als Video-Stream zu empfangen und in der World View von Pupil Capture darzustellen. Dies war ebenfalls ein wichtiges Kriterium, da wie bereits zuvor erwähnt, eine wichtige Anforderung an das Projekt war, dass dem Anwender die Möglichkeit gegeben wird in Echtzeit zu sehen, was der Proband sieht um ggf. kontrollierend einzugreifen oder in Echtzeit eventuell notwendige Handlungen besser dirigieren zu können.

Unity

Es wurde sich für eine Entwicklung mithilfe der Unity-Engine aus vielerlei Gründen entschieden. Die Engine ist weit verbreitet und bietet vielfältige Integrationsmöglichkeiten. Da bereits das Google Cardboard verwendet wurde, war es naheliegend, dass die verwendete Engine auch das zugehörige Google VR SDK unterstützt. Auch für Implementierung von AR-Anwendungen gab es Möglichkeiten, allen voran kann mit wenig Aufwand eine der weit verbreitetsten AR Tool-Sets, Vuforia, problemlos eingebunden werden. Wie bereits im vorherigen Abschnitt beschrieben konnte Unity als Input in Pupil Capture eingebunden werden, was in Verbindung mit der Tatsache, dass im Entwicklerteam bereits Unity-Expertise vorhanden war, die Entscheidung bzgl. der Unity-Engine zusätzlich begünstigt hat. Zudem unterstützt Unity inzwischen bis zu 25 Zielplattformen [Uni]. Auch wenn das Projekt als Android-Applikation konzipiert ist, lässt die mit der Unity-Engine gegebene Portabilität auch eine Integration in andere gängige VR- und AR-Systeme zu (z.B. Playstation VR, HTC Vive, Occulus Rift, Microsoft HoloLens, usw.).

6.2.3. Entwicklung des Konzeptes

In den vorherigen Abschnitten wurde bereits sowohl die verwendete Software, als auch das erstellte Framework beschrieben. Ziel dieses Kapitels ist es die Entwicklung dieser beiden Aspekte im Verlauf der Projektgruppe zu beschreiben.

Entscheidungsfindungen in der Anfangsphase

Als zentraler Baustein des Projektes kristallisierte sich bereits zu Anfang die Unity Engine (vgl. 6.2.2) als Entwicklungsumgebung heraus. Aufgrund des kostenlosen Zugangs und einer umfangreichen Dokumentation bietet sie einen guten Einstiegspunkt in die Entwicklung von virtuellen Welten. Zudem lässt sich das Google VR SDK für Unity mit der Engine kombinieren, was das Erstellen von VR-Anwendungen für das Android Smartphone deutlich erleichtert. Für den Einsatz im Bereich der erweiterten Realität existieren ebenfalls Toolkits, die den Nutzer bei der Erstellung von Anwendungen behilflich sind. Die Entscheidung fiel hierbei zunächst auf das AR-Toolkit (vgl. 2.3.4). Es ermöglicht eine einfache Erstellung von AR-Anwendungen und lässt sich mit der Unity Engine kombinieren. Als zusätzlicher Pluspunkt kommt die Lizenzierung als Open-Source-Software hinzu, was der Grundidee des Projektes zugutekommt und somit viel die Wahl auf dieses Toolkit.
Bei der Auswahl der Eye-Tracking-Software fiel die Entscheidung auf Pupil Capture (vgl. 2.4.3) von Pupil Labs. Auch diese Software ist frei zugänglich und trägt so zum Ziel einer möglichst kostengünstigen Lösung bei. Des Weiteren waren die neusten Versionen der Software sehr jung und dies ließ uns, zusammen mit einem gepflegten Github-Auftritt drauf schließen, dass die Software noch aktiv entwickelt und verbessert wird. Neben Pupil Capture bietet Pupil Labs auch noch andere Eye-Tracking-Software an. Die Wahl fiel letztendlich auf Pupil Capture, da es über ein User Interface verfügt und so dem Anwender erlaubt sowohl das erkannte Auge der Person, als auch einen Einblick darüber zu erlangen, wo die Person hinsieht und welchen Bereich sie fokussiert.

Die Entwicklung des Frameworks

Durch die Wahl der Unity Engine als zentrale Schnittstelle zwischen den anderen Softwareelementen, lag die Entscheidung nah, unser Framework für die Interaktion zwischen Augendaten und virtuellen Objekten ebenfalls in der Unity Engine anzusetzen. Die Engine bietet die Möglichkeit zusätzliche Funktionalitäten in Form von Skripten zu integrieren. Dabei kann zwischen C# und JavaScript als Programmiersprachen gewählt werden. Die Entscheidung fiel dabei auf C#, da die generelle Expertise in Java unter den Projektmitgliedern höher war und C# einige Ähnlichkeiten mit Java aufweist [mic]. Zudem war anzunehmen, dass das zu erstellende Framework ein größeres, mehrere Skripte umfassendes Projekt werden würde, worin wir einen weiteren Vorteil von C# gegenüber der Skriptsprache JavaScript sahen.
Die Entwicklung des Frameworks begann mit der Planung und dem Entwurf einer Architektur. Diese Architektur sollte ermöglichen, dass ein Anwender schnell und

ohne viel Aufwand eigene Interaktionstechniken implementieren kann. Zudem sollte das Framework einfach erweiterbar sein. Für die Erweiterbarkeit drängte sich zwangsläufig die Idee auf, Daten über das Eye-Tracking an einer zentralen Stelle zur Verfügung zu stellen. Aus diesem Gedanken entstand ein zentrales Skript des Frameworks, der *PupilListener*. Hier laufen die Daten des Eye-Trackings zusammen und können für die Weiterverarbeitung durch andere Skripte bereitgestellt werden. Doch wie könnten diese weiteren Schritte aussehen? Um diese Frage zu beantworten sammelten wir mögliche Anwendungsszenarien und Arten von Interaktionstechniken. Dabei orientierten wir uns auch an den Arbeiten von Hans Gellersen. Demnach lassen sich Interaktionen in unterschiedliche Kategorien einteilen, wie zum Beispiel Interaktion durch den Blick als Zeiger, oder durch Verfolgung eines Objektes [**?**]. Diese Einteilung legte nahe, zentrale Stellen einzurichten, die für je eine Gruppe an Interaktionstechniken zuständig wären. So entstand der in vorherigen Kapiteln beschriebene Entwurf der Services und die dazugehörigen Interfaces (vgl. 6.2.1). Als ersten Service entwickelten wir den *FixationDetectionService*, um damit den großen Bereich der fixationsbasierten Interaktionstechniken abzudecken. Aufbauend darauf folgte der Service zum Erkennen von Gesten und abschließend der zum Erkennen von Verfolgungen. Eine Integration weiterer Services und damit Arten von Interaktionstechniken entfiel, aufgrund der beschränkten Dauer der Projektgruppe. Dieses ist aber aufgrund des modularen Aufbaus jederzeit denkbar. Um dem Nutzer Arbeit beim Einrichten des Frameworks abzunehmen und sicher zu stellen, dass die Funktionalität nicht durch mehrfaches Ausführen der Services gefährdet ist, wurde schließlich ein zentrales Skript erstellt, welches die Erstellung und Verwaltung der anderen Skripte überwacht, der *ServiceProvider*. Über diesen können Instanzen der jeweiligen Services erlangt werden und zudem bietet der *ServiceProvider* eine zentrale Stelle für die Einstellung weiterer Optionen.
Um genaue Eye-Tracking-Daten ermitteln zu können, wurde das Framework um eine Kalibrierungsfunktion erweitert. Durch ein zusätzliches Skript ist es möglich Marker in einer Szene einzublenden, die von der Eye-Tracking-Software erkannt werden können. Durch den Einsatz einer Fernbedienung ist es möglich, die Marker auf Wunsch ein- und auszublenden. Damit ersetzt diese Methode die Kalibrierung durch einen reellen Marker und bietet so auch in der virtuellen Realität eine zuverlässige Variante der Kalibrierung.

Die Erweiterung der Eye-Tracking-Software

Durch die durchgeführten Interviews mit für die Zielgruppe repräsentativen Versuchspersonen erreichte uns schon früh der Wunsch, als außenstehende Person eine Möglichkeit zu haben, ein Bild dessen zu erhalten, was ein Träger des Head-Mounted Displays wahrnimmt. Die verwendet Eye-Tracking Software Pupil Capture lässt eine Darstellung solcher Inhalte zwar bereits zu, doch gab es keine

Möglichkeit die Videodaten, die auf dem Smartphone dargestellt wurden zu Pupil Capture zu übertragen. So entwickelten wir die erste Version eines Skriptes, um den von der Unity Engine generierten Frame abzugreifen und auf den PC via Netzwerk zu übertragen. Auf der Seite von Pupil Capture wurde ein weiteres Backend zum Empfangen des Streams implementiert. So konnte das Bild zwar übertragen werden, doch stellte sich hierdurch heraus, dass der hohe Rechenaufwand nach jedem Frame einen deutlichen Abfall der Framerate in der Unity Engine bedeutete. Im Laufe der Projektgruppe wurde deshalb die Auflösung des Streams reduziert und das Framework um eine Möglichkeit erweitert den Stream nur in Graustufen darzustellen. So konnte eine Reduktion des Rechenaufwands und somit eine Erhöhung der Framerate realisiert werden. Diese Lösung funktionierte zwar besser, doch gab es immer noch eine große Hürde bei der Installation. Eine direkte Änderung des Sourcecodes der Pupil Capture Software erforderte es vom Nutzer eine Version von Pupil Capture selbst zu kompilieren und alle Abhängigkeiten selbst einzurichten. Um dem Nutzer diese Hürde zu nehmen, änderten wir unser Backend zu einem leichter zu integrierenden Plugin. Die genaue Funktionsweise wird in einem späteren Kapitel beschrieben (vgl.6.3.4).

Die Kombination des Frameworks mit Augmented Reality

Wie bereits beschrieben, resultierte aus der Idee, dem Nutzer durch ein AR-Framework Arbeit beim Erstellen von AR-Programmen abzunehmen, die Wahl auf AR-Toolkit als zusätzlicher Bestandteil unserer Lösung. Bei der Entwicklung des Frameworks stellte sich allerdings heraus, dass sich die Arbeitsweise des AR-Toolkits sehr von der erwarteten und somit auch von der des Google VR SDKs unterschied. Dies erschwerte die Kombination mit unserem Framework. Hinzu kam, dass große Teile der Logik des AR-Toolkits in externe Bibliotheken ausgelagert sind, was eine Erweiterung erschwerte. So war es notwendig eine andere Lösung für die Realisierung von AR zu finden. Doch gab es keine Open-Source-Lösung, die uns vom Funktionsumfang und der Aktualität von ihrer Nutzung überzeugte. So entwarfen wir den Plan für eine eigene Umsetzung einer AR-Einbindung. Diese sollte grundlegende Funktionen der erweiterten Realität bieten und gleichzeitig das Framework bei Wunsch durch eine umfangreichere Lösung erweiterbar sein. Dabei viel unsere Wahl als externe Software auf das bereits beschriebene Vuforia (vgl. 2.3.3). Dieses ließ sich problemlos in das Framework integrieren und bietet so eine Möglichkeit das Framework für kompliziertere AR-Anwendungen auszubauen.

6.3. Implementierung

Der nachfolgende Abschnitt legt das Vorgehen bei der Softwareentwicklung dar. Es wird beschrieben, wie das übergeordnete Konzept und die Architektur erarbeitet wurde und daraufhin die weitere Implementierung stattfand. Weiterhin wird darauf eingegangen, welche Funktionen geboten werden und wie die Schnittstellen zur Fremdsoftware gestaltet worden sind.

6.3.1. Vorgehen

Unser Vorgehen beim Entwickeln des Framework begann mit einer gemeinsamen Konzeptionierungsphase während unserer Powerwoche (vgl. 3.5.2). Das Konzept wurde dort anhand der erhobenen Anforderungen und einer Auswahl an Interaktionstechniken, die implementiert werden sollten, erstellt.
Die ausgewählten Techniken wurden genutzt, indem wir mittels des entworfenen Konzeptes die Kommunikation der Komponenten durchgegangen sind. Als dann das Konzept und die grundlegende Architektur des Frameworks erarbeitet war, wurde mit der Entwicklung des ersten Bestandteils, dem *FixationDetectionService*, begonnen. Die Implementierung erfolgte mittels „Pair Programming“, bei dem eine Person den Code schreibt und eine weitere achtet auf Probleme im Code und denkt weiter über die Problemstellungen nach. Dadurch wird eine bessere Code Qualität erreicht und Probleme oft schneller gelöst, da sie nicht von einer einzelnen Person, sondern im Dialog angegangen werden. Auch die weiteren Komponenten des Frameworks wurden mittels „Pair Programming“ entwickelt.
Nach Fertigstellung der Implementierung folgte in unserem Arbeitsablauf das Feedback durch ein anderes Gruppenmitglied, das nicht an der Aufgabe gearbeitet hat. Dabei wurden noch oft Probleme gefunden, die dann auch behoben werden konnten.
Nachdem die Services des Frameworks implementiert waren, wurden diese im Laufe des Projektes noch um weitere Features erweitert. Weiterhin wurden aufgrund aufgetretener Verwendungsprobleme weitere Änderungen vorgenommen. Auch diese wurden im Rahmen unseres Arbeitsablaufs behoben und überprüft.

6.3.2. Services

Aus der Architektur ist zu erkennen, dass die unterschiedlichen Services die zentralen Bausteine des Frameworks sind (vgl. 6.2.2). Sie stehen je für eine Familie von Interaktionstechniken und es ist ihr Ziel Interaktionsversuche des Nutzers mit virtuellen Objekten zu erkennen. Dabei ähneln sie sich in ihrem Aufbau. Jeder Service ist selbst eine Implementation des von der Unity Engine bereitgestellten *MonoBehavior* Interfaces. Dieses Interface ermöglicht es dem Service verhalten zu

definieren, dass bei seiner Erstellung oder auch zu jedem laufenden Frame ausgeführt wird.
Jeder Service verfügt über eine *Collection* in der es Interaktionstechnik repräsentierenden Objekten speichern kann. Diese Objekte enthalten sowohl einen Verweis auf ein Skript, welches eine Interaktionstechnik beschreibt, zum anderen weitere Informationen, die die Interaktionstechniken genauer beschreiben. Nur Interaktionstechniken, die sich in dieser Liste befinden, können ausgeführt werden, denn nur für diese Interaktionstechniken sucht der Service aktiv nach dem spezifizierten Auslöser. Die Services arbeiten also nur, wenn sich bei ihnen überhaupt Skripte angemeldet haben. Ansonsten sind sie im Leerlauf. Damit sich Interaktionstechniken anmelden können, bietet der Service eine *Subscribe*-Methode. Diese Methode kann aus dem Skript, welches die Interaktionstechnik spezifiziert, aufgerufen werden. Dabei übergibt das Skript eine Referenz an sich selber und wird durch den Service in die besagte *Collection* eingetragen. Zusätzlich können weitere optionale Parameter angegeben werden, um die Auslösebedingungen der Interaktionstechnik genauer zu spezifizieren. Der Service erstellt dann ein gekapseltes Objekt aus den Optionen und der Interaktionstechnik an sich und speichert es in seiner *Collection*. Zusätzlich bietet jeder Service eine *Unsubscribe*-Methode, mit der sich eine Interaktionstechnik aus der *Collection* austragen und damit nicht mehr ausgeführt werden kann.
Jeder Service holt sich in der *FixedUpdate*-Methode der Unity Engine, also zu jedem Frame, den momentanen Blickpunkt des Nutzers, der durch die Augenerkennung ermittelt wurde. Diesen benutzen die Services, um unter Einbezug von älteren Blickpunkt Daten zu ermitteln, ob eine Interaktion ausgelöst werden soll. Ist dies der Fall wird die entsprechende Methode der Interaktionstechnik ausgeführt. Der *FixationDetectionService* bezieht alte Blickpunkt Daten ein und berechnet die Streuung der Koordinaten. Die akzeptierte maximale Streuung wird dabei durch jede Interaktionstechnik spezifiziert. Der *GestureDetectionService* wird bei Fixationen aktiviert. Demzufolge ist er ein Beispiel für eine fixationsbasierte Interaktionstechnik. Bei jeder neuen Fixation überprüfte er, ob die momentane Fixation in Kombination mit den bisherigen zu einer der angemeldeten Gesten passt. Der *PursuitDetectionService* benutzt den momentanen Blickpunkt und die Position des zu verfolgenden Objektes. Daraus berechnet dieser die Korrelation über eine zuvor angegebene Zeit. Erreicht die Korrelation nach dieser Zeit eine ebenfalls zuvor definierte Schranke, wird die Interaktion ausgelöst.
Aufgrund des ähnlichen Aufbaus der Services ist eine Erweiterung des Frameworks um zusätzliche Services denkbar. Auf diesem Weg könnten noch weitere Arten von Interaktionstechniken ermöglicht werden.

6.3.3. Schnittstellen

Um der Anforderung, eine schnelle Implementierung von Interaktionstechniken im Sinne des Rapid-Prototyping zu ermöglichen, gerecht zu werden, haben wir uns dafür entschieden, die für eine Interaktionstechnik benötigten Funktionen über die Implementierung eines Interfaces bereitzustellen. Je nachdem welche Art von Interaktionstechnik implementiert werden soll, können drei verschiedene Interfaces

- Fixationen,
- Gesten oder
- Verfolgungen

über eine vorgegebene Schnittstelle implementiert werden.
Um die Einstiegsbarrieren zur Nutzung des Frameworks so gering wie möglich zu gestalten, folgen die implementierten Methoden der *Start*- und *Update*-Logik von Unity. Somit können auch Anwender mit weniger ausgeprägten Programmierkenntnissen in wenigen Zeilen Code eigene Interaktionstechniken für das Eye-Tracking entwerfen.

6.3.4. Video-Streaming zu Pupil Capture

Der Video-Stream zu Pupil Capture wurde sowohl Unity- als auch Pupil Capture-seitig selbst entwickelt. Auf beiden Seiten mussten dafür, ein Skript bzw. ein Plugin geschrieben werden, die die Videoübertragung realisieren. Die folgenden beiden Abschnitte behandeln daher wie in Unity als auch in Pupil Capture die Umsetzung stattfand und welche Probleme dabei auftraten.

Empfänger - Pupil Capture Plugin

Auf dieser Seite wurde erst ein Streaming-Backend auf Basis des „Fake Backends“ von Pupil Capture geschrieben. Dies ließ sich jedoch nur in die nicht kompilierte Version von Pupil Capture einbinden, sodass erweiterte Kenntnisse zur Kompilierung des Programms benötigt wurden. Im späteren Verlauf des Projektes wurde das Backend daher in ein Plugin umgeschrieben, dass sich auch in die jede vorkompilierte Version von Pupil Capture integrieren lässt. Hierdurch änderte sich aber nichts am funktionalen Anteil des Codes aufgrund dessen im weiteren Verlauf nur auf das Plugin eingegangen wird.

Die signifikante Änderung des Codes erfolgte durch die beiden Methoden „def __init__(self, g_pool)“ und „def__init__(self, g_pool)“ in der Klasse „Unity_Stream_Plugin_Source“, da diese die eigentliche Netzwerkkommunikation implementieren.

```
1  class Unity_Stream_Plugin_Source(video_capture.Base_Source)
       :
2
3  def __init__(self, g_pool):
4          super().__init__(g_pool)
5          self.frame_count = 0
6          self.presentation_time = time()
7          self.fps = 30
8          self.lock = Lock()
9          self.img_for_this_frame = None
10         self.thread_running = True
11
12         port = 8051
13
14         # To connect to a different device running a unity
               app we need some connection stuff
15         self.socket = socket.socket(socket.AF_INET, socket.
               SOCK_DGRAM)
16         self.socket.bind  (("", port ))
17         self.socket.settimeout(3)
18         self.network_thread()
```

Initial wird der benötigte Socket mit dem Port, dem Protokoll und den Netzwerk-Thread erstellt.

```
1  def network_thread(self):
2  def run():
3  while True:
4
5          self.lock.acquire()
6          if not self.thread_running:
7                  self.lock.release()
8                  return
9
10         self.lock.release()
11
12         # Lets try to get some data
13         try:
14                 data, adr = self.socket.recvfrom(64000)
15         except socket.timeout:
16                 logger.warning("No connection could be
                       established. ")
17                 self.lock.acquire()
```

```
18                      self.img_for_this_frame = None
19                      self.lock.release()
20                      continue
21
22              # Recreate the frame
23              npArray1d = np.fromstring(data, dtype=np.uint8)
24
25              mode = npArray1d[0]
26              npArray1d = np.delete(npArray1d, [0])
27
28              width = 0
29              height = 0
30
31              if mode == 3:
32                      width = 195
33                      height = 109
34                      npArray3d = np.reshape(npArray1d, (height,
                            width, 3))
35                      npArray3d = cv2.cvtColor(npArray3d, cv2.
                            COLOR_BGR2RGB)
36                      npArray3d = cv2.flip(npArray3d, 0)
37              else:
38                      width = 330
39                      height = 185
40                      npArray3d = np.reshape(npArray1d, (height,
                            width, 1))
41                      npArray3d = cv2.flip(npArray3d, 0)
42                      # frameU = cv2.imdecode(frameU, cv2.
                            IMREAD_COLOR)
43
44              self.lock.acquire()
45              self.height = height
46              self.width = width
47              self.mode = mode
48              self.img_for_this_frame = npArray3d
49              self.lock.release()
50
51      self.network_thread = Thread(target = run, args = ())
52      self.network_thread.start()
```

Im Thread selbst (Code siehe oben) wird dauerhaft auf dem den Port 8051 gehorcht. Sobald ein UDP-Package empfangen wird, wird es in ein *NumPy-Array* umgewandelt, um es in OpenCV als Bild einlesen zu können. Dabei gibt der erste

Bit immer den beim Empfänger gewählten Qualitätmodus an. Nachdem das Bild dekodiert wurde, kann es in der Pupil Capture Graphical User Interface (GUI) angezeigt werden und ggf. das nächste Bild empfangen werden.

Sender - Unity Skripte

Das Unity-Skript sendet in unterschiedlichen Qualitätsmodi (Graustufe und Farbe) mit einer Auflösung von 330 mal 185 bzw. 195 mal 109 ein Bild pro Frame. Ein UDP-Package entspricht dabei, genau einem Bild in der jeweiligen Qualität. Das Besondere dabei ist, dass jedes Bild von einer *RenderTexture* gelesen, auf einen Stack gepusht und innerhalb eines Threads nach und nach verarbeitet und gesendet wird. Das Problem, das zu dieser Lösung führte, war, dass es in Unity bisher keine performante Möglichkeit gibt, ein Bild des Kameraobjektes oder ein Video-Stream von einer Szene zu erhalten. Jedoch ist es notwendig möglichst wenig Verzögerung beim Senden des Streams zu gewährleisten, da auf Empfängerseite möglicherweise z.B. dem Probanden Anweisungen gegeben werden sollen und der Versuchsleiter sonst nicht auf das Geschehen eingehen kann. Auch die Methode *ReadPixels*, die auf die Textur angewendet wird, um ein Bild bei unserer Lösung zu erhalten, ist zurzeit der Flaschenhals.

6.3.5. Kommunikation zu Pupil Capture

Die Kommunikation zu Pupil Capture aus Unity heraus, wurde auf Basis des *Pupil-Listener* Scripts von Pupil Labs realisiert. Grundsätzlich dient unser modifiziertes Skript dazu, alle für uns relevanten Augendaten z.B. Blickdaten in gefilterter oder ungefilterter Form zu erhalten. Jedoch werden auch alle durch Pupil Capture zur Verfügung gestellten Daten auch empfangen. Das Framework kann dadurch bei Bedarf, um noch nicht enthaltene Funktionen mit den gewünschte Daten, jederzeit erweitert werden. Z.B. im Falle um eine weitere Interaktionstechnik, die auf der Pupillengröße basiert.
Die Modifikation für unseren Anwendungszweck brachte eine Vielzahl von Änderungen mit sich. Einige werden im Folgenden beschrieben:

Umstrukturierung der Datenhaltung

Zur einfacheren Verwendung haben wir die Datenhaltung maßgeblich umstrukturiert. Dazu wurden, die vorher verwendeten internen Klassen in eigene Klassen ausgegliedert und diese innerhalb des *PupilListeners* instantiiert.

```
1  protected readonly float ThresholdConfidence = 0.8f;
2  protected readonly float ThresholdDistance = 0.2f;
3  protected readonly double ThresholdTime = 0.5;
```

```
4
5   protected PupilData eyeLData = new PupilData();
6   protected PupilData eyeRData = new PupilData();
7   protected GazeData gazeData = new GazeData();
8   protected PupilData eyeLFilteredData = new PupilData();
9   protected PupilData eyeRFilteredData = new PupilData();
10   protected GazeData gazeFilteredData = new GazeData();
```

Sowohl *PupilData* als auch *GazeData* halten dabei die von Pupil Remote, der Schnittstelle zu Pupil Capture, die Eye-Tracking-Daten außerhalb der Anwendung nutzbar macht, zur Verfügung gestellten Daten. U.a. enthält jeder *GazeData*-Datensatz dabei z.B. einen Zeitstempel, einen Konfidenzwert sowie den *PupilData*-Datensatz auf dem es basiert. *PupilData* selbst repräsentiert den Datensatz eines Auges und hält dabei Daten z.B. zur Konfidenz, der Ausrichtung des Blickes in Polarkoordianten oder auch einen Zeitstempel wann der Datensatz erzeugt wurde.

Thread-Handling

Ordentliches Thread-Handling z.B. durch stoppen des Threads im Falle des Schließens der Applikation:

```
1       void OnApplicationQuit()
2       {
3           lock (thisLock_)stop_thread_ = true;
4           client_thread_.Join();
5           Debug.Log("Quit the thread.");
6       }
```

Filterung der Daten

Der nachfolgende Code-Abschnitt zeigt die Filterung der *GazeData*, der *eyeLData* sowie *eyeRData* auf Aktualität und Unsicherheit der Pupillen-Erkennung. Falls z.B. ein *EyeData*-Datensatz verworfen wird, wird der dazugehörige Gaze-Datensatz (per *timestamp*-Zuordnung mit Toleranz, da die Daten nicht gleichzeitig versendet werden) ebenfalls verworfen.

```
1  // filtering gazeData: check if it inside the Screen and
       has a newer timestamp, if not, throw it away
2   if (this.gazeData.timestamp - this.gazeFilteredData.
       timestamp >= this.ThresholdTime
3  || (currentGazePosition.x > 0 && currentGazePosition.x <
       1.0f
4   && currentGazePosition.y > 0 && currentGazePosition.y <
       1.0f
```

```
5  && this.gazeFilteredData.timestamp < this.gazeData.
       timestamp
6   && this.gazeData.confidence < this.ThresholdConfidence))
7  {
8  this.gazeFilteredData = this.gazeData;
9  }
```

6.4. Tests

Um die Funktionalität des Frameworks sicherzustellen, wurde das Framework getestet. Dazu haben wir zum einen Unit-Tests, zum anderen Integrationstest genutzt bzw. durchgeführt. Im Folgenden werden zunächst die Unit-Tests, anschließend die Integrationstests vorgestellt.

6.4.1. Unit-Tests

Es wurden Unit Tests für alle drei Services erstellt. Hauptaugenmerk der Unit-Tests war zum einem die Verarbeitung der Blickdaten und zum anderen ob diese richtig bestimmt wurde und eine Interaktion gestartet werden sollte oder nicht.
Beim *FixationDetectionService* und *PursuitDetectionService* sind dies die *FixedUpdate*-Methoden, da in diesen die Datenverarbeitung stattfindet. Beim *GestureDetectionService* ist es die *OnFixStarted*-Methode. Neben der Blickdatenverarbeitung wurden weitere Methoden wie die An- und Abmeldung getestet.
Die Tests des *FixationDetectionService* bestanden aus Datensätzen von Blickpunkten, mit denen die verschiedenen Arten von Fixationen getestet wurden. Dafür wurden die folgenden Fälle getestet:

- Es soll keine Fixation stattfinden
- Es sollen Interaktionen ausgelöst werden

Ähnlich sieht es bei den Tests für des *PursuitDetectionServices* aus. Auch dort wurde durch gegebene Datensätze die Erkennung von Interaktionen getestet. Dort werden die folgenden Fälle überprüft:

- Die Verfolgung wird erkannt
- Die Verfolgung wird nicht erkannt

- Die Verfolgung eines Objektes wird erkannt, während sich noch ein weiteres Objekt bewegt

Die Tests des *GestureDetectionService* bestehen zum einen aus Tests, die überprüfen ob eine einzelne Geste korrekt ausgelöst oder abgebrochen wird. Des Weiteren wird überprüft, ob dies auch für mehrere Gesten richtig ausgelöst wird.

- Die Geste wurde richtig ausgelöst
- Die Geste wurde korrekt abgebrochen
- Eine Geste wurde richtig ausgelöst während eine andere Geste nicht ausgelöst wurde
- Mehrere Gesten wurden richtig ausgelöst

6.4.2. Integrationstests

Im Folgenden werden die drei Services durch Integrationstests evaluiert. Hierbei werden pro Service mehrere Ereignisse getestet und verschiedene Interaktionsmöglichkeiten genutzt.
Wie Sommerville beschreibt, werden durch einen Integrationstest abschließend alle „einzelnen Programmeinheiten oder Programme [...] integriert und als Ganzes getestet um sicherzustellen, dass die Softwareanforderungen erfüllt werden“ ([Som12], S.57). Anschließend erfolgt meistens die Auslieferung an den Kunden.
Für die Integrationstests des Frameworks wurden ebenfalls die einzelnen Komponenten als Ganzes in realer Anwendungsumgebung getestet. Die einzelnen Integrationstests werden in den folgenden Abschnitten in Tabellenform dargestellt.

Fixation

Dieser Abschnitt beschäftigt sich mit den verschiedenen Ereignissen, die bei einer Fixation passieren können. Zu jedem Test wurde jeweils eine Szene erstellt, in der die beschriebenen Bedingungen herrschen.

Testbezeichnung	Allgemeine Fixation testen
Service	FixationDetectionService
Ablauf	Eine AR-Szene wird mit dem FixationDetectionService ausgestattet. Die Aufgabe besteht darin, ein gezeichnetes X im Raum zu finden und dieses zu fixieren. Dafür muss das Auge kalibriert werden.
Anfangsbedingung	Der Blickpunkt des Probanden befindet sich nicht auf dem gezeichneten X, daher muss kalibriert werden (vgl. 6.4.1).
Abschlussbedingung	Durch die Kalibrierung stimmt der Blickpunkt mit dem realen Blickpunkt des Probanden, der auf das gezeichnete X blickt, überein. Somit befindet sich der Blickpunkt ebenfalls auf das X (vgl. 6.4.2)

Tabelle 6.1.: Allgemeine Fixation

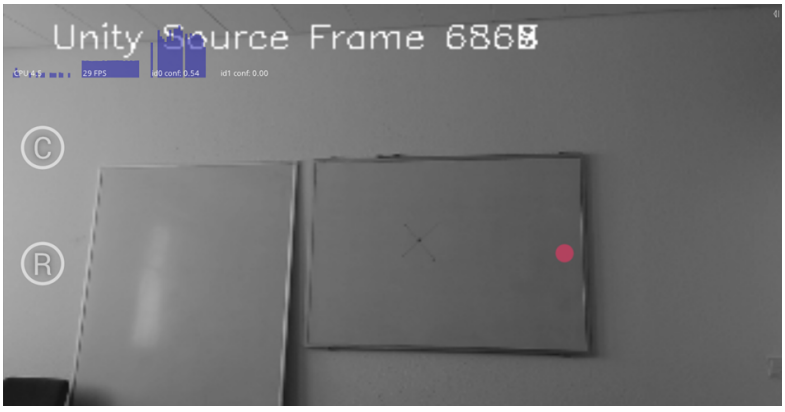

Abbildung 6.4.1.: Blickpunkt des Probanden ohne Kalibrierung befindet sich nicht auf dem X

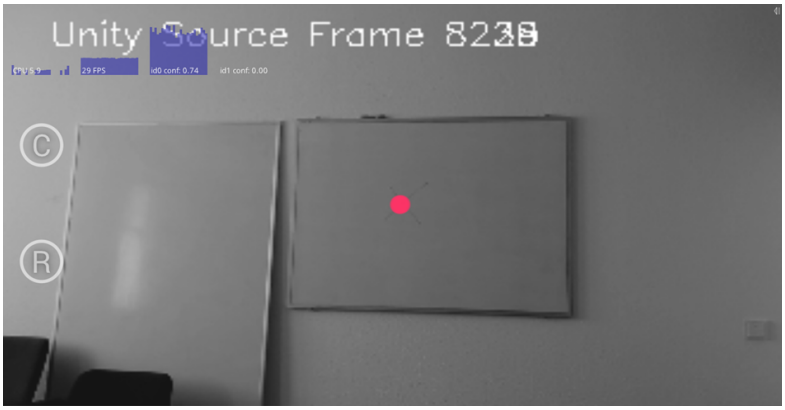

Abbildung 6.4.2.: Blickpunkt des Probanden mit Kalibrierung befindet sich auf dem X

Testbezeichnung	Bildschirmbezogene Fixation testen
Service	FixationDetectionService
Ablauf	Es wird eine AR-Szene gestartet, in der der rechte Bereich vom Bildschirm fixiert werden soll, und dadurch der Würfel in der Mitte die Farbe verändern soll.
Anfangsbedingung	Zu Beginn sind die Augen des Probanden nicht kalibiert und der Würfel im Mitte des Bildschrimbereich ist weiß. (vgl. 6.4.3)
Abschlussbedingung	Der Würfel hat seine Farbe in rot geändert dadurch das der Proband den rechten Bildschirmbereich fixiert hat. (vgl. 6.4.4)

Tabelle 6.2.: Bildschirmbezogen Fixation

Abbildung 6.4.3.: Mittige Fixation des Probanden führt nicht zum Auslösen der Interaktionstechnik

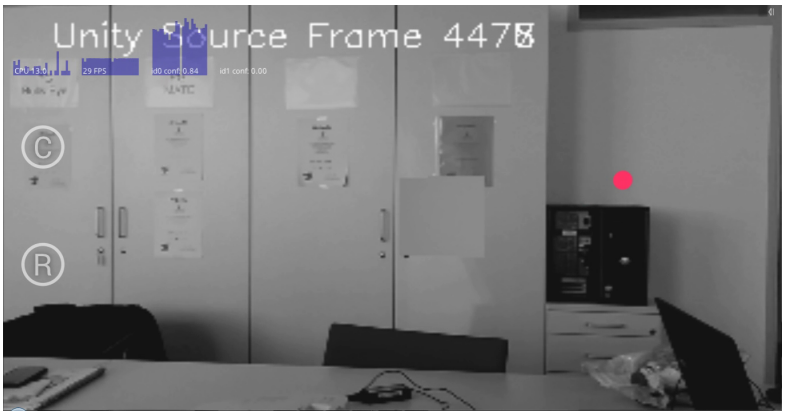

Abbildung 6.4.4.: Fixation des Bildschirmrandes hat die Aktion ausgelöst

Testbezeichnung	Objektbezogene Fixation testen
Service	FixationDetectionService
Ablauf	Es wurde eine AR-Szene gestartet, in der der Proband eine Kugel fixieren soll, die dadurch verschwindet und an einer neuen Stelle im Raum auftaucht.
Anfangsbedingung	Die Kugel befindet sich im Raum und der Blickpunkt des Probanden muss kalibriert werden um eine Fixation aufzulösen (vgl. 6.4.5).
Abschlussbedingung	Durch den kalibrierten Blickpunkt fixiert der Proband die Kugel, die dadurch verschwindet und an einer anderen Stelle im AR-Raum wieder auftaucht (vgl. 6.4.6 und 6.4.7)

Tabelle 6.3.: Objektbezogene Fixation

Abbildung 6.4.5.: Objekt im AR-Raum ohne Fixation

Abbildung 6.4.6.: Annäherung der Blickdaten an die Kugel

Abbildung 6.4.7.: Fixation des Probanden hat die Aktion ausgelöst

Gesten

Dieser Abschnitt führt die Integrationstests in Bezug auf die Gesten auf. Hierbei werden sowohl unterschiedliche wie auch sehr ähnliche Gesten evaluiert. Die dazu genutzte Szene wurde als Proof-of-Concept konzipiert. Nähere Informationen im Kapitel 7.2.2.

Testbezeichnung	Allgemeine Geste testen
Service	GestureDetectionService
Ablauf	Es wird eine AR-Szene (näher beschrieben im Kapitel 7.2.2) gestartet in der eine gerade Linie von links nach rechts und danach wieder zurück als Geste mit den Augen gezeichnet werden soll, um dadurch die Eingabe abzubrechen.
Anfangsbedingung	Es wurde eine Spielzug vorher durchgeführt und die Szene befindet sich nicht mehr im Ursprungszustand. (vgl. 6.4.8)
Abschlussbedingung	Die Geste wurde erfolgreich durchgeführt, erkannt und dadurch die Eingabe des aktuellen Spielzugs abgebrochen. (vgl. 6.4.9)

Tabelle 6.4.: Gesten testen

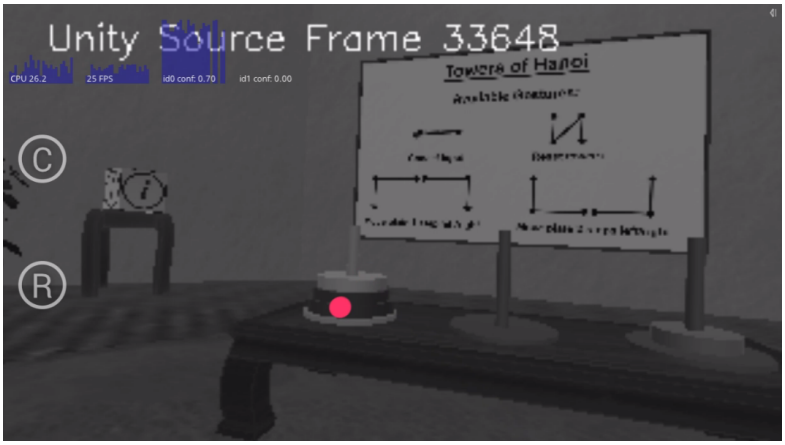

Abbildung 6.4.8.: Ausgangssituation

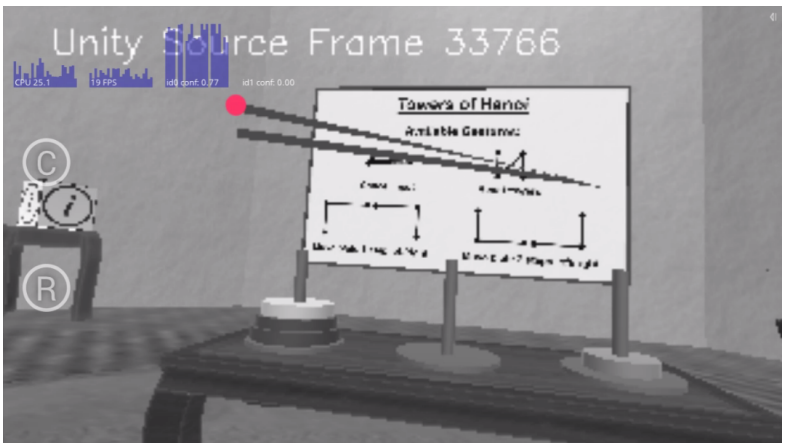

Abbildung 6.4.9.: Erfolgreich erkannte Geste

Testbezeichnung	Zwei ähnliche Gesten testen
Service	GestureDetectionService
Ablauf	Es wird eine AR-Szene (näher beschrieben im Kapitel 7.2.2) gestartet in der einerseits mit einer geraden Linie von rechts nach links und dann nach unten, ein Spielzug von rechts nach links symbolisiert wird. Andererseits mit einer geraden Linie von links nach rechts und danach wieder nach unten, ein Spielzug von links nach rechts getätigt wird. Der Proband zeichnet mit den Augen erst die Geste von rechts nach links danach von links nach rechts.
Anfangsbedingung	Die Ausgangssituation sieht vor, dass ein Spielzug in die gewünschte Richtung durchgeführt werden kann. (vgl. 6.4.10 und 6.4.12)
Abschlussbedingung	Die jeweilige Geste bewirkte einen Spielzug, bei der die Platte nach links bewegt wurde. (vgl. 6.4.11 und 6.4.13)

Tabelle 6.5.: Ähnliche Gesten

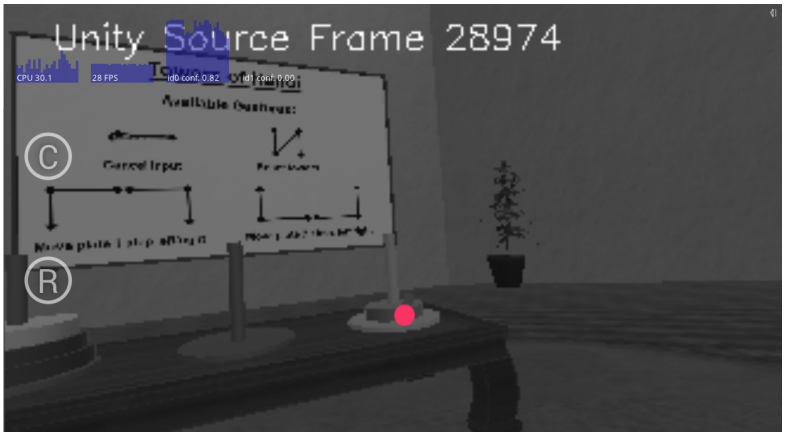

Abbildung 6.4.10.: Beispielhafte Ausgangssituation eines Spielzugs von rechts nach links

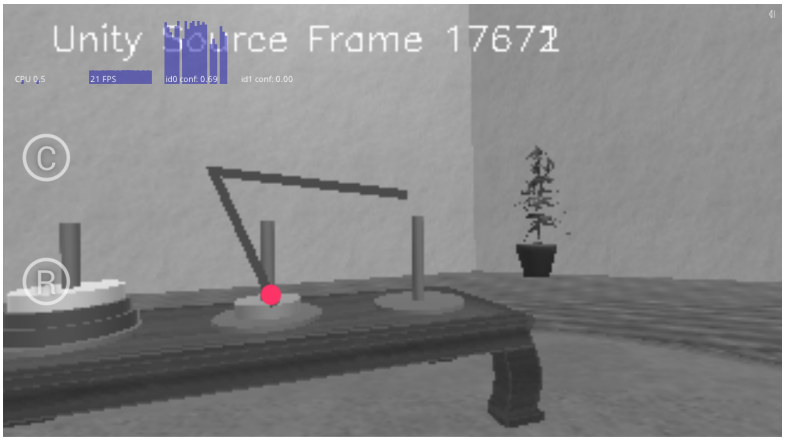

Abbildung 6.4.11.: Erfolgreich erkannte Geste für einen Spielzug von rechts nach links

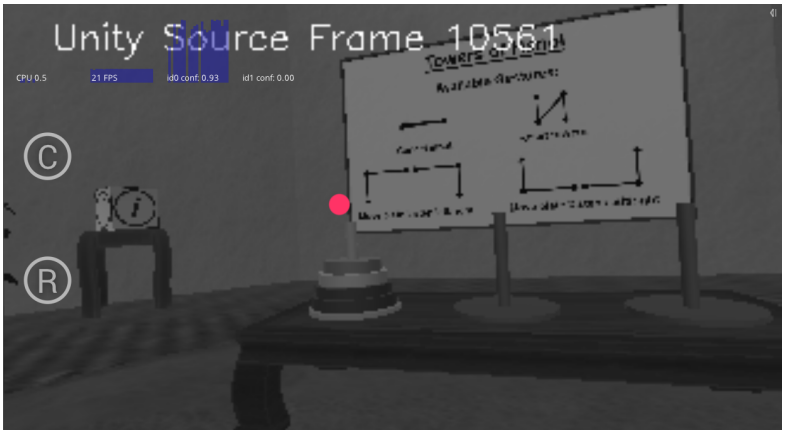

Abbildung 6.4.12.: Beispielhafte Ausgangssituation eines Spielzugs von links nach rechts

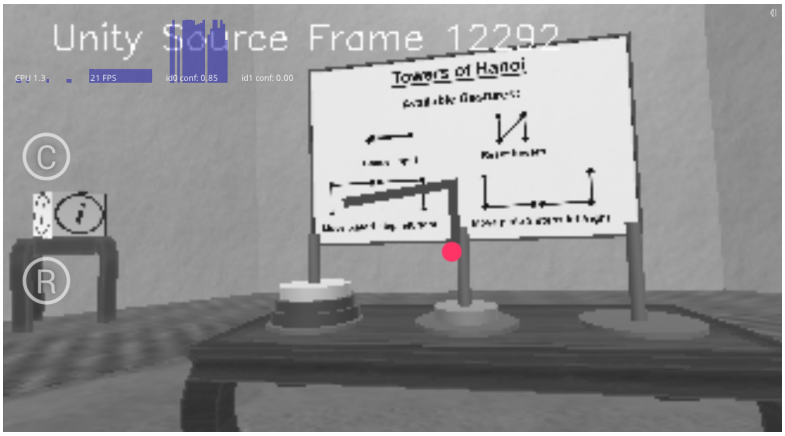

Abbildung 6.4.13.: Erfolgreich erkannte Geste für einen Spielzug von links nach rechts

Testbezeichnung	Zwei unterschiedliche Gesten testen
Service	GestureDetectionService
Ablauf	Es wird eine AR-Szene (näher beschrieben im Kapitel 7.2.2) gestartet in der zwei unterschiedlichen Gesten vorliegen. Die eine Geste ist der Buchstabe N, die andere ein liegendes L. Durch das Zeichnen eines Ns wird die komplette Szene zurückgesetzt und durch ein liegendes L zwei Spielzüge nach rechts getätigt. Der Proband zeichnet mit den Augen erst ein N, danach ein liegendes L.
Anfangsbedingung	Es wurden Spielzüge durchgeführt. (vgl. 6.4.14)
Abschlussbedingung	Die Gesten wurden korrekt erkannt und es konnte nach dem Zurücksetzen durch die Eingabe einer N Geste (vgl. 6.4.15 und 6.4.16), die liegende L Geste gezeichnet werden, um zwei Spielzüge nach rechts zu bewirken. (vgl. 6.4.17)

Tabelle 6.6.: Unterschiedliche Gesten

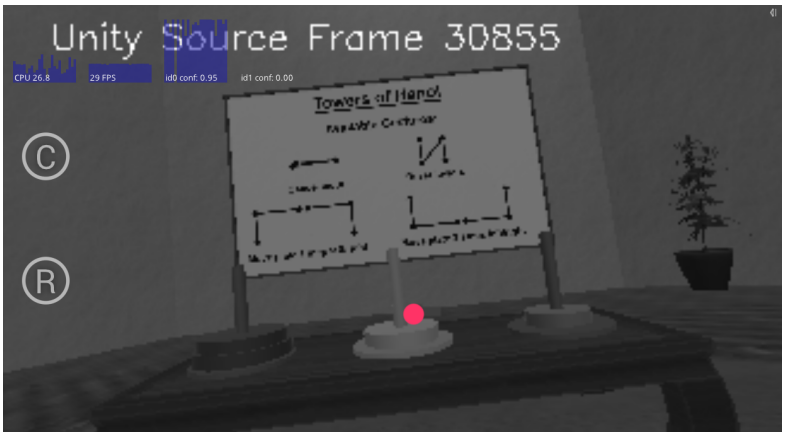

Abbildung 6.4.14.: Beispielhafte Ausgangssituation für das Zurücksetzen der Szene

Abbildung 6.4.15.: Erfolgreich erkannte N-Geste zum Zurücksetzen der Szene

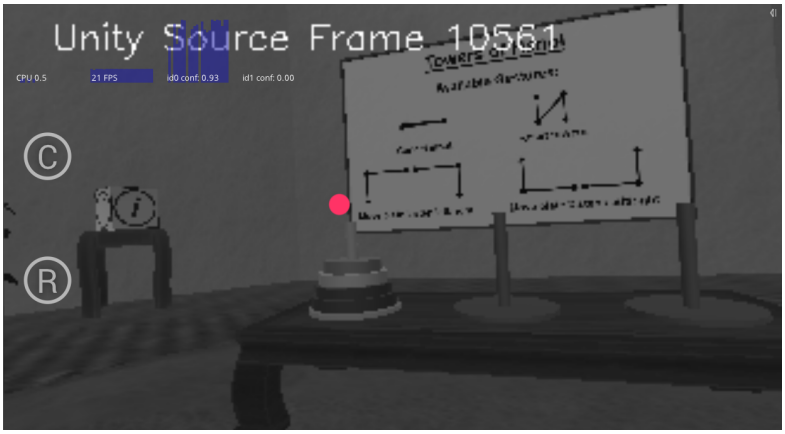

Abbildung 6.4.16.: Ausgangssituation nach dem Zurücksetzung der Szene

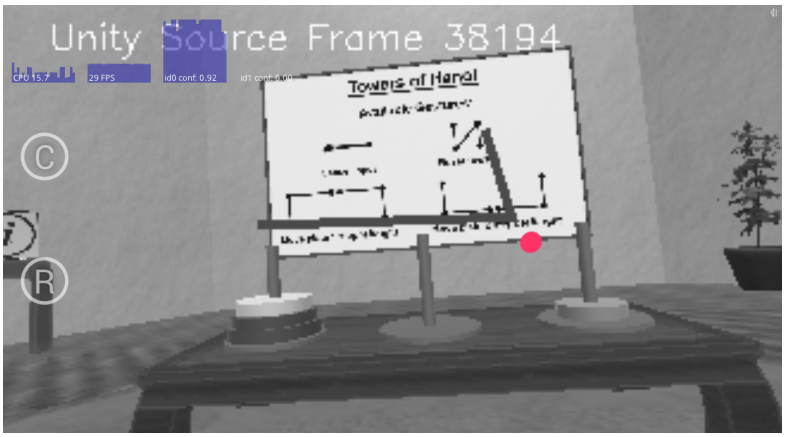

Abbildung 6.4.17.: Erfolgreich erkannte liegende L-Geste zum Durchführen zweier Spielzüge nach rechts

Verfolgung

Der Abschnitt beschreibt die Tests in Bezug auf die Verfolgung von Objekten. Hierzu wurde wieder eigenständigen Szenen gebaut, die die Verfolgung der Bewegung einer Kugel ermöglichen.

Testbezeichnung	Verfolgung eines bewegenden Objektes
Service	PursuitDetectionService
Ablauf	Es wird eine AR-Szene gestartet in der zwei Kugeln sich nebeneinander gegen den Uhrzeigersinn im Kreis drehen. Es soll hierbei die rechte Kugel verfolgt werden, ohne dass der Blickpunkt sich auf dieser Kugel befindet. Hierfür dient die linke Kugel als Hilfestellung für den Probanden. Sobald die linke Kugel verfolgt wird, verfärbt sich die rechte Kugel. Die Bewegungen der Kugeln sind identisch.
Anfangsbedingung	Es wurde eine Kalibrierung durchgeführt und die Szene gestartet. Die rechte Kugel ist schwarz gefärbt. (vgl.6.4.18)
Abschlussbedingung	Der Proband folgt der linken Kugel (als Hilfestellung) und die Verfolgung wird erkannt. Dies bewirkt, dass sich die rechte Kugel heller färbt und sich farblich der Linken angleicht. (vgl.6.4.19)

Tabelle 6.7.: Verfolgung eines Objektes

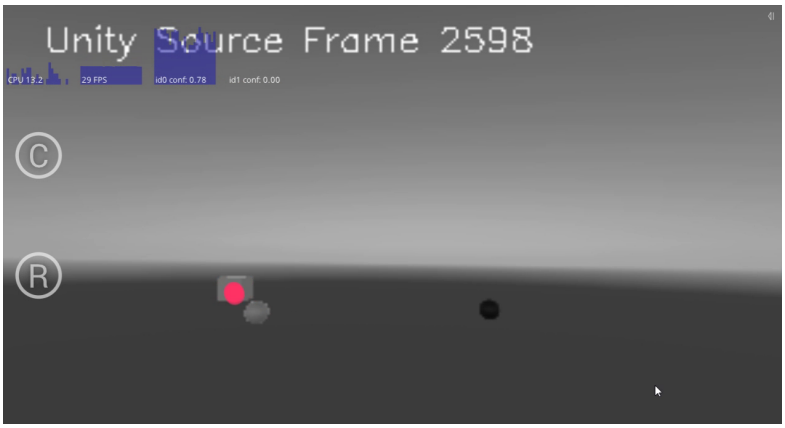

Abbildung 6.4.18.: Ausgangssituation der Szene

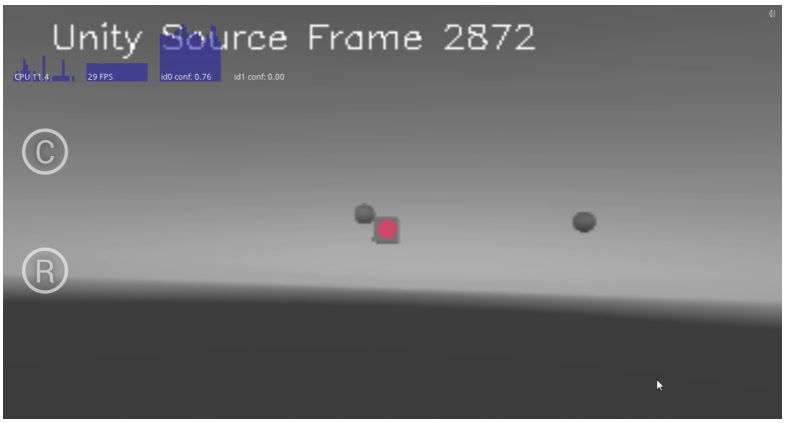

Abbildung 6.4.19.: Erfolgreich erkannte Verfolgung. Die linke Kugel verfärbt sich.

Testbezeichnung	Verfolgung eines bewegenden Objektes, unter zwei gegebenen Objekten das „Richtige“ auswählen
Service	PursuitDetectionService
Ablauf	Es wird eine AR-Szene gestartet in der zwei Kugeln sich nebeneinander im Kreis drehen. Die linke Kugel dreht sich gegen und die rechte Kugel mit dem Uhrzeigersinn. Sobald eine der jeweiligen Kugeln verfolgt wird, verfärbt sich diese.
Anfangsbedingung	Es wurde eine Kalibrierung durchgeführt und die Szene gestartet. Beide Kugeln sind schwarz gefärbt. (vgl.6.4.20)
Abschlussbedingung	Der Proband wählt eine Kugel aus und verfolgt die Bewegung. Die Verfolgung der Bewegung der gewählten Kugel wird erkannt und sie verfärbt sich. Die Farbe der anderen Kugel verändert sich nicht. (vgl.6.4.21)

Tabelle 6.8.: Auswahl des richtigen Objektes

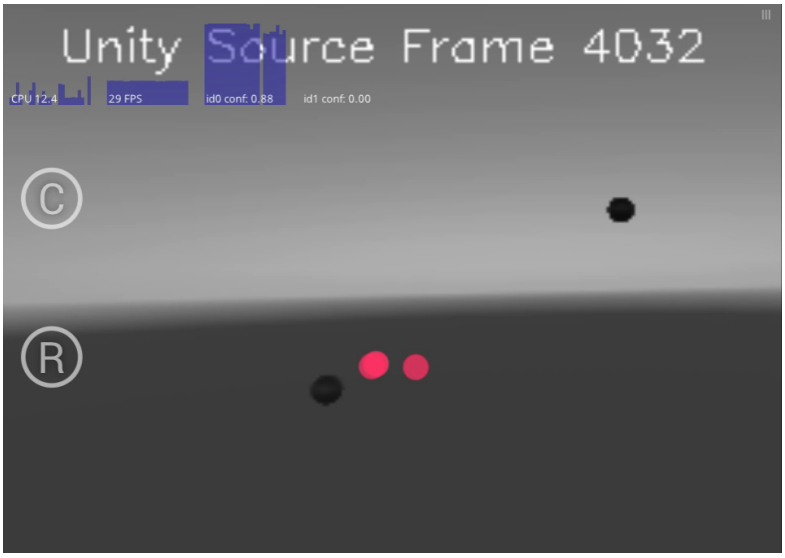

Abbildung 6.4.20.: Ausgangssituation der Szene

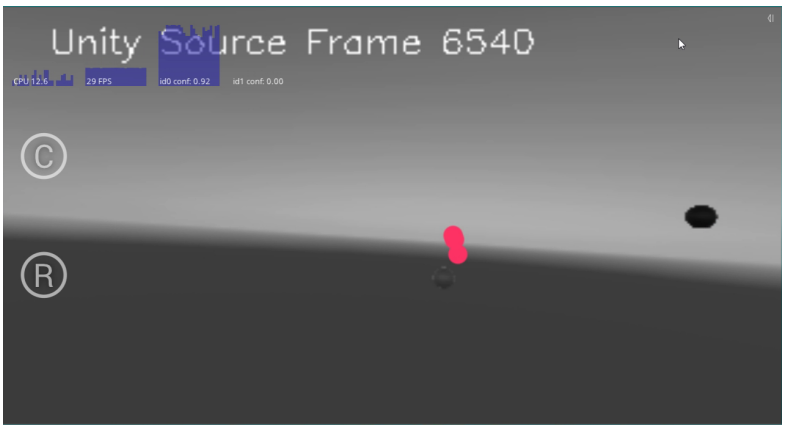

Abbildung 6.4.21.: Erfolgreich erkannte Verfolgung. Die linke Kugel wurde verfolgt und verfärbt sich. Die rechte Kugel mit entgegengesetzter Bewegung nicht.

7. Evaluation

In diesem Kapitel geht es um die durchgeführten Nutzertests, die den Teilnehmern der Studien einen Einblick in die Nutzung unseres Frameworks und dem HMD geben sollte. Zum einen aus der Perspektive des Wissenschaftlers und zum anderen auch aus der des Probanden. Zudem wurde die Einfachheit der Implementierung getestet.
Außerdem besteht das Kapitel aus den für die Nutzertests erstellten Proof-of-Concept Szenen. Diese werden sowohl mit Bild- als auch mit Codematerial dargestellt.

7.1. Nutzertests

Im Rahmen des Projektes wurden Studien in Form von Nutzertests durchgeführt, um die Benutzerfreundlichkeit des Frameworks zu testen, sowie Feedback sowohl für das Framework als auch für den Hardware-Prototyp und die Anleitungen zu erhalten. Es wurden insgesamt zwei Studien durchgeführt. Die erste Studie fand gegen Ende des zweiten Quartals und die zweite Studie gegen Ende des vierten Quartals statt.
In den folgenden Abschnitten wird der Aufbau und Durchführung beider Studien beschrieben.

7.1.1. Erste Studie

In der ersten Studie lag der Fokus auf dem Konzept und Nutzung des Frameworks. Es fanden insgesamt fünf Nutzertests während der ersten Studie statt. Bei jedem Nutzertest waren jeweils ein Versuchsleiter, ein Protokollant, ein Versuchsassistent sowie eine Testperson anwesend. Der Ablauf der Studie ist in mehrere Abschnitte unterteilt.
Nach der Begrüßung und Einführung durch den Versuchsleiter wird der Testperson die Nutzung des Frameworks aus der Sicht des Wissenschaftlers gezeigt, wobei die Testperson zum „lauten Denken“ motiviert wird. Hierfür setzt sich der Versuchsassistenten den Prototypen des HMDs auf, während auf einem Rechner die Eye-Tracking Software Pupil Capture gestartet wird. Der Testperson wird die World View sowie Eye View von Pupil Capture erklärt und Fragen bzgl. der Software

gestellt.
Anschließend setzt die Testperson das HMD auf und schaut sich die VR-Applikation aus der Sicht des Probanden an. Hierbei wird der Testperson Fragen zur Hardware und dessen Tragekomfort gestellt. Zum Abschluss des Nutzertests werden der Testperson Fragen zum Gesamteindruck, der Nutzung des Frameworks sowie dem Prototyp gestellt und abschließend sich verabschiedet.

7.1.2. Zweite Studie

Der Fokus in der zweiten Studie lag auf der Benutzerfreundlichkeit, der Implementierung mit dem Framework und der Anleitungen. Zusätzlich sollte der Tragekomfort des Prototyps erneut evaluiert werden. Es fanden diesmal insgesamt sechs Nutzertests statt. Der Ablauf dieser Studie war, wie schon bei den ersten Nutzertests, ebenfalls in mehrere Abschnitte unterteilt.
Nach der Begrüßung und Einführung musste die Testperson eine einfache Interaktionstechnik mit Hilfe der bereitgestellten Anleitungen selbst implementieren, dabei wurde auch hier die Testperson zum „lauten Denken“ motiviert. Anschließend wurden Fragen bzgl. der Benutzerfreundlichkeit der Anleitung und Implementierung gestellt. Danach wurde die Applikation mit der implementierten Interaktionstechnik auf das Projekt-Smartphone gebaut und die Testperson setzte den Prototypen auf. Anschließend wurde für das Eye-Tracking kalibriert und die Testperson konnte die selbst implementierte Interaktion ausprobieren. Zum Abschluss wurden nach dem Testen weitere Fragen bzgl. des Tragekomforts und Nutzung des Prototyps sowie des Gesamteindrucks gestellt und sich verabschiedet.

7.2. Proof-of-Concept-Szenen

In diesem Unterkapitel werden die von uns entwickelten Proof-of-Concept-Szenen thematisiert. Hierbei handelt es sich um Implementierungsbeispiele, die zeigen sollen, dass das entwickelte Framework im Sinne des Rapid-Prototyping genutzt werden kann. Zur einfachen Evaluations des Funktionsumfangs und der Benutzerfreundlichkeit des Frameworks, wurde zu jedem der drei Bausteine - Fixationen, Gesten und Verfolgungen - jeweils eine Beispielszene entwickelt welche im nachfolgenden beschrieben werden.

7.2.1. Fixation

Zur Evaluation des Fixations-Services wurde eine Proof-of-Concept-Szene entwickelt, die auch als Implementationsaufgabe in den zweiten Nutzertests verwendet

wurde. Die Szene benutzt unsere AR-Umgebung und enthält zusätzlich noch virtuelles Objekt. Dieses Objekt wurde mit einer fixationsbasierten Interaktionstechnik versehen. Diese wird ausgelöst, sobald der Nutzer das Objekt fixiert. Daraufhin wird das virtuelle Objekt zufällig in einem zuvor bestimmten Abstand vom Nutzer platziert. Danach kann der Nutzer das Objekt erneut suchen und die Interaktionstechnik nochmals auslösen.

7.2.2. Gesten

Als Proof-of-Concept für eine gestenbasierte Interaktionstechnik, die mit unserem Framework implementiert wurde, wurde eine Unity-Szene erstellt, bei der ein Nutzer drei Ringtürme vor sich sieht sowie eine Tafel mit Anweisungen und einen Info-Würfel. Zum Beginnen kann der Info-Würfel fixiert und so die Tafel geöffnet werden, dessen Anweisungen gefolgt werden kann. Die Tafel gibt Hinweise, wie mit bestimmten Gesten die Ringe bewegt werden können, sodass alle die sich in der ersten, linken Halterung befinden, vollständig zur rechten Halterung gelangen und sich wieder ein Turm aufbaut. Falls dies gelingt ist die Aufgabe erfolgreich abgeschlossen. Die dafür verwendbaren Gesten sind in Abbildung 7.2.1 zu sehen.

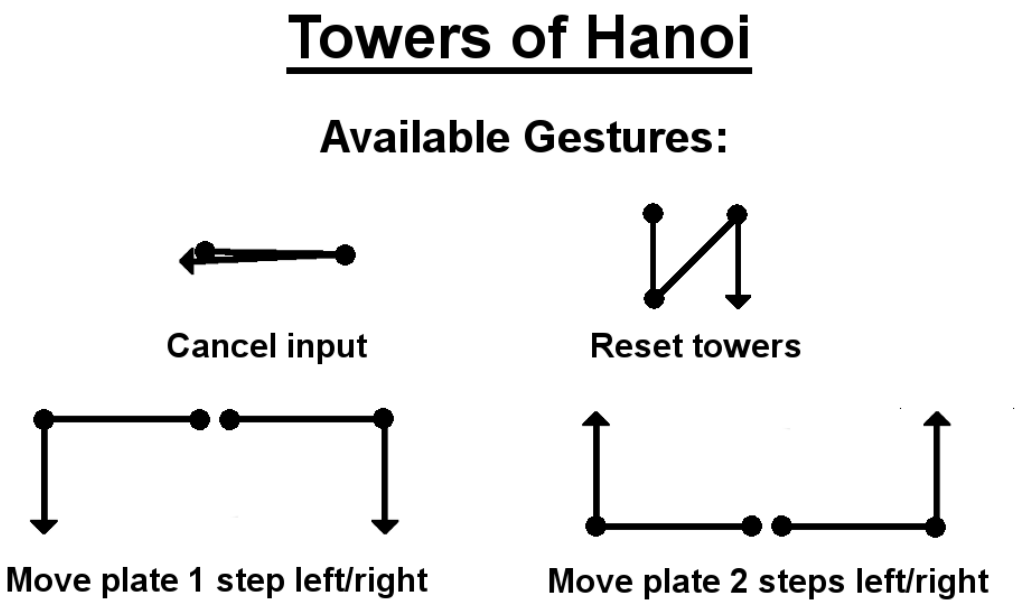

Abbildung 7.2.1.: Die verwendbaren Gesten der gestenbasierten Proof-of-Concept-Szene

7.2.3. Verfolgung

Als Proof-of-Concept des *PursuitDetectionServices* wurde eine AR-Szene entwickelt. Diese nutzt Vuforia, um mittels des integrierten Marker-Trackings die Bewegung eines Objektes zu erreichen. Dieses Objekt sollte sich bei Betrachtung vergrößern, bis er eine gewisse Größe erreicht wurde, um dann wieder bis zu einer vorgegebenen Größe zu schrumpfen und wieder zu wachsen. Auch bei einer Bewegung des Markers im reellen Raum, sollte der Würfel dieses Verhalten beibehalten, solange er durch den Blick des Nutzers verfolgt wird. Dieses wurde innerhalb eines Skriptes erreicht, dessen Implementation in Abbildung 7.2.2 zu sehen ist.

```
/// <summary>
/// Method called by unity on start.
/// </summary>
public void Start()
{
    ServiceProvider.Get<PursuitDetectionService>().Subscribe(this, 0.5f);
}

/// <summary>
/// Called if the pursuit continues. Update the size of the gameObject here.
/// </summary>
public void OnPursuitUpdate()
{
    // Calc the game objects new size between 0.1 and 0.3.
    var newSize = Mathf.PingPong(this.oldSize += 0.01f, 0.3f);

    // Add 0.2 to the size to get a min. size of 0.2 and a max. size of 0.5
    newSize += 0.2f;

    // Update the size of the gameobject.
    this.gameObject.transform.localScale = new Vector3(newSize, newSize, newSize);
}
```

Abbildung 7.2.2.: Beispielhafte Implementation der Pursuit-Szene

7.2.4. Kombination verschiedener Techniken

Neben den Szenen, die jeweils einzelne Aspekte des Frameworks gezeigt haben, wurde noch eine weitere Szene erstellt, die alle Facetten miteinander kombiniert. In dieser Szene wird Vuforia für die Markererkennung benutzt und sobald ein Marker im Bild zu sehen ist, kann der Nutzer mit einer einfachen Geste ein Spiel starten. Daraufhin werden Ballons erzeugt, die vom Marker aufsteigen und vom Nutzer per Fixation zum Platzen gebracht werden können. Sollten Ballons zu weit

nach oben fliegen platzen sie dort und der Nutzer verliert eins von anfangs fünf Leben.
Zusätzlich kann über einen zweiten Marker ein Menü aufgerufen werden, dessen Menüpunkte -Spiel stoppen, Spiel neustarten und Applikation beenden- über einen Orbit ausgewählt werden können.

8. Abschluss

In diesem Abschnitt werden die Ergebnisse und gewonnen Erkenntnisse dieser Projektgruppe zusammenfassend dargestellt und kritisch reflektiert. Da das Projekt unter der Open-Source-Lizenz veröffentlicht wird, erfolgt daraufhin noch ein Ausblick, welche Features durch die Funktionsweise des Frameworks in Zukunft denkbar sind. Ebenso werden mögliche Grenzen der Anwendung aufgezeigt.

8.1. Fazit

Die Aufgabe der Projektgruppe Bull's Eye war es, innerhalb eines einjährigen Projektes eine integrierte Lösung für Eye-Tracking in AR und VR für Rapid-Prototyping Prozesse in der Forschung zu entwickeln. Aus dem spezifischen Anwendungskontext des Rapid-Prototyping resultierten besondere Anforderungen. Die zu entwickelnde Lösung sollte unter niedrigen Kosten schnell und einfach reproduzierbar sein und die Veröffentlichung sollte zudem als Open-Source-Projekt unter MIT-Lizenz erfolgen.

8.1.1. Projektablauf

Das Projekt gliederte sich in vier Phasen, welche zunächst noch einmal zusammengefasst dargestellt und anschließend kritisch reflektiert werden. Die Phasen hatten jeweils ungefähr die Länge eines Quartals. Die erste Phase begann mit dem theoretischen Einarbeiten in die Materie, welches durch Seminararbeiten und anschließende Präsentationen sichergestellt wurde. Zudem sollte ein erster Hardware-Prototyp konzipiert werden. In der zweiten Phase sollte daraufhin auf Basis des erstellten Hardware-Prototypen ein Software-Framework implementiert werden. In der dritten Phase sollten die zugehörigen Bauanleitungen geschrieben, die Dokumentation verfasst und ein Test der „Bauanleitung“ durchgeführt werden. In der vierten und letzten Phase sollte das fertige Produkt schließlich evaluiert und mithilfe von Proof-of-Concept-Szenarien getestet werden.
Die Seminarphase in ersten Quartal war ein hilfreicher Einstieg in die theoretischen Inhalte. Die Projektgruppe konnte auch zu späteren Projektphasen von dem in den Arbeiten und Präsentationen aufbereitetem Wissen profitieren. Die angefertigten Ausarbeitungen dienten der Projektgruppe immer wieder als Nachschlagewerk. Bei

der Erstellung des Hardware-Prototypen wurde daraufhin deutlich, dass Hardware- und Softwareentwicklung nicht komplett losgelöst voneinander sein können, da bereits für die Entwicklung des Prototyps zugehörige Software entwickelt werden musste, damit die Funktionsweise demonstriert werden konnte.
In der zweiten Phase wurden dann aufgrund des entwickelten Software-Frameworks schnell die Grenzen des Hardware-Prototypen aufgezeigt, was eine iterative Neuentwicklung notwendig gemacht hat. Eine Besonderheit der zweiten Phase war die Powerwoche, in der sich alle Mitglieder eine Woche lang jeden Tag getroffen haben, um ganztägig an dem Software-Framework und dem Hardware-Prototypen zu arbeiten. Die Zusammenarbeit wurde von allen Beteiligten als äußerst positiv bewertet. Besonders die ersten beiden Phasen des Projektes vermittelten den Projektmitgliedern zudem die Fähigkeiten, Anforderungen in Nutzerinterviews zu erheben und diese hardware- und softwaretechnisch umzusetzen. Diese Fähigkeit kann sich, ungeachtet des Studienganges der jeweiligen Projektmitglieder, im späteren beruflichen Umfeld als sehr hilfreich erweisen und kann damit als wertvolle Praxiserfahrung angesehen werden. Zudem konnten durch die hohen Anforderungen an das Software-Framework die Programmierfähigkeiten aller Teilnehmer deutlich verbessert werden. Insbesondere da noch nicht alle Teilnehmer Erfahrungen hatten komplexe Software kollaborativ zu entwickeln.
In der dritten Phase wurde zunächst, wie ursprünglich im Projektplan vorgesehen, mit der Bauanleitung begonnen. Nach kurzer Zeit wurde jedoch seitens der Betreuer vorgeschlagen, die Phasen 3 und 4 zu tauschen, da es zu dem Zeitpunkt als sinnvoller erachtet wurde, zunächst das entwickelte Framework anhand von Proof-of-Concept-Szenarien zu evaluieren, bevor die finale Dokumentation angefertigt wurde. Dies hat der Projektgruppe, und insbesondere der Projekteleitung, zuerst einen hohen Mehraufwand bereitet, da die gesamte Quartalsplanung neu durchgeführt werden mussten. Langfristig betrachtet konnte hierdurch jedoch die Qualität des Endproduktes gesteigert werden, da somit genug Zeit vorhanden war, die in den Proof-of-Concept-Szenarien identifizierten Probleme zu beheben.
Die vierte Phase konzentriere sich auf die Anfertigung der Dokumentation und weiterer Fehlerbehebungen. Hier kam der Projektgruppe der Quartalstausch ebenso zugute, da deutlich weniger Fehler hätten behoben werden können, wenn die Proof-of-Concept-Tests erst im letzten Quartal stattgefunden hätten.

8.1.2. Organisation und Zusammenarbeit

Die Projektgruppe bestand aus sechs Studierenden aus unterschiedlichen Fachrichtungen. Es war von Anfang an ein einstimmig beschlossenes Nicht-Ziel der Projektgruppe, einzelne Personen die gesamte Projektlaufzeit über an einzelne Aufgabengebiete zu binden. Hierdurch konnte sichergestellt werden, dass die einzelnen Mitglieder Erfahrungen aus verschiedenen Bereichen sammeln und auch in

spezifischen Aufgabengebieten die Ideen von mehreren Personen genutzt werden konnten. Des Weiteren wurden verschiedene Rollen eingeführt und den Personen der Projektgruppe zugewiesen, sodass jede Person der Projektgruppe mehrere Rollen ausgeführt hat. Das Rollenmodell wurde innerhalb der Projektgruppe weitestgehend positiv aufgefasst, da somit Ansprechpartner und Verantwortlichkeiten für bestimmte Bereichen geschaffen werden konnten.
Es wurde sowohl in Einzel- als auch in Gruppentreffen zusammengearbeitet. Mit den Betreuern wurde sich einmal wöchentlich getroffen, um den aktuellen Stand vorzustellen, Feedback einzuholen und eventuelle Fragen zu klären. Im Anschluss an das Betreuertreffen hat die Projektgruppe die Zeit noch weiter für kollaboratives Arbeiten genutzt. Zusätzlich gab es in den ersten zwei Quartalen mindestens ein, in den letzten beiden Quartalen sogar mindestens zwei weitere Gruppentreffen pro Woche, in denen sich die Mitglieder getroffen haben, um gemeinsam an Lösungen zu arbeiten. Besonders durch die Powerwoche im zweiten Quartal wurde der Projektgruppe aufgezeigt, dass in Gruppentreffen deutlich effektiver an Lösungen gearbeitet werden konnte, was der Grund dafür war, dass es ab dem dritten Quartal insgesamt drei Gruppentreffen pro Woche gab. Hierbei wurde jedoch dynamisch Rücksicht auf die Termine einzelner Gruppenmitglieder genommen, sodass nicht immer alle sechs Teilnehmer präsent sein konnten.
Die Zusammenarbeit der Gruppe konnte außerdem durch mehrere Social-Events weiter gestärkt werden. Die Social-Event fanden anfangs ohne, später jedoch mit den Betreuern statt, wodurch das gute Verhältnis zwischen den Gruppenmitgliedern und den Betreuern ebenfalls gestärkt werden konnte.
Zusammenfassend lief sowohl die gruppeninterne Zusammenarbeit als auch die Zusammenarbeit mit den Betreuern in den meisten Fällen sehr positiv ab. Die aufgrund der geringen Teilnehmerzahl der Projektgruppe relativ hohe Arbeitslast konnte durch das häufige Zusammenarbeiten optimal begegnet werden.

8.1.3. Betreuung und langfristiger Nutzen

Ein weiteres Ziel der Projektgruppe war, im Sinne des Lehrauftrages der Universität, die Studierenden mit einer Aufgabe von hoher Komplexität zu konfrontieren, wodurch Selbstorganisation, Zeitmanagement, Zusammenarbeit, Projektmanagementmethoden und Kommunikationsfähigkeiten geschult werden sollten, was nahezu uneingeschränkt gelungen ist. Die Betreuer haben hierbei immer konstruktives Feedback gegeben und konnten, bei nicht durch die Projektmitglieder zu überwindenden Hürden, geeignete Hilfestellungen geben.
Es ist davon auszugehen, dass alle Mitglieder der Projektgruppe von den gewonnenen Erkenntnissen profitieren konnten und auch die Betreuer als Stakeholder, sowie die Abteilung Medieninformatik und Multimedia-Systeme können sowohl in praktischer wie auch theoretischer Hinsicht die erarbeiteten Lösungen langfristig

für die eigene Arbeit nutzen.

8.2. Ausblick und Abgrenzung

Auch nach erfolgreichem Abschluss der Projektgruppe gibt es noch Möglichkeiten, das entwickelte Framework und die modifizierte Hardware zu verbessern. Ein paar mögliche Verbesserungen werden im Folgenden vorgestellt.

8.2.1. Marker-Tracking

Die in das Framework integrierte Umsetzung von AR unterstützt momentan kein Marker-Tracking. Um diese durchaus wichtige Funktion allerdings doch in Kombination mit dem Framework verwenden zu können, wurde auf Vuforia zurückgegriffen. Da Vuforia allerdings nicht quelloffen verfügbar ist, können wir dieses nicht zusammen in einem Software-Paket mit unserem Framework anbieten. Eine eigene Implementierung des Marker-Trackings würde uns daher unabhängig von nicht quelloffener Software machen und es uns ermöglichen die gesamte Funktionalität in einem Paket zu veröffentlichen.

8.2.2. Verbesserung des Video-Streamings

Das Video-Streaming vom Smartphone zu dem Rechner, auf dem Pupil Capture läuft, ist momentan in einer sehr geringen Qualität. Diese Art der Videoübertragung wurde gewählt, um die Verzögerungen und Performanceminderungen minimal zu halten. Des Weiteren verwenden wir zum übertragen der Bilddaten nur ein einzelnes UDP-Paket, wodurch die Auflösung des Streams limitiert ist. In Verwendung des Frameworks zeigte sich jedoch, dass wir in der Lage wären ohne große Performance-Verluste eine höhere Auflösung zu übertragen. Dazu müsste das Streaming-Skript auf der Unity-Seite und das Backend für Pupil Capture an das Aufteilen und Zusammensetzen mehrere UDP-Pakete angepasst werden.

8.2.3. Erweiterung durch neue Arten von Services

Eine weitere Art das Framework zu erweitern, ist das Hinzufügen neuer Services, neben den von uns implementierten. In der Entwicklung des Frameworks wurde zwar durch die Services die in wissenschaftlichen Arbeiten gängigen Interaktionstechniken abgedeckt, jedoch sind durchaus weitere Arten von Interaktionen denkbar. Diese könnten ebenfalls standardisiert als Services verfügbar gemacht werden und wie die bereits vorhanden Services in dem Produkt inkludiert werden können.

8.2.4. Hinzufügen neuer Beispielinteraktionen

Für den einfachen Einstieg in die Verwendung unseres Frameworks wurden Szenen und Techniken erstellt, durch die in einfachen Beispielen gezeigt wird, wie das Framework verwendet werden kann. Durch weitere Techniken, die bereits fertig implementiert und damit Teil des Paketes sind, könnte das Framework die Nutzerfreundlichkeit weiter erhöhen, da weniger Techniken selbst implementiert werden müssten.

Literaturverzeichnis

[ABP17] Tom Alby, David Braun, and Sabine Pfleger. Agiles projektmanagement mit scrum — projektmanagement: Definitionen, einführungen und vorlagen, 2017. Abgerufen am 25.08.2017 von `http://projektmanagement-definitionen.de/glossar/scrum/`.

[BCL15] Mark Billinghurst, Adrian Clark, and Gun Lee. A survey of augmented reality. *Foundations and Trends in Human-Computer Interaction*, 8(2-3):73–272, 2015.

[BG10] Andreas Bulling and Hans Gellersen. Toward mobile eye-based human-computer interaction. *IEEE Pervasive Computing*, 9(4):8–12, October 2010.

[Bri09] M. Brill. *Einleitung*, pages 1–3. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[Bru03] F. Wilhelm Bruns. Lernen mit mixed reality. *ABWF (Ed): Kompetenzentwicklung 2003*, 2003.

[CMR+16] Daniel Clarke, Graham McGregor, Brianna Rubin, Jonathan Stanford, and T.C. Nicholas Graham. Arcaid: Addressing situation awareness and simulator sickness in a virtual reality pac-man game. In *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts*, CHI PLAY Companion '16, pages 39–45, New York, NY, USA, 2016. ACM.

[Cor16] Valve Corporation. Steam store front page. `http://store.steampowered.com/`, 2016. Accessed: 2016-11-4.

[DAQ17] DAQRI. Artoolkit feature comparison. Webseite, 2017. Abgerufen am 05.08.2017 von `https://github.com/artoolkit/artoolkit-docs/blob/master/1_Getting_Started/about_feature_comparison.md`.

[DBGJ13a] Ralf Dörner, Wolfgang Broll, Paul Grimm, and Bernhard Jung. *Virtual und Augmented Reality (VR / AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. EXamen.press. Imprint: Springer Vieweg, Berlin, Heidelberg, 2013.

[DBGJ13b] Ralf Dörner, Wolfgang Broll, Paul Grimm, and Bernhard Jung. *Virtual und Augmented Reality (VR / AR): Grundlagen und Methoden der Virtuellen und Augmentierten Realität*. Springer Vieweg, Berlin-Heidelberg, Deutschland, 2013.

[Dud16] Dudenverlag. Duden, gesichtsfeld. `http://www.duden.de/rechtschreibung/Gesichtsfeld`, 2016. Accessed: 2016-11-3.

[Elm] Rafiq Elmansy. Characteristics of human-centered design. Website. Abgerufen am 26.08.2017 von `http://www.designorate.com/characteristics-of-human-centered-design/`.

[Fel16] Jamie Feltham. Abrash: Vr needs to hit 16k to match 'retinal resolution'. `https://www.vrfocus.com/2015/03/abrash-vr-needs-hit-16k-match-retinal-resolution/`, 2016. Accessed: 2016-11-5.

[FK09] M. Friedrich and O. Kindel. *Softwareentwicklung mit AUTOSAR: Grundlagen, Engineering, Management in der Praxis*. dpunkt.verlag, 2009.

[Gei10] Thomas Geis. Neue iso 9241-210 „prozess zur gestaltung gebrauchstauglicher systeme“ ersetzt die iso 13407. Website, 2010. Abgerufen am 26.08.2017 von http://www.procontext.com/aktuelles/2010/03/iso-9241210-prozess-zur-entwicklung-gebrauchstauglicher-interaktiver-systeme-veroeffentlicht.html.

[GKR15] Ioannis Giannopoulos, Peter Kiefer, and Martin Raubal. Watch what i am looking at! eye gaze and head-mounted displays. 2015.

[Gmb16] Samsung Electronics GmbH. Samsung.com. `http://www.samsung.com/de/consumer/mobile-devices/wearables/gear/SM-R320NPWADBT`, 2016. Accessed: 2016-11-2.

[Gol09] Will Goldstone. *Unity game development essentials*. Packt Publishing Ltd, 2009.

[goo17a] Google vr sdk for unity. Webseite, 2017. Abgerufen am 13.08.2017 von `https://developers.google.com/vr/unity/`.

[Goo17b] Google. Eye-tracking - erkunden - google trends, 11.09.2017. Abgerufen am 11.09.2017 von `https://trends.google.de/trends/explore?date=2004-01-01%202017-09-11&q=%2Fm%2F0591td`.

[Goo17c] Google. Google cardboard – offizielle virtual-reality-brille – google store, 2017.

[Hes12] Roland Hess. *Blender Production: Creating Short Animations from Start to Finish*. CRC Press, 2012.

[HKBL14] Amy Henderson, Nicol Korner-Bitensky, and Mindy Levin. Virtual reality in stroke rehabilitation: a systematic review of its effectiveness for upper limb motor recovery. *Topics in stroke rehabilitation*, 2014.

[Hof13] D.W. Hoffmann. *Software-Qualität*. eXamen.press. Springer Berlin Heidelberg, 2013.

[Inc16] Oculus VR Inc. Oculus rift startseite. `https://www.oculus.com/`, 2016. Accessed: 2016-11-4.

[Jac90] Robert J. K. Jacob. What you look at is what you get: Eye movement-based interaction techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '90, pages 11–18, New York, NY, USA, 1990. ACM.

[Jac09] J.A. Jacko. *Human-Computer Interaction. New Trends: 13th International Conference, HCI International 2009, San Diego, CA, USA, July 19-24, 2009, Proceedings*. Number Teil 1 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

[JCL97] Sankar Jayaram, Hugh I Connacher, and Kevin W Lyons. Virtual assembly using virtual reality techniques. *Computer-Aided Design*, 29(8):575–584, 1997.

[Jon13] S.E. Jones. *The Emergence of the Digital Humanities*. Taylor & Francis, 2013.

[Koe09] Fachhochschule Koeln. Eyetracking, 2009.

[KR06] Uwe Kettermann and Andreas Rohde. *Spiele effektiv programmieren mit VB.net und DirectX*. Springer-Verlag, 2006.

[KR12] Greg Kipper and Joseph Rampolla. *Augmented Reality: an emerging technologies guide to AR*. Elsevier, 2012.

[Kra16] Gregory Kramida. Resolving the vergence-accommodation conflict in head-mounted displays. *IEEE transactions on visualization and computer graphics*, 22(7):1912–1931, 2016.

[Kri] Alexander Kriegisch. Scrum - auf einer seite erklärt. Abgerufen am 25.08.2017 von http://scrum-master.de/Was_ist_Scrum/Scrum_auf_einer_Seite_erklaert.

[LPA13] Trevor Lewis, Tobin Pereira, and Donita Almeida. Article: Smart scrolling based on eye tracking. *International Journal of Computer Applications*, 80(10):34–37, October 2013.

[Mag01] Martin Maguire. Methods to support human-centred design. Website, 2001. Abgerufen am 26.08.2017 von http://www.cse.chalmers.se/research/group/idc/ituniv/courses/06/ucd/papers/maguire%202001a.pdf.

[mic] Zusammenfassung (c# und java im vergleich. Website. Abgerufen am 13.08.2017 von https://msdn.microsoft.com/de-de/library/ms228361(v=vs.90).aspx.

[mil17] The linear mixed-reality spectrum of milgram and kashino. Website, 2017. Abgerufen am 27.09.2017 von https://www.researchgate.net/figure/284204855_fig_Figure-1-The-linear-mixed-reality-spectrum-of-Milgram-and-Kashino-1994.

[NN16] Max M North and Sarah M North. Virtual reality therapy. *Computer-Assisted and Web-Based Innovations in Psychology, Special Education, and Health*, page 141, 2016.

[Pre11] Prof. Dr. Lutz Prechelt. Vorlesung anwendungssysteme - bedienbarkeit/benutzbarkeit. http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-AWS-2011/42_Benutzbarkeit.pdf, 2011. Accessed: 2017-09-3.

[Pso95] Joseph Psotka. Immersive training systems: Virtual reality and education and training. *Instructional science*, 23(5-6):405–431, 1995.

[RPJ+14] Barbara Olasov Rothbaum, Matthew Price, Tanja Jovanovic, Seth D Norrholm, Maryrose Gerardi, Boadie Dunlop, Michael Davis, Bekh Bradley, Erica J Duncan, Albert Rizzo, et al. A randomized, double-blind evaluation of d-cycloserine or alprazolam combined with virtual reality exposure therapy for posttraumatic stress disorder in iraq and afghanistan war veterans. *American Journal of Psychiatry*, 171(6):640–648, 2014.

[SGR+02] Neal E Seymour, Anthony G Gallagher, Sanziana A Roman, Michael K O'Brien, Vipin K Bansal, Dana K Andersen, and Richard M Satava.

Virtual reality training improves operating room performance: results of a randomized, double-blinded study. *Annals of surgery*, 236(4):458–464, 2002.

[Shu11] Randall Shumaker. *Virtual and Mixed Reality - Systems and Applications: International Conference, Virtual and Mixed Reality 2011, Held as Part of HCI International 2011, Orlando, FL, USA, July 9-14, 2011, Proceedings, Part II*, volume 6774 of *SpringerLink : Bücher*. Springer-Verlag GmbH Berlin Heidelberg, Berlin, Heidelberg, 2011.

[SMH+12] K. Schoeffmann, B. Mérialdo, A.G. Hauptmann, C.W. Ngo, Y. Andreopoulos, and C. Breiteneder. *Advances in Multimedia Modeling: 18th International Conference, MMM 2012, Klagenfurt, Austria, January 4-6, 2012, Proceedings*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012.

[Som12] I. Sommerville. *Software Engineering*. Pearson Deutschland GmBH, 2012.

[Sta16] Digital Trends Staff. Spec comparison: Does the rift's touch update make it a true vive competitor? `http://www.digitaltrends.com/virtual-reality/oculus-rift-vs-htc-vive`, 2016. Accessed: 2016-10-31.

[Taz07] Johor Darul Tazim. Overview of open source augmented reality toolkit. 2007.

[Tea] Blender Documentation Team. *Decimate Modifier*. Zugriff am 02. November 2016.

[Tec13a] Unity Technologies. Unity - manual: Gameobjects, 2017-08-13.

[Tec13b] Unity Technologies. Unity - manual: Learning the interface, 2017-08-13.

[Tec13c] Unity Technologies. Unity - manual: Using components, 2017-08-13.

[Tec13d] Unity Technologies. Unity - multiplatform - publish your game to over 10 platforms, 2017-08-13.

[TPA+15] S Theis, C Pfendler, Th Alexander, A Mertens, Ch Brandl, and Ch M Schlick. Head-mounted displays–bedingungen des sicheren und beanspruchungsoptimalen einsatzes. *Physische Beanspruchung beim Einsatz von HMDs. Bundesanstalt für Arbeitsschutz und Arbeitsmedizin*, 2015.

[Tre12] T. Trepper. *Agil-systemisches Softwareprojektmanagement*. SpringerLink : Bücher. Springer Fachmedien Wiesbaden, 2012.

[Uni] Unity multiplatform support. Webseite. Abgerufen am 14.08.2017 von `https://unity3d.com/de/unity/features/multiplatform`.

[VB12] Vytautas Vaitukaitis and Andreas Bulling. Eye gesture recognition on portable devices. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 711–714, New York, NY, USA, 2012. ACM.

[VBG13] Mélodie Vidal, Andreas Bulling, and Hans Gellersen. Pursuits: Spontaneous interaction with displays based on smooth pursuit eye movement and moving targets. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 439–448, New York, NY, USA, 2013. ACM.

[Vur13] Benjamin Vurlod. Blender tutorial: textures optimization and baking for game environment. Video, September 2013. Zugriff am 05. November, 2016.

[WGA+17] Mike Weber, Stephan Gauch, Faruch Amini, Jens Tiemann, Carsten Schmoll, Lutz Henckel, Gabriele Goldacker, Petra Hoepner, Nadja Menz, Maximilian Schmidt, Michael Stemmer, Florian Weigand, Christian Welzel, Jonas Pattberg, Nicole Opiela, Florian Friederici, Jan Gottschick, and Jens Fromm. Mensch-maschine-interaktion - trendschau - aktuelle trend- und themensammlung — öffentliche it (öfit), 2017. Abgerufen am 12.09.2017 von `http://www.oeffentliche-it.de/-/mensch-maschine-interaktion`.

[Whe16] Andrew Wheeler. Duden, gesichtsfeld. `http://www.engineering.com/Hardware/ArticleID/12699/Understanding-Virtual-Reality-Headsets.aspx`, 2016. Accessed: 2016-11-3.

[ZMC+14] Yanxia Zhang, Jörg Müller, Ming Ki Chong, Andreas Bulling, and Hans Gellersen. Gazehorizon: enabling passers-by to interact with public displays by gaze. In *The 2014 ACM Conference on Ubiquitous Computing, UbiComp '14, Seattle, WA, USA, September 13-17, 2014*, pages 559–563, 2014.

A. Nutzertests-Fragen

A.1. Erste Studie

1. Haben Sie bereits Erfahrung mit Eye-Tracking-Software?
   a) Wenn ja:
      i. Welche Software haben sie verwendet?
      ii. Was gefällt Ihnen an Pupil Capture?
      iii. Was fehlt Ihnen an Pupil Capture?
   b) Wenn nein:
      i. Was können Sie sich an Ansprüchen an eine solche Software vorstellen?
2. Wie wichtig ist es Ihnen, während der Nutzer den Prototypen aufgesetzt hat, zu sehen, was der Proband sieht?
3. Gab es etwas, das beim Aufsetzen oder benutzen des Prototypens gestört hat?
   a) Konnten Sie das Display gut erkennen?
4. Wie empfinde Sie die Positionierung der Komponenten?
5. Wie ist Ihr Gesamteindruck von dem Prototypen?
6. Können Sie sich Aspekte vorstellen, die Sie bei der Benutzung des Produktes stören würden, oder die Sie von einer Nutzung abhalten würden.
7. Angenommen es wäre Ihr Ziel den Prototypen anhand einer Bauanleitung zusammenzubauen. Inwiefern glauben Sie stellt die Beschaffung der einzelnen Bestandteile eine Hürde für sie dar?
8. Inwiefern sind Sie mit den verwendeten Hardwarekomponenten bereits vertraut?
9. Fallen Ihnen Alternativen zu der verwendeten Hardware ein, die Ihrer Meinung nach einen Vorteil bieten würden?

10. In wie weit sehen Sie es als Problem an, dass der Prototyp mit einem Rechner verbunden ist?

11. Welche Feature vermissen Sie noch an dem Prototypen?

12. Was würden Sie am Prototypen ändern?

13. Eine mögliche Weiterentwicklung des Prototypen könnte die Verwendung eines weiteren Pi Zeros und einer weiteren Augenkamera beinhalten. Wie würden Sie bei dieser Modifikation den zusätzlichen Nutzen, aber auch den zusätzlichen Beschaffungs- und Installationsaufwand einschätzen?

A.2. Zweite Studie

1. Hatten Sie bereits Erfahrung mit der Unity-Engine?
   a) Wenn ja: Wie schätzen sie ihre Expertise in dem Umgang mit Unity ein?
   b) Wenn nein: Haben sie Erfahrungen in der Programmierung mit C#?

2. Gab es etwas dass ihnen bei der Implementierung der Interaktionstechnik Probleme bereitet hat? Wenn ja, was?

3. Wie ist ihr Gesamteindruck bzgl. der Bedienbarkeit und der Nutzerfreundlichkeit des Frameworks?

4. Wie bewerten sie die intuitive Bedienbarkeit?

5. Was glauben sie könnte noch verändert werden um die Implementierung einer Interaktionstechnik noch intuitiver zu gestalten?

6. Wie hilfreich fanden sie die Anleitung? Würden sie sich zusätzliche Hilfestellungen oder Inhalte von der Anleitung wünschen?

7. Haben Sie schon andere HMDs genutzt? Wenn ja, welche?
   a) Wenn Sie den Tragekomfort dieser vergleichen müssten, was wären nennenswerte Aspekte?

8. Wie ist Ihr Gesamteindruck vom Prototypen?

9. Gab es etwas, das beim Aufsetzen oder benutzen des Prototypens gestört hat?

10. Konnten Sie das Display vollständig erkennen?

11. Können Sie sich Aspekte vorstellen, die Sie bei der Benutzung des Produktes stören würden, oder die Sie von einer Nutzung abhalten würden.

12. Bei Brillenträger: Störte Sie Ihre Brille bei der Benutzung?

13. Angenommen es wäre Ihr Ziel den Prototypen anhand einer Bauanleitung zusammenzubauen. Inwiefern glauben Sie stellt die Beschaffung der einzelnen Bestandteile eine Hürde für sie dar?

14. Sehen Sie die Möglichkeit, das Framework und/oder das HMD im Forschungskontext einzusetzen?
    a) Wenn nein: Was würden sie sich wünschen damit das Produkt in ihrer Forschung zum Einsatz kommen kann?