UNIGIS Salzburg
Educating GIS Professionals Worldwide

geoinformation engineering ikg eth zurich

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Zur Erlangung des Grades

Master of Science in

Geographical Information Science & Systems

Erstellung und Evaluation einer service-orientierten Systemarchitektur zur ortsbezogenen blickbasierten Interaktion mit 3D-Geoobjekten

vorgelegt am 28. Februar 2014 von:

dipl. Forsting. ETH

Simon Haesler

u1552 Unigis Msc 2011

Lehrgangsleitung
Ao. Univ. Prof. Dr. Josef Strobl
Paris Lodron Universität
Unigis Msc Salzburg

Leitung und Bewertung
Prof. Dr. Martin Raubal
ETH Zürich
Geoinformations-Engineering

Betreuung
Dr. Peter Kiefer
Ioannis Giannopoulos
ETH Zürich
Geoinformations-Engineering

Danksagung

An dieser Stelle möchte ich dem gesamten UNIGIS-Team und insbesondere Julia Moser für die ausgezeichnete Betreuung während des gesamten UNIGIS-Studiums meinen Dank aussprechen. Sämtliche Anliegen und Anfragen wurden stets innert kürzester Zeit gelöst und beantwortet.

Ein ganz besonderer Dank gilt Herrn Professor Martin Raubal des Instituts für Kartografie und Geoinformation der ETH Zürich. Er ermöglichte es mir diese spannende Masterarbeit an der Professur für Geoinformation–Engineering zu absolvieren, dabei standen mir die notwendigen Räumlichkeiten, die IT-Infrastruktur sowie ein mobiler Eye Tracker der Professur stets zur Verfügung.

Ein weiterer herzlicher Dank geht an Dr. Peter Kiefer und Ioannis Giannopoulos, sie betreuten mich herausragend während der Erarbeitung der Thesis und standen mir stets für eine fachliche und auch persönliche Beratung zur Seite. Für die fachliche Unterstützung bei der Einarbeitung ins Thema der serviceorientierten Systemarchitektur möchte ich mich speziell bei Dr. Ionut Iosifescu Enescu bedanken.

Ein dickes Dankeschön geht auch an mein liebes Mami und an Andrea Leimgruber, welche sich die Zeit nahmen, meine Arbeit gegenzulesen und zu korrigieren.

Am innigsten und herzlichsten möchte ich mich bei meiner lieben schwangeren Frau Eva, meinem Sohn Michel und meiner Familie für die verständnisvolle Unterstützung bedanken. Während meines Studiums mussten sie oft gänzlich auf meine Anwesenheit und Mithilfe verzichten, so dass sie so manche Herausforderung alleine meistern mussten.

Danke!

Zusammenfassung

Diese Arbeit befasst sich damit, wie zukünftige Anwendungen Informationen über blickbasierte Interaktionen abrufen können. Dazu muss geklärt werden, wie ein solches System zu konzipieren ist. Weiter ist zu klären wie mit den Echtzeit Anforderungen, der grossen Anzahl Blickdaten und der nicht zu vermeidenden Berechnungszeit umzugehen ist. Aus Benutzersicht ist zu eruieren, ob der Benutzer die angeforderten Angaben erhält oder ob er durch unnötige Information überhäuft wird.

In dieser Arbeit wurde ein Prototyp entwickelt der einem erlaubt, nur mittels Blicken Gebäudenamen abzufragen. Der Prototyp besteht aus einer Client– und einer Server–Anwendung. Zur Positionsbestimmung und zur Blickaufzeichnung mit einer Frequenz von *25 Hz*, wurden ein Smartphone und ein mobiler Eye Tracker eingesetzt. Basierend auf diesen Sensordaten wurden die 3D–Blicke im schweizerischen Koordinatensystem berechnet. Die einzelnen Vektoren werden anschliessend zur Verschneidung mit einem 3D–Gebäudemodell an einen GIS–Server gesendet, so dass die Gebäudenamen ermittelt werden können.

Die Prototyp Evaluation zeigte, dass die Berechnungszeit zur Ermittlung des betrachteten Gebäudes im Schnitt 112 $\pm$11.86 Millisekunden dauert. Um eine Kumulation der Verzögerungen zu verhindern, sind nur die Fixationen im Blickverlauf zu berücksichtigen. Durch eine iterative Parameteranpassung im Fixationsalgorithmus, wurde eine verzögerungsfreie Berechnungsfrequenz erreicht. Entscheidend war, dass die ausgewiesene Berechnungsdauer, die Dauer der Fixationsintervalle nicht überschreitet. Zudem konnte eine mittlere horizontale Abweichung von 2.08° $\pm$0.46° und eine vertikale Abweichung von 0.14° $\pm$0.68° ausgemacht werden. Wird der Benutzer nur informiert, falls die kumulierte Fixationsdauer des betrachteten Objekts mindestens drei Sekunden beträgt, so sind 83% der erhaltenen Serverantworten relevant.

Als technischer Schwachpunkt konnte der GPS–Sensor bestätigt werden. Die Genauigkeit des Sensors lässt sich unter Umständen durch Echtzeit Korrekturdienste wie *„swipos–Nav“* verbessern oder durch komplexere Map–Matching–Algorithmen. In künftigen Arbeiten könnte auch geprüft werden, inwiefern sich eine Echtzeit Blickmustererkennung in die Anwendung integrieren lässt, um nicht angeforderte Informationen auszuschliessen.

Abstract

This work investigates how gaze-based interactions could be used – in future applications – to retrieve information about 3D–Objects. For this purpose, it must be clarified how such a system should be designed. Further it has to be clarified how to deal with real-time requirements, the large number of gaze data and the unavoidable computation time. From the user's point of view, it's going to be evaluated whether the user receives the requested information or if he is overwhelmed by unnecessary information.

In this work a prototype has been developed to get a building name only by looking at the target object. The prototype consists of a client and a server application. To determine the position and to record gazes with a frequency of *25 Hz*, a smartphone and a mobile eye tracker were used. Based on these sensor data, 3D-gazes were calculated in swiss coordinates. Afterward each vector is sent to a GIS-Server, to calculate the intersection with a 3D-building-model, so that the building name can be determined.

The evaluation of the prototype showed that the computation, to determine the building name, takes about 112 $\pm$11.86 milliseconds. By sending only fixations of a gaze scanpath, accumulation of the delay times could be avoided. Through an iterative adjustment of the parameters in the fixation algorithm, an instantaneous computation frequency could been reached. Thereby the crucial factor was, that the computation takes less time than a fixation interval. In addition the mean horizontal deviation of 2.08° $\pm$0.46° and the vertical deviation of 0.14° $\pm$0.68° was identified. If the user only gets a response, when the cumulated fixation duration - on the observed object -is at least three seconds, then 83% of the received information is relevant.

The GPS sensor was identified as a technical weakness of the system. The accuracy of the location data can be improved by using real–time correction services like „swipos–Nav“ or by using more sophisticated map–matching–algorithms. Future work may also consider to integrate a real time „gaze pattern recognition“ into the application.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Abkürzung:	**Bedeutung:**
AOI	Area-Of-Interest
d.h.	das heisst
DTM	Digitales Terrain Modell
GIS	Geografisches Informationssystem
GPS	Global Positioning System
GP-Service	Geoprocessing Service (hier: ESRI spezifischer Dienst)
GUI	Graphical User Interface
IDE	Integrierte Entwicklungsumgebung
JVM	Java Virtual Machine
REST	Representational State Transfer
RTT	Rount Trip Time (Huang et al., 2010)
SOAP	Simple Object Access Protocol
SOE	Server Object Extension
UTC	Coordinated Universal Time

Teil I

Einleitung

„Nichts ist im Verstand,
was nicht zuvor in der Wahrnehmung wäre.“

T. von Aquin, Theologe und Philosoph ⋆ 1225 – † 1274

Kapitel 1

Einführung ins Thema

1.1 Motivation

Wir nehmen unsere Umwelt über die uns zur Verfügung stehenden Sinne wahr. Dabei ist es nicht erstaunlich, dass bei einem gesunden Menschen rund 80% der Reizaufnahme über das Auge erfolgt (Dahm, 2006, S. 41). Dies ist der Grund, wieso die visuelle Wahrnehmung in der Forschung und insbesondere bei der Software–Ergonomie von grossem Interesse ist.

Zur Erforschung der visuellen Wahrnehmung, werden seit den Anfängen die Blicke des Menschen untersucht. Früher wurden die Blicke auf einer Glasscheibe aufgezeichnet, heute erfolgt die Aufzeichnung mittels stationären oder mobilen Eye Trackern. Die Auswertung der Blicke erfolgt beispielsweise über ihren zurückgelegten Weg (Scanpath), über Blickdichte–Karten (Heatmaps) oder „Areas of Interests".

Software-Ergonomie befasst sich seit Jahrzehnten damit, die Arbeitsbedingungen bei der Mensch–Computer–Interaktion auf die sensorischen und kognitiven Fähigkeiten des Menschen abzustimmen. Daher konnten auf die menschliche Wahrnehmung abgestimmte, demzufolge übersichtliche und benutzerfreundliche graphische Benutzeroberflächen gestaltet werden (Wandmacher, 1993).

Seitdem mobile Geräte – und all die darin enthaltenen Sensoren – sich auf dem Markt durchsetzten, brachte dies eine Fülle an neuen Softwarefähigkeiten mit sich. Hier seien speziell die ortsbezogenen Dienste und die erweiterte Realität erwähnt.

Ortsbezogene Dienste „*local based services*" ermöglichen es dem Benutzer, Informationen basierend auf seinem aktuellen Standort zu erhalten. Falls mit einem Smartphone in Luzern nach einem Kino gesucht wird, so werden aufgrund der GPS–Position bevorzugt Kinos der unmittelbaren Umgebung angezeigt.

Technologien der erweiterten Realität „*augmented reality*“ reichern in Echtzeit Abbildungen der realen Welt mit zusätzlichen, computergenerierten Informationen an (Carmigniani et al., 2011). Wird beispielsweise die Kamera eines Smartphones aktiviert, so ist der fokussierte Ausschnitt auf dem Display ersichtlich. Ausserdem wird das Video zusätzlich mit einer animierten 3D–Abbildung der alten Stadtmauer angereichert.

Das Wissen über die Wahrnehmung und das Sehen wird dabei bestenfalls zur benutzerfreundlichen Gestaltung der Anwendung genutzt und die Blicke dienen lediglich zur Informationsaufnahme. Die eigentliche Mensch–Maschine–Interaktion erfolgt nach wie vor über die klassischen Methoden und geschieht oftmals mit den Händen. Es wäre jedoch in vielen Situationen sehr hilfreich, wenn eine Anwendung mittels Blick gesteuert werden könnte. Dabei denke man an die unmittelbare Auswertung von Eye Tracker Aufnahmen oder die „handfreie“ Bedienung eines Navigationsgerätes während der Fahrt.

Diese Arbeit soll einen Beitrag dazu leisten, um zukünftig blickbasierte Mensch–Computer–Interaktion zu ermöglichen. Dabei liegt der Fokus auf der Aufbereitung von Informationen zu einem gerade betrachteten 3D–Objekt. Ein möglicher Anwendungszweck wäre eine Applikation, die einem Touristen Hintergrundinformationen zu der unmittelbar bestaunten Sehenswürdigkeit in Echtzeit liefert.

1.2 Zielsetzung der Arbeit

Die blickbasierte und ortsbezogene Interaktion mit Geoinformation ist ein noch weitgehend offenes Forschungsthema. In dieser Masterarbeit sollen dazu erste Erkenntnisse erzielt werden, sowohl aus technischer wie auch aus Benutzersicht. Aus technischer Sicht ist ein Prototyp zu entwickeln, dabei sind folgende Herausforderungen und Fragen zu lösen:

- Kombination der Eye Tracker Daten mit der aktuellen GPS–Position und Kopfausrichtung des Nutzers, zur Berechnung des 3D–Blickvektors
- Wie kann der existierende lokale Algorithmus[1] zum Schnitt im 3D-Stadtmodell in einen Geo Web Service umgewandelt werden?
- Bereitstellen von Informationen zum betrachteten Gebäude

Die aus der Evaluation des Prototyps gewonnenen Erkenntnisse werden eingesetzt Fragen zu beantworten, die sich aus Benutzersicht stellen:

- Wie ist mit den Echtzeit Anforderungen umzugehen, bezüglich der nicht zu vermeidenden Antwortzeit eines Web Services?
- Erhält der Nutzer die gewünschten Angaben oder eher Informationen, die nicht gefragt waren?

1.3 Aufbau der Arbeit

Diese Masterarbeit ist in fünf Teile gegliedert. Die Einleitung und der Appendix dienen insbesondere der übersichtlichen Strukturierung. Den Kern der Arbeit bilden aber die restlichen drei Bestandteile, diese sind in Abbildung 1.1 zu sehen.

Abbildung 1.1:
Visuelle Darstellung der drei Hauptbestandteile der vorliegenden Masterarbeit

Im Teil II werden die Methoden zur mobilen räumlichen Interaktion vorgestellt. In einem ersten Schritt werden Systeme vorgestellt, die zur Erfassung von Blicken dienen. Weiter werden ausgewählte mobile Anwendungen, die über annähernd blickbasierte Interaktionen gesteuert werden, vorgestellt. In diesem Teil erfolgt auch die inhaltliche Abgrenzung zu bestehenden Systemen und den entsprechenden wissenschaftlichen Arbeiten.

Der dritte Teil besteht aus einer prägnanten Ist–Analyse mit dem Ziel, die einzelnen Sensordaten und deren Zusammenhang zu erklären. Die bestehende Anwendung von Mosimann (2013) wird thematisiert. Anschliessend wird die service–orientierte Systemarchitektur des Prototyps und deren Umsetzung beschrieben.

Abschliessend wird im vierten Teil der erstellte Prototyp mit ausgewählten Versuchen getestet, mit dem Ziel, die bereits thematisierten Forschungsfragen zu beantworten. Die Ergebnisse werden hinsichtlich der Qualität der Sensoren und möglichen Verbesserungsvorschlägen besprochen. Der Teil wird durch einen Ausblick in Richtung künftiger wissenschaftlicher Arbeiten abgerundet.

[1] Mosimann (2013) entwickelte einen lokalen Algorithmus zur Verschneidung von Blickvektoren und 3D–Gebäuden. Zur Berechnung der Blickvektoren wurden georeferenzierte Eye Tracker Daten eingesetzt.

Teil II

Grundlagen und Stand der Technik

In diesem Teil werden die theoretischen Grundlagen vorgestellt, durch welche eine service–orientierte blickbasierte Interaktion mit 3D–Geoobjekten ermöglicht wird.

Kapitel 2

Grundlagen

Der nicht allzu kurze Titel dieser Masterarbeit enthält implizite Anspielungen auf aktuelle Forschungsbereiche und Technologien. In diesem Kapitel werden die Grundlagen zu den Begriffen der service–orientierten Systemarchitektur und der ortsbezogenen, blickbasierten Interaktion kurz erläutert.

2.1 Service Orientierte Architektur

Ein wichtiger Erfolgsfaktor eines Unternehmens ist, dass die wesentlichen Geschäftsprozesse optimal durch entsprechende Hilfsmittel aus dem Bereich der Informatik unterstützt werden. Die Erfahrung zeigte jedoch, dass oftmals die bestehenden Softwarelösungen zu wenig agil sind, um rasch auf die sich ändernden Geschäftsbedingungen zu reagieren (Krafzig et al., 2005, S. 28). Dies hatte damit zu tun, wie die IT–Umgebung innerhalb eines Unternehmens gewachsen ist. Ursprünglich wurden monolithische Applikationen gebaut, welche spezifische Aufgabenbereiche abdeckten. Später änderten sich die Anforderungen dahingehend, dass nun gesamte Geschäftsabläufe zu unterstützen waren. Dies führte zu neuen Herausforderungen bei der Integration der auf unterschiedlichen Technologien basierenden Anwendungen. Es wurden Konzepte von Punkt–zu–Punkt Schnittstellen bis hin zu einer Integration über sogenannte Middleware entwickelt (Schönherr, 2007, S. 11ff).
Die zunehmende Komplexität der Anwendungs– und Schnittstellenlandschaft führte dazu, dass die Wartbarkeit der IT–Systeme sehr zeit– und kostenintensiv wurde, weil die Interoperabilität zwischen den Systemen nicht gewährleistet war. Dies führte zu einer Evolution der grundlegenden Architekturprinzipien. Ähnlich wie bei den Programmiersprachen, als die Evolution vom ursprünglichen Maschinen Code über die prozedurale bis hin zur Objekt Orientierten Programmierung erfolgte. Was bei den Programmiersprachen die Stufe der objektorientierten Programmierung darstellt, ist bei Konzepten der *Enterprise Architecture* die „Service Orientierte Architektur“ (Elrad et al., 2001).

Die Service Orientierte Architektur (SOA) beschreibt in dem Sinne keine eigentliche techni-

sche Spezifikation, sondern stellt viel mehr das abstrakte Idealbild einer Systemarchitektur dar. Das Ziel dabei ist, eine äusserst flexible Unterstützung der Geschäftsprozesse zu gewährleisten, indem IT-Infrastruktur aus lose gekoppelten, wiederverwendbaren Diensten zusammengesetzt wird und so eine Komponenten–basierte Systementwicklung ermöglicht wird (Melzer, 2010). Einzelne separierbare Teilprozesse werden erkannt und gekapselt, so dass diese als Dienste angeboten werden können. Da die Dienste über standardisierte Schnittstellen verfügen, kann die gewünschte Gesamtfunktion einer Anwendung aus bereits bestehenden oder noch zu erstellenden Diensten aufgebaut werden. Eine weitere wichtige Eigenschaft dabei ist, dass Dienste auch während der Laufzeit einer Anwendung dynamisch eingebunden werden können. Aufgrund dieses Legoprinzips kann sehr schnell und kostengünstig auf die sich ändernden Geschäftsprozesse reagiert werden (Masak, 2007, S.195).

Eine SOA wird im wesentlichen durch drei Rollen charakterisiert. Der Dienstanbieter betreibt und wartet eine Plattform, um darauf Dienste zu veröffentlichen. Eine Veröffentlichung bedeutet in diesem Kontext, dass ein Dienst in einem lokalen Netzwerk oder dem Internet verfügbar und nutzbar gemacht wird. Damit ein potentieller Nutzer die gewünschten Dienste suchen und finden kann, muss ein Dienst immer auch in einem Dienstverzeichnis abgelegt werden. Zum besseren Verständnis kann ein solches Dienstverzeichnis mit den gelben Seiten eines Telefonbuchs verglichen werden. Der Dienstnutzer selektiert aufgrund spezifischer Auswahlkriterien den gewünschten Dienst, bevor er diesen aufruft und ausführt. Im Gegensatz zu einer klassischen Client–Server Architektur – wo der Funktionsumfang der Anwendung hardcodiert vorliegt – kann der Dienstnutzer während der Ausführung einer Anwendung weitere Dienste einbinden und nutzen (Melzer, 2010).

2.2 Web Services

Eine SOA könnte grundsätzlich mittels diverser Servicearten realisiert werden. In der Praxis geschieht dies aber in der Regel aber über sogenannte *Web Services*, für die bis dato unterschiedliche Definitionen existieren. Für einige sind es beispielsweise Programme die durch andere Anwendungen über das Internet ausgeführt werden können (Menasce & Almeida, 2002). Das World Wide Web Consortium (W3C) ging 2002 bei der Anforderungsbeschreibung an einen *Web Services* sehr viel weiter:

> *„A Web Service is a software application identified by a URI, whose interfaces and binding are capable of being defined, described and discovered by XML artifacts and supports direct interactions with other software applications using XML based messages via Internet-based protocols.“ (*W3C*, 2002)*

Diese Definition betont, dass *Web Services* über eine URI eindeutig identifizierbar sind. Darüber hinaus müssen die Schnittstellen und Einbindungsparameter der Web Dienste über Artefakte der Auszeichnungssprache XML definierbar, beschreibbar und auffindbar

sein. Dies ermöglicht es Applikationen, die über ein standardisiertes Internet Protokoll XML–Daten austauschen können, direkt mit den *Web Services* zu interagieren. Erst durch diese Definition wird klar wie potentielle Client–Anwendungen einen solchen Web Dienst einbinden können und dies insbesondere weil Web Service in der Lage sein müssen über sich selbst Auskunft zugeben.

2.2.1 SOAP/WSDL Web Services

Eine auf Web Services basierende Anwendung muss in der Lage sein, Nachrichten zwischen den beteiligten Komponenten auszutauschen. Dabei legt die SOAP Spezifikation lediglich fest wie eine Nachricht aufgebaut sein muss, um als *SOAP–Message* erkannt zu werden. SOAP–Nachrichten sind weder an ein bestimmtes Betriebssystem noch an eine bestimmte Programmiersprache gebunden. Sie bestehen aus einem XML–Wurzelelement namens *„Envelope“*, welches die eigentliche Nachricht im erforderlichen „SOAP–Body–Element“ enthält. Zusätzlich kann die *„Envelope“* mit dem optionalen Element *SOAP–Header* ergänzt werden (Melzer, 2010). Die Kombination von SOAP–Web Services und *Web Service Description Language (WSDL)* wurde so stark durch Unternehmen wie IBM, SAP oder Sun forciert, dass die jüngste *Web Service Definition* der W3C sehr stark darauf ausgerichtet wurde.

> *„A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“ (*W3C*, 2004)*

Web Service Description Language (WSDL) ist im Prinzip nichts anderes als eine Sammlung von *„Fachbegriffen“*, die auf Basis von XML–Artefakten erstellt wurde und gemäss obiger Definition zur Beschreibung von SOAP Web Services herangezogen wird. Eine Client–Anwendung kann die entsprechende WSDL–Datei lesen und interpretieren. Dadurch ist dem Client bekannt, welche Funktionen ein Service anbietet, was für Argumente zur Ausführung des Services mitzuliefern sind und welche Datentypen als Rückgabewerte erwartet werden können (W3C, 2001).

Im Zusammenhang mit SOAP Web Services muss hier kurz die Vision von *Universal Description, Discovery and Integration (UDDI)* erwähnt werden. Die Grundidee dahinter ist, ein standardisiertes Verzeichnis für Web Dienste zu schaffen, damit Clients die gewünschten Web Services finden und nutzen können. Wichtig zu erwähnen ist, dass dabei Verzeichniseinträge auch nur einem eingeschränkten Nutzerkreis zugänglich gemacht werden können. Die Spezifikation der dazu notwendigen Datenstrukturen wurde durch das Konsortium *„Organization for the Advancement of Structured Information Standards“* erstellt (OASIS, 2004). Der Umfang von UDDI und WSDL machen die Implementation von SOAP Web

Services komplex. Da diese im Verlaufe dieser Arbeit keine wesentliche Rolle mehr einnehmen werden, sei hier auf weiterführende Literatur verwiesen: wie die W3C– und die OASIS–Spezifikationen oder auf das Buch von Alonso et al. (2010).

2.2.2 RESTful Web Services

Das Internet ist aus unserem Alltag kaum mehr wegzudenken. Das Geheimnis dieses Erfolges beruhte grösstenteils auf der Einfachheit des Web's, die heutigen *SOAP–Web Services* sind jedoch alles andere als einfach. Dies weil *SOAP–Web Services* insbesondere darauf ausgerichtet sind, das Architekturprinzip der SOA umzusetzen. Dies führt aber dazu, dass bereits sehr einfache Web Dienste auf umfangreiche WSDL–Dateien angewiesen sind und deshalb auch als „grosse Web–Services" bezeichnet werden (Richardson & Ruby, 2007). Zudem ist es oftmals nicht notwendig sämtliche Möglichkeiten der SOA–Vision auszunutzen. Eine sehr gute und doch sehr mächtige Alternative wird in Form von RESTful–Web Services angeboten.

Die Bezeichnung REST (Representational State Transfer) wurde durch die Dissertation von Fielding (2000) geprägt, welche Beurteilungskriterien zur Bewertung von netzwerkbasierten Architekturen thematisiert. RESTful Web Services stellen über eine URL abrufbare *Ressourcen* zur Verfügung, welche einen eindeutigen Namen (*Uniform Resource Identifier* URI) besitzen müssen. Eine *Ressource* kann dabei alles sein was ein Client anfragt: Informationen, Referenzen auf weiteren Ressourcen, Objekte oder die Zeile einer Datenbanktabelle. Der Client kann nur mittels eines HTTP–Statements auf die gewünschte Ressource zugreifen (Richardson & Ruby, 2007). Die wichtigsten HTTP–Ausdrücke dazu sind: GET (Anforderung einer Ressource), POST (Aktualisierung einer Ressource), PUT (Erstellung einer neuen Ressource) und DELETE (zum Löschen einer Ressource).

2.2.3 OGC Web Services

Das *Open Geospatial Consortium (OGC)* wurde 1994 als gemeinnützige Organisation gegründet, mit dem Ziel offene Standards für Geoinformationssysteme zu entwickeln. Dem Gremium gehören Vertreter von Staaten, Universitäten und der Wirtschaft an. Das OGC definiert seinen Aufgabenbereich wie folgt:

> *„Definition einer Technologie, welche einem ... ermöglicht, jede Art von geocodierten Daten und Geofunktionalität ... zu nutzen... Angestrebt wird ein breiter Einsatz interoperabler SW-Komponenten von der Stange ..., die vollständige Integration der Geodatenverarbeitung ... und der Schritt von Geodaten zu Geoinformationsdiensten."*(giswiki, 2007)

Dies zeigt deutlich, dass sowohl die Geodatenformate wie auch die Geofunktionalitäten zwischen den unterschiedlichen GI–Systemen austauschbar werden sollen. Dabei stehen sicherlich die klassischen Integrationsmethoden im Vordergrund, vorausschauend wurde aber

bereits früh der Schritt Richtung *Web Services* angestrebt. Daher ist es wenig erstaunlich, dass die OGC seit 2007 ständiges Mitglied vom *World Wide Web Consortium* ist.

Für die Veröffentlichung von Geodaten im Internet wurden spezifische *Web Service Standards* entwickelt, welche auf dem Konzept der SOAP–Web Dienste aufbauten. Hier sei auf die Spezifikationen der OGC (2007) verwiesen und es werden lediglich zwei in dieser Arbeit verwendete Dienste namentlich erwähnt: *Web Map Service* und *Web Process Service*.

Ein *Web Map Service* dient dazu, eine georeferenzierte Karte als Dienst anzubieten. Die Karte kann in Form von Raster–, Vektordaten oder einer Kombination der beiden angeboten werden. WMS bieten per Definition drei Funktionen an, auf welche in Form einer HTTP–Anfrage zugegriffen werden kann. Die *GetCapabilities Anfrage* liefert Angaben über die Fähigkeiten, die Metadaten eines WMS wie beispielsweise die unterstützten Koordinatensysteme oder Angaben über die Datengrundlage. Um die Karte in einem Browserfenster auszugeben, muss die *GetMap Anfrage* ausgeführt werden. Optional kann ein WMS die Funktion *GetFeatureInfo Anfrage* anbieten, die es einem Client ermöglicht Informationen über den Wert eines Features abzufragen.

Der *Web Process Service* ist in der Lage Geofunktionen auszuführen oder räumliche Analyse von Geodaten durchzuführen. Dabei können WPS sowohl Raster– als auch Vektordaten verarbeiten. WPS bieten ebenfalls drei obligatorische Funktionen an: *GetCapabilities*, *DescribeProcess* und *Execute*. Die erstgenannte gibt Auskunft über die Metadaten des Services, *DescribeProcess* beschreibt den Zweck des WPS's und informiert darüber, welche Eingabedaten benötigt werden und welche Rückgabewerte ausgegeben werden. Damit die Geoprozessing Schritte ausgeführt werden, wird eine *Execute Anfrage* getätigt.

2.2.4 Local Based Services

Local Based Services sind Dienste, welche die geographische Lage eines Objektes berücksichtigen, um damit Informationen zu erlangen oder diese weiterzugeben. Die Positionsbestimmung erfolgt mittels diversen Technologien wie: GPS, Mobilfunk, WLAN–basierte Ortung oder auch über *Radio Frequency identification* (Rao & Minakakis, 2003). Junglas & Watson (2008) unterscheiden zwischen *location–tracking–* und *location–aware–Services*. Die erstgenannten Dienste zeichnen die Lage des zu verfolgenden Objektes auf und geben die Position an eine Überwachungsinstanz weiter. Diese Art von Diensten kommt häufig in Notfallzentralen zum Einsatz (Maglogiannis & Hadjiefthymiades, 2007), damit beispielsweise die aktuellen Positionen der Rettungsfahrzeuge in der Einsatzzentrale auf einer Karte dargestellt werden können. Dies ermöglicht die Gewährleistung einer raschen Koordination aller Fahrzeuge. Somit ist das Objekt, welches die eigene Lage ermittelt, der Sender der angeforderten Information. Im Gegensatz dazu verwenden die *location–*

aware–Services die eigene Lage dazu – um aufgrund ihrer relativen geographischen Lage zu anderen Objekten – eine georeferenzierte Informationen zu erhalten. Daher ist hier das positions–ermittelnde Objekt der Informationsempfänger (Junglas & Watson, 2008).

2.3 Eye Tracking

In der Psychologie, im Produktmarketing oder im Bereich des Softwaredesigns spielt die visuelle Wahrnehmung des Menschen eine zentrale Rolle. Dabei ist es von grossem Interesse, welche visuellen Stimuli unsere Achtsamkeit auf sich ziehen und welche nicht. Damit unter anderem diese spannende Fragestellung beantwortet werden konnte, mussten Messverfahren entwickelt werden, um die Blicke erforschen zu können. Die Verfahren zur Blickregistrierung sind auch unter der englischen Bezeichnung *„Eye Tracking“* bekannt.

Bei der Messung von Blickbewegungen unterscheidet Hofer & Mayerhofer (2010, S. 149ff) zwei Verfahrensarten: die objektiven und die subjektiven Methoden. Bei den subjektiven Methoden wird das Blickverhalten durch den Testleiter oder durch die Testperson selbst beschrieben. Diese direkte Beobachtung ist allerdings sehr ungenau. Aufgrund dieser Ungenauigkeit wurden bereits sehr früh objektive, apparative Methoden zur Messung der Blickbewegungen entwickelt. Karslake (1940) setzte bereits sehr früh Kameras ein, um die Wirkung und den Einfluss von Werbung auf Testpersonen zu erforschen.

Eine heute gängige Form zur kamerabasierten Aufzeichnung ist die *Cornea–Reflex–Methode*. Dabei wird ein Infrarot–Lichtstrahl auf das Auge projiziert, dieser reflektiert auf der Hornhaut (Cornea) und die so hervorgerufene Reflexion wird mit einer Augenkamera aufgezeichnet. Mittels der Veränderung der Reflexionen kann die Position des schärfsten Sehbereichs ausfindig gemacht werden und so die Blicke ermittelt werden. Oftmals wird nebst der Augenkamera noch eine weitere Kamera, zur Aufnahme des Blickfeldes der Testperson, eingesetzt. Die Information aus dem Augenkamera–Video kann genutzt werden, um auf dem Video des Blickfelds einen Blickcursor einzuzeichnen. Damit können die Augenbewegungen der Probanden stets leicht nachvollzogen werden (Büttner, 2009, S.31). Im Nachfolgenden ist mit dem Begriff *Eye Tracker* stets ein auf dem *Cornea–Reflex* basierende Vidoaufzeichnung gemeint.

Schlütz & Möhring (2013, S. 373) unterteilen die kamerabasierten Eye Tracker in die Kategorien der stationären und der mobilen Systeme (siehe Abb. 2.1 S.12). Die Eye Tracker der stationären Systeme befinden sich in der Regel auf einem Tisch, neben oder in einem Computerbildschirm montiert, und die Probanden sitzen unmittelbar davor. Die Blickcursor können direkt während oder erst nach der Aufnahme auf dem Bildschirm angezeigt werden. Die mobilen Systeme zeichnen sich hingegen dadurch aus, dass sie beispielsweise


Abbildung 2.1:
Auf dem linken Photo ist ein stationärer Eye Tracker der Firma LC Technologies Inc.[1] *abgebildet. Dabei repräsentiert die rote Fläche die Infrarot Lichtstrahlen und die gelbe Fläche die Blicke. Auf dem rechten Photo ist ein mobiler Eye Tracker mit Augen– und Blickfeldkamera der Firma Ergoneers GmbH*[2] *abgebildet.*

auf einem Helm oder einer Brille angebracht werden können und damit mobil einsetzbar sind (Büttner, 2009, S.31). Daher sind diese sehr viel flexibler und können selbst für Testanordnungen im Freien genutzt werden. Dabei wird der zur Datenaufnahme notwendige Computer in einem Rucksack mitgeführt (Schlütz & Möhring, 2013, S. 373).

Der seit Jahren anhaltende Trend, der Weiterentwicklung von mobilen Geräten, hielt auch bei den mobilen Eye Tracker Einzug. Obwohl es sich hier um ein Nischenprodukt handelt, gibt es nur schon in Europa mehrere Hersteller. Dazu zählen Ergoneers GmbH, Tobii Technology GmbH oder SMI GmbH, um nur einige namentlich zu erwähnen. Auch die Forschung leistet ihren Beitrag dazu, um mobile Eye Tracker Studien für einen breiteres Publikum attraktiver und erschwinglicher zu produzieren. Hierzu entwickelten beispielsweise Lukander et al. (2013) einen druckbaren und auf Open–Source basierenden, mobilen Eye Tracker mit Produktionskosten von lediglich 350 €.

In jüngster Zeit wurden mobile Eye Tracker mit weiteren Sensoren verbunden, um so neue Forschungs– und Anwendungsbereiche zu erschliessen. So koppelten Kiefer et al. (2012) einen mobilen Eye Tracker mit dem GPS-Sensor eines Smartphones und zeigten anhand einer damit durchgeführten Pilotstudie, dass nicht nur der individuelle Blick in die Studie einbezogen werden kann, sondern auch dessen Ausgangspunkt und die Geschwindigkeit der Testperson. Dies kann sehr aufschlussreich sein, um beispielsweise zu eruieren wie sich Personen mit einer Karte in einer neuen Umgebung orientieren.

Im Anschluss an die Blickregistrierung werden die Daten in meist zeitintensiven Verfahren

[1] http://www.eyegaze.com zuletzt besucht am 13.02.2014
[2] http://www.ergoneers.de zuletzt besucht am 13.02.2014

Abbildung 2.2:
Links: hypothetischer Blickverlauf mit Fixationen und Sakkaden. Rechts: Wirkungsanalyse eines deutschen Wahlplakates der CDU 2005 mit Blickverläufen und einer Heat–Map.

ausgewertet. Dabei ist es eine verbreitete Methode, die einzelnen Blickpunkte entsprechend ihrer Aufnahmereihenfolge mit Linien zu verbinden. Der so aufgezeichnete Blickverlauf kann in Bereiche von *Fixationen* und *Sakkaden* unterteilt werden. Zeitliche und räumliche Anhäufungen von Blicken werden als Fixationen bezeichnet. Es sind jene Momente, in welchen das Auge relative ruhig auf einem Bereich ruht. Nur während diesen Phasen ist der Mensch in der Lage seine Umwelt wahrzunehmen. Sakkaden hingegen sind Phasen von sehr schnellen und ruckartigen Augenbewegungen. (Hofer & Mayerhofer, 2010). Salvucci & Goldberg (2000) leisten in ihrem Artikel einen wertvollen Beitrag, um unterschiedliche Fixationsalgorithmen miteinander zu vergleichen. Dabei verglichen sie flächen-, geschwindigkeits– und verteilungsbasierte Fixationsalgorithmen miteinander und kamen zum Schluss, dass insbesondere die zwei letztgenannten gute Ergebnisse liefern.

Oftmals werden die Blickverläufe generalisiert indem für die Fixationen repräsentative Punkte, beziehungsweise Kreise eingesetzt werden. Dabei widerspiegelt die Kreisgrösse die Fixationsdauer. Die Fixationsdauer ist selten kürzer als 100 Millisekunden (ms) und liegt oftmals zwischen 200 – 400 ms (Salvucci & Goldberg, 2000). Eine weitere Analysemethode beinhaltet die Darstellung von Blickpunkten in Form von Heat-Maps. Dabei gilt, je rötlicher eine eingefärbte Fläche ist, desto mehr Blickpunkte enthält sie. Je grünlicher eine Fläche ist, desto geringer ist dort die Blickdichte. In Abbildung 2.2 ist ersichtlich, dass das Baby und der Text wahrgenommen werden, allerdings findet keine Verknüpfung zum Produkt der CDU statt und daher verfehlt es seinen Zweck.

Damit Blickverläufe vergleichbar wurden, mussten Methoden gefunden werden, um diese objektiv zu beschreiben. Goldberg & Helfman (2010) zeigten in ihrem Artikel diesbezüglich verschiedene Methoden auf, um Blickverläufe aufgrund ihrer Winkel, ihrer horizontalen oder vertikalen Ausrichtungstendenz zu klassifizieren. Oder Kiefer et al. (2013) nutzten charakteristische Augenbewegungen wie Blinzeln und Blickverläufe, um zu erkennen, ob eine Person etwas auf einer Karte sucht oder beispielsweise Objekte vergleicht.

2.4 Blickbasierte Interaktion mit 3D–Geoobjekten

Zur Steuerung von Computern werden traditionell Joysticks, Mäuse und Tastaturen verwendet. Alternative Steuerungen waren in der Vergangenheit nur in Randbereichen anzutreffen, beispielsweise um behinderten Menschen adäquate Kommunikationsmittel zur Verfügung zu stellen (Laffont et al., 2008). Erst in jüngster Zeit sind neue Technologien der Mensch–Maschine–Interaktion in unseren Wohnzimmern anzutreffen, wie Fernsehgeräte die durch Gesten gesteuert werden können (Juhlin & Önnevall, 2013).

Eine vielversprechende Vision ist es, eine blickbasierte Mensch–Maschine–Interaktion zu ermöglichen, dafür sprechen mehrere wesentliche Argumente (Vertegaal, 2008):

- Stehen die Hände für eine Steuerung aufgrund von anderen Tätigkeiten oder einer Behinderung nicht zur Verfügung, können Blicke unabhängig davon als Interaktionsmittel genutzt werden.
- Bevor ein Mensch eine manuelle Interaktion ausführt, wird das Zielobjekt fokussiert. Deshalb müssten blickbasierte Interaktionen, was die Geschwindigkeit betrifft, einer traditionellen handbasierten Steuerung überlegen sein.
- Der Mensch ist in der Lage unzählige Augenbewegungen zu tätigen, ohne deswegen sichtlich zu ermüden. Demzufolge könnten so monotone, sich wiederholende Handlungen vermieden werden und infolgedessen das Verletzungsrisiko durch einseitige Belastung reduziert werden.
- Das Einsetzten von Blicken muss nicht erlernt werden, sondern entspricht der natürlichen Veranlagung des Menschen und dient dazu gezielt Information aus dem Kontext heraus zu erkennen. Dies lässt sich an einem Gespräch veranschaulichen, bei welchem Informationen – nebst der verbalen Nachricht – auch über Gestik und Mimik gesendet oder empfangen werden können.

Wie sich zeigte bringen Blicksteuerungen gewisse Herausforderungen wie das *„Midas Touch“* Problem mit sich. Die Bezeichnung geht auf einen griechischen Mythos zurück der besagt, dass alles was König Midas mit seinen Händen berührte zu Gold wurde. Für eine blickbasierte Bedienung eines Computers bedeutet dies, dass der Benutzer sämtliche Buttons anklickt, welche er betrachtet und somit ungewollte Aktionen auslöst (Jacob, 1990).

Erfolgt eine Objektauswahl nur dann, wenn eine Fixation durch eine zusätzliche Aktion bestätigt wird, so kann das *„Midas Touch“* Problem vermindert werden. Die Bestätigung kann wie bei Jacob (1990) durch einen Tastenklick ausgelöst oder wie bei Miniotas et al. (2006) durch ein akustisches Kommando unterstützt werden. In der Anwendung von Baldauf et al. (2010) wird erst eine Aktion ausgelöst, sobald der Benutzer seine Augen für zwei Sekunden schliesst, danach wird der zuletzt berechnete Blickvektor verwendet.

Tanriverdi & Jacob (2000) zeigten einen Lösungsvorschlag anhand eines Histogramms, in welchem die aufsummierten Fixationen pro potentiellem Zielobjekt dargestellt wurden. Das Objekt mit den aktuell höchsten Diagrammwerten wird jeweils ausgewählt und entspricht demjenigen, an welchem der Nutzer das grösste Interesse bekundet. Diese Idee wurde von Park et al. (2008) aufgegriffen und mit einem Alterungsalgorithmus ergänzt. Dieser ordnet allen Objekten ein Alter zu, welches das Interesse des Betrachters an ihm repräsentiert. Jedes betrachtete Objekt altert, alle anderen hingegen werden jünger. Dementsprechend wird nur immer das älteste Objekt selektioniert.

Ein weiterer Ansatz ist, dass der Blick für eine bestimmte Verweildauer *(„dwell time")* auf dem Objekt ruht, um es zu selektieren (Jacob, 1990). Der Ansatz mit Verweildauer wurde bereits in diversen Bereichen untersucht, sei dies mit der Steuerung einer Bildschirmtastatur oder der Selektion von 3D–Objekten in einem virtuellen Raum. Zur Bedienung von blickbasierten Tastaturen wird üblicherweise eine Verweildauer von 600 - 1'000 Millisekunden gewählt (Majaranta & Räihä, 2002) (Miniotas et al., 2003). Zur Gebäudeselektion in einer urbanen Umgebung wählten Moniri & Müller (2012) eine Verweildauer zwischen 600 - 1'200 Millisekunden. Es ist allerdings zu erwähnen, dass der gesamte Zielvorgang mindestens sechs Sekunden dauerte.

Moniri & Müller (2012) entwickelten eine multimodale Anwendung die es ermöglicht, aus einem fahrenden Fahrzeug Informationen über ein gewünschtes Gebäude abzufragen. Dabei können die Gebäude über die Kopf– oder die Smartphoneausrichtung, das Sichtfeld oder den Blick selektioniert werden. Dazu musste das Fahrzeug mit einer WebCam, einem stationären Eye Tracker und einem GPS ausgestattet sein. Die WebCam zeichnet die gesamte Fahrt auf. Dieses Video wird verwendet, um daraus die genaue Fahrzeugposition zu bestimmen. Darüber hinaus stellten sie einen Scanning–Algorithums vor, der dazu diente, für den momentanen Fahrzeugstandort alle sichtbaren Gebäude zu ermitteln . Dies ermöglichte den Erhalt von Informationen über das betrachtete Gebäude

Baldauf et al. (2010) und Mosimann (2013) entwickelten jeweils eine Applikation, um in einem städtischen Umfeld mittels blickbasierter Interaktion Informationen über 3D–Objekte abzufragen. Beide Systeme wurden auf Basis eines mobilen Eye Trackers, einem Notebook und einem Smartphone aufgebaut. Die lokale Anwendung von Mosimann (2013) liess die Blickvektoren mit den 3D–Objekten schneiden und ermittelte dadurch die effektiven Blickschnittpunkte sowie die entsprechenden Gebäudenamen. Die Anwendung von Baldauf et al. (2010) basierte hingegen auf einer service–orientierten Architektur und sendete die Blickvektoren an einen Dienst der aufgrund eines 2.5D Gebäude–Modells und einer Sichtbarkeitsanalyse, die sichtbaren *Points of Interest* auflistet.

In der vorliegenden Arbeit soll insbesondere der Einfluss der gesamten Berechnungszeit zur Ermittlung des betrachteten Gebäudes und die dadurch hervorgerufene Verzögerung untersucht werden. In den soeben vorgestellten Arbeiten wurde dieser Aspekt nicht explizit berücksichtigt und auch die Berechnungszeiten wurden nicht ausgewiesen. Für eine kontinuierliche, blickbasierte Informationsabfrage ist dieser Faktor aber zwingend zu beachten. Erst dadurch kann eine stetige Systemausgabe in Echtzeit gewährleistet werden, ohne dass die aufgrund der Berechnungszeiten entstehenden Verzögerung kumuliert werden.

Wie gezeigt wurde, spielen Blick– oder Fixationspunkte in Wahrnehmungsanalysen eine zentrale Rolle. Daher soll der zu entwickelnde Prototyp die sichtbaren Schnittpunkte ausgeben, welche aus der Verschneidung von Blickvektor und Gebäude resultieren. Dieses Kriterium wurde einzig in der Arbeit von Mosimann (2013) berücksichtigt, allerdings ohne eine service–orientierte Systemarchitektur zu wählen. Dienst–basierte Systeme bringen jedoch entscheidende Vorteile, wie einfache Skalierung, die Nutzung durch mehrere Personen oder die Wiederverwendung des Dienstes mit sich.

Der Prototyp soll uneingeschränkt, handfrei und mobil einsetzbar sein und folglich aus einer minimalen Anzahl an Hardwarekomponenten bestehen. Deshalb wird auf eine Bestätigung – zur Auslösung der Berechnung einer Verschneidung – mittels Tastenklicks verzichtet. Somit werden kontinuierlich und für sämtliche Fixationen die Verschneidung berechnet.

Um das „Midas Touch Problem“ dennoch zu berücksichtigen, wurde ein neuer Ansatz gewählt. Eine Informationsausgabe an den Benutzer erfolgt nämlich nur, wenn eine einzelne Fixation oder mehrere Fixationen zusammen – bei identischem Verschneidungsergebnis – eine minimale Betrachtungsdauer überschreiten. Ein Vorschlag zur Festlegung dieser minimalen Betrachtungsdauer wird in Kapitel 8.6 auf Seite 77 erarbeitet. Dieser neue Ansatz bringt den Vorteil, dass ein Gebäude an mehreren unterschiedlichen Stellen betrachtet werden kann und der Benutzer dennoch eine Angabe über das Gebäude erhält. Im Gegensatz zur Arbeit von Baldauf et al. (2010), ermöglicht dies dem Benutzer stets die Augen offen zuhalten und sich somit sicher in einem urbanen Umfeld zubewegen.

Teil III

Prototyp Entwicklung

Im dritten Teil wird die eigenständig entwickelte Anwendung vorgestellt. Dabei werden die Zielsetzung der Entwicklung, die eingesetzten Materialien und die Systemarchitektur erläutert.

Kapitel 3

Analyse der Sensordaten

Bevor mit der Entwicklung der Anwendung begonnen werden kann, sind die Datenstrukturen der Ausgabedateien zu verstehen, welche der Eye Tracker und das Smartphone erstellen. In diesem Kapitel werden lediglich die in der Anwendung verwendeten Daten beschrieben.

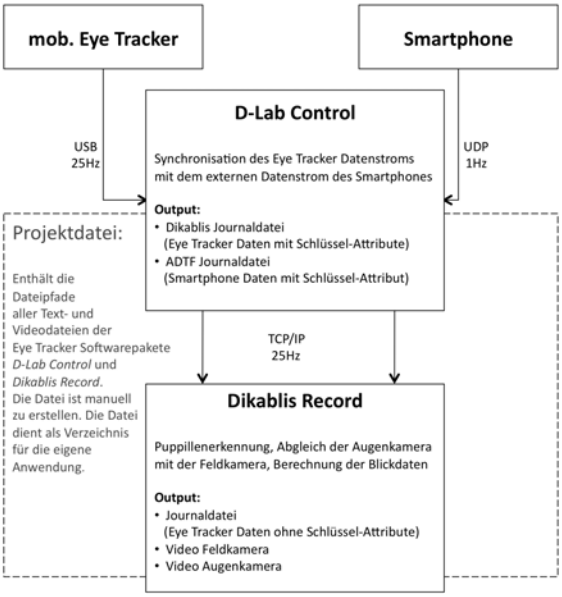

Abbildung 3.1:
Die Abbildung zeigt wie die Datenströme des mobilen Eye Trackers und des Smartphones – über die Softwarepakete „D-Lab Control“ und „Dikablis Record“– synchronisiert resp. gespeichert werden. Weiter ist ersichtlich, durch welche Softwarepakete die entsprechenden Dateien ausgegeben werden.

Die Abbildung 3.1 ist ein zentraler Bestandteil des Kapitels. Sie zeigt die Entstehungsquellen der Daten und veranschaulicht deren Zusammenhang. Da die Namen der Ausgabedaten nicht selbsterklärend sind und durch die Eye Tracker Software vorgegeben ist, ist es empfehlenswert diese Darstellung in Erinnerung zu behalten.

In der Abbildung 3.1 ist zu erkennen, dass sich die Aufnahmefrequenz des mobilen Eye Trackers von jener des Smartphones unterscheidet. Der Eye Tracker berechnet 25 Blicke pro Sekunde, das GPS hingegen nur eine Position pro Sekunde. *D-Lab Control* synchronisiert die Datenströme, indem es bei allen Blickdaten ohne GPS-Position jeweils die zuletzt gemessene Position einsetzt.

Hier muss noch angemerkt werden, dass die angegebenen Aufnahmefrequenzen theoretische Richtwerte sind. Der Eye Tracker müsste beispielsweise alle 40ms Blickdaten ausgeben, in Wirklichkeit geschieht dies jedoch nur durchschnittlich alle 38.5ms mit einer Standartabweichung von 0.998ms (Grob, 2012, S.20).

3.1 Struktur der Blickdaten

Wird der mobile Eye Tracker ohne Hinzunahme von externen Datenströmen verwendet, so kann die Datenaufnahme nur mittels der Software *Dikablis Record* erfolgen. Dieses Softwarepaket speichert die Videodaten der Augenkamera und der Blickfeldkamera zusammen mit der Journaldatei. Diese Datei enthält alle notwendigen Daten, um jegliche rein blickbasierte Analysen durchzuführen. Erst wenn nebst den Blickdaten zusätzliche Daten von externen Sensoren notwendig sind, ist *D-Lab Control* zu verwenden. Die zwei Softwarepakete führen zu geringfügig unterschiedlichen Ausgabedateien, im Folgenden werden diese Unterschiede genauer erläutert.

3.1.1 Datei erstellt durch *Dikablis Record*

Die ***Journal Datei***, die durch *Dikablis Record* erzeugt wird, beinhaltet alle notwendigen Informationen wie den Index, den Zeitstempel, die verwendeten Aufnahme- und Kalibriereinstellungen, die Position der Pupille im Video der Augenkamera sowie die entsprechende Position des Blickes im Video der Blickfeldkamera.

Tabelle 3.1:
Die Tabelle listet die zur Blickberechnung benötigten Werte der Journal Datei auf. Die Nummer referenziert auf die entsprechende Spaltenzahl. Die Werte der FieldX und FieldY sind in Pixel angegeben.

Nr.	Name	Werte	Bedeutung
2	timestamp	Zahl	Aufsteigend in Millisekunden, beginnt wenn Recording Computer gestartet wird
16	FieldX	$0 - 768$	Horizontale Pupillen Koordinate im Video der Feldkamera
17	FieldY	$0 - 576$	Vertikale Pupillen Koordinate im Video der Feldkamera

Die 3D-Blickvektoren können alleine aufgrund der Pupillenposition im Feldvideo und den optischen Kennwerten der Feldkamera berechnet werden (siehe Tabelle 3.1). Im Anhang A.1 und A.2 ist die Journal-Datei detailliert beschrieben, die Angaben stammen aus dem Dikablis Benutzerhandbuch (Ergoneers, 2011, S.33).

3.1.2 Datei erstellt durch *D-Lab Control*

Die ***Dikablis Journal Datei*** wird – entgegen der Vermutung des Dateinamens– vom Softwarepaket *D-Lab Control* erzeugt.

Die Datei enthält alle Daten der Journal-Datei, welche in Kapitel 3.1.1 vorgestellt wurde. *D-Lab Control* verändert jedoch den ursprünglichen Index, indem der Buchstaben J als Präfix angefügt wird. Zudem werden zwei zusätzliche Attribute erzeugt, welche Grob (2012, S.21) als Systemzeit und einen neuen Dikablis Journal Index identifizierte (siehe Tabelle 3.2).

Tabelle 3.2:
Die Tabelle zeigt exemplarisch die durch D-Lab Control *zusätzlich erzeugten Attribute. Diese werden bei der Verknüpfung der Daten des mobilen Eye Tracker's mit dem externen Datenstrom benötigt. Ein Beispiel einer Dikablis-Journal-Datei ist im Anhang A.3 beigefügt, die Dikablis Journal Datei verfügt leider über keine Spaltenbeschriftung.*

Dikablis Journal Systemzeit	**Dikablis Journal Index**	**J + Index der Journal Datei**	**weitere Journal Spalten**
63490987290303	2	J3	983291 ...

3.2 Struktur der Smartphone Daten

Das Smartphone selbst speichert lokal keine Daten, die Sensoren senden ihre Werte direkt an das Eye Tracker Softwarepaket *D-Lab Control*. Die Kommunikation zwischen den Komponenten erfolgt über eine Netzwerkschnittstelle mittels einem UDP-Protokoll (siehe Abb. 3.1 S. 18).

3.2.1 Datei der Smartphone-Sensorwerte erstellt durch *D-Lab Control*

Die durch *D-Lab Control* empfangenen Smartphonedaten werden in die sogenannte ***ADTF Journal Datei*** geschrieben. Dabei enthält die erste Spalte die Systemzeit und die zweite Spalte den Index. Beide entsprechen somit den ersten Spalten der *Dikablis Journal Datei*. Die dritte Spalte gibt Auskunft darüber, welcher Sensor für die Ausgabe der aktuellen Zeile verantwortlich war. Die nachfolgenden vier Spalten enthalten die spezifischen Sensorwerte (Mosimann, 2013, S.12).

63490987290265	0	gps_coord	8.50688003	47.4077400751	572.1999511719	5
63490987290267	1	gps_coord	8.50688003	47.4077400751	572.1999511719	5
63490987290303	2	gps_coord	8.50688003	47.4077400751	572.1999511719	5
63490987290343	3	gps_coord	8.50688003	47.4077400751	572.1999511719	5
63490987290383	4	gps_coord	8.50688003	47.4077400751	572.1999511719	5
63490987290423	5	gps_coord	8.50688003	47.4077400751	572.1999511719	5
63490987290463	6	orienta_angle	223.69351	28.060228	5.8632765	
63490987290503	7	orienta_angle	223.69351	28.060228	5.8632765	
63490987290543	8	orienta_angle	223.69351	28.060228	5.8632765	
63490987290583	9	orienta_angle	223.69351	28.060228	5.8632765	
63490987290623	10	orienta_angle	223.69351	28.060228	5.8632765	
63490987290663	11	orienta_angle	223.69351	28.060228	5.8632765	
63490987290703	12	orienta_angle	223.69351	28.060228	5.8632765	
63490987290743	13	orienta_angle	223.88298	28.128971	5.8806987	

Abbildung 3.2:
Die Zahlenreihen entsprechen einem Auszug aus einer ADTF Journal Datei. In den Zeilen sind die Werte vom GPS-, sowie die des Orientierungssensors ablesbar.

Gemäss Abbildung 3.2 gibt das GPS die Längen-/ Breitengrade, die Höhe und die GPS-Genauigkeit in Metern aus. Es ist der einzige Sensor, welche alle vier erwähnten Spalten zur Ausgabe benötigt. Der Orientierungssensor gibt das Azimut, den Höhenwinkel und den Rollwinkel (Drehung um die Längsachse des Smartphones) in Grad aus. Der Magnetsensor hingegen gibt die Angaben über die Magnetfeldstärke in X, Y und Z Richtung aus, die Masseinheit ist Mikrotesla (Mosimann, 2013, S.12).

3.2.2 Projektdatei

Die Projektdatei ist eine reine Hilfsdatei und wird manuell erstellt. Sie beinhaltet alle Pfade der drei Journal Dateien sowie die des Videos der Feldkamera. Gäbe es dieses Hilfsmittel nicht, so müsste jede einzelne Datei beim Programmstart ausgewählt werden. Durch die Verwendung dieser einfach zu erstellenden Datei ist nur noch ein Pfad zu wählen.
Bei der Erstellung der Projektdatei gilt es die Reihenfolge der Pfade zu beachten, damit die Anwendung die Informationen korrekt einliest. Die ADTF-Datei ist als erstes aufzulisten, danach folgen die Pfade der Dikablis Datei sowie der Journaldatei und zum Schluss noch den Pfad zum Video.

3.3 Verknüpfung der Sensordaten

Bei der Themenausarbeitung dieser Arbeit ging man davon aus, dass die Eye Tracker Software (*D-Lab Control*) einen synchronisierten Datenstrom ausgibt, welcher durch eine eigene Applikation empfangen werden kann. Während der Entwicklung stellte sich jedoch heraus, dass eine solche Schnittstelle nicht existiert. Es wurde vergebens versucht, die in Abbildung 3.1 dargestellte TCP/IP-Schnittstelle anzuzapfen. Selbst nach Rücksprache mit dem Eye Tracker Hersteller konnte keine Lösung gefunden werden. Damit die Fragestellung dennoch beantwortet werden konnte, war es notwendig, die Daten in einem ersten Schritt zu verknüpfen und in einer einzigen Datei zu speichern.

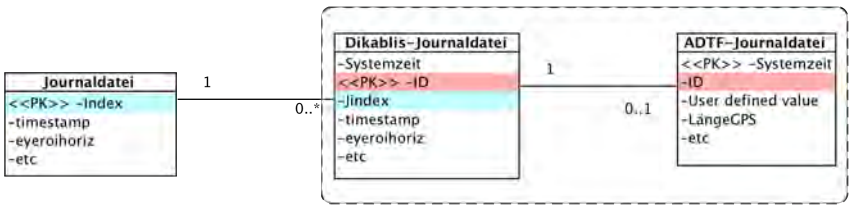

Abbildung 3.3:
Reduziertes Klassendiagramm zur Visualisierung der Beziehungen der einzelnen Journal Dateien. Die Verknüpfung zwischen den Blickdaten und den GPS-Daten erfolgt über die Dikablis Journaldatei.

Zur Verknüpfung der Daten konnte auf die Arbeit von Grob (2012, S.25) zurückgegriffen werden. In der Abbildung 3.3 ist das Prinzip der Relationen zwischen den Aufnahmedatensätzen ersichtlich. Die Primärschlüssel sind in den jeweiligen Klassen farbig hinterlegt. Einerseits beinhalten die Klassen innerhalb des gestrichelten Rahmens alle Informationen, um die Blickdaten mit den GPS-Daten zu verknüpfen. Anderseits wurde ihre Aufnahmefrequenz durch das Softwarepaket *D-Lab Control* auf annähernd 25Hz geregelt.

Die Journal Datei hingegen wurde mit einer unregelmässigeren Frequenz aufgenommen, dies ist durch die Kardinalität zur Dikablis-Journaldatei dargestellt. Falls mehr als ein Blick pro 40ms aufgenommen wurde, erscheint jeweils nur einer davon. Konnte innert 40ms kein Blick aufgenommen werden, wird der zuletzt berechnete Blick ein bis mehrere Male verwendet. Die ADTF-Datei weist ebenfalls nicht immer einen entsprechenden Wert auf, nämlich dann, wenn kein Sensor-Signal empfangen wurde (Grob, 2012, S.26).

Technisch gesehen erfolgt die Verknüpfung der Dateien über eine sogenannte *„Hashmap“*. Eine *„Hashmap“* dient dazu, einen Schlüssel (ID) mit genau einem Wert (Objekt) abzuspeichern. Die Sensordaten werden in der *„Hashmap (adtfHsmp)“* mit der ID als Schlüssel gespeichert. Anschliessend wird eine neue *„Hashmap (adtfZeileMitDikKeyHSMP)“* angelegt, mit dem Jindex als Schlüssel – wobei der Buchstabe J zu entfernen ist – und es werden erneut die Smartphonedaten als Werte abgelegt.

Als Letztes wird die Journal Datei eingelesen. Falls für den Index der Zeile ein Wert in der *„Hashmap“(adtfZeileMitDikKeyHSMP)* vorhanden ist, werden die Blickdaten und Sensordaten in einer 2D Array Liste gespeichert. Im Gegensatz zu den Arbeiten von Grob (2012) und Mosimann (2013), wurde die 2D Array Liste mit dem Attribut „waitMsec“ ergänzt. Dieses beinhaltet die effektiv verstrichene Zeit zwischen den Blickaufnahmen in Millisekunden. Die Verwendung des Attributes wird in Kapitel 4.4 abgehandelt.

Kapitel 4

Systemarchitektur

Bei Softwareentwicklung kann die Gliederung in funktionale und nicht funktionale Anforderung sehr hilfreich sein, um die geeignete Systemarchitektur zu entwerfen oder die Ablaufprozesse detailliert zu planen.

Klassisch gesehen ist die Entwerfung der Systemarchitektur nicht Bestandteil des Anforderungsmanagements. Paech et al. (2002) argumentieren in ihrem Position-Paper jedoch, dass eine integrale, einer isolierten Betrachtungsweise vorzuziehen ist. Mit anderen Worten sind also sowohl die funktionalen als auch die nicht–funktionalen Anforderungen und die Systemarchitektur gleichwertig bei der Systementwicklung zu berücksichtigen.

Dieser Ansatz wird auch zur Entwerfung von Applikationen mit starkem GIS-Bezug eingesetzt, so wie dies beispielsweise auch Hangel et al. (2012) beim Systementwurf zur Geovisualisierung von Controlling Daten taten. In diesem Kapitel werden ebenfalls zuerst die definierten Anforderungen aufgelistet und darauf basierend die Systemarchitektur und die Aufnahmeprozesse vorgestellt.

4.1 Anforderungen an die Applikationsentwicklung

Funktionale Anforderungen definieren was die geplante Anwendung können muss. Dabei werden zum Beispiel folgende Fragen beantwortet: Welche Daten gibt der Benutzer ein? Was macht die Anwendung damit? Welche Antworten erhält der Benutzer?

Die nicht–funktionalen Anforderungen sind Anforderungen an die Qualität, in welcher die gewünschten Funktionalitäten zu liefern sind. Dies sind beispielsweise Anforderungen an die Sicherheit, an die Performance oder die Zuverlässigkeit eines Systems. Ähnlich wie bei der Definition von Zielen, gilt es bei beiden Anforderungstypen zu beachten, dass nur messbare, respektiv eindeutig kontrollierbare Anforderungen ausgearbeitet werden.

Im Folgenden werden die Anforderungen aufgelistet, welche bei der Entwicklung der eigenen Anwendung berücksichtigt wurden. Einerseits liessen sich diese aus der Zielsetzung der Arbeit ableiten, anderseits wurden sie aber auch durch den Verfasser selbst oder in Rücksprache mit den Betreuern definiert.

Funktionale Anforderungen:

1. Benutzer interagiert nur über das GUI mit dem System
2. Das System erstellt aus den Sensordateien eine einzelne synchronisierte Datei
3. Aus den Sensordaten ist ein 3D-Blickvektor in Landeskoordinaten zu berechnen
4. Die Verschneidung zwischen Blickvektor und 3D-Stadtmodell ist zu berechnen
5. Die Verschneidungsergebnisse werden dynamisch im Kartenfenster dargestellt
6. Der Name des betrachteten Gebäudes ist mittels einem Label darzustellen
7. Die Karte zentriert sich automatisch auf die aktuelle GPS-Position des Benutzers
8. Anweisungen oder Ergebnisse werden im Ergebnisfenster dynamisch dargestellt

Nicht−Funktionale Anforderungen:

1. Serverantwortzeiten treffen nach max. 200ms ($\sim$ Realtime) ein
2. Vertikale Systemabweichung so gut wie bei Mosimann (2013, S.47)
3. Aufbau der Client-Anwendung aus Open-Source Bestandteilen
4. Die grafische Benutzerschnittstelle ist intuitiv bedienbar
5. Strikte Einhaltung des Model-View-Controll Entwurfsmusters
6. Der Client ist nebst den Sensordaten auf keine weiteren lokalen Daten angewiesen
7. Räumliche und a-räumliche Methoden sind getrennt zu implementieren

Hier ist noch anzufügen, dass ursprünglich die Intention bestand, das Gesamtsystem auf Open-Source-Komponenten aufzubauen. Es konnten zwar potentielle Komponenten wie die Java-Bibliotheken *„Geotools“* oder die Datenbank *„PostGIS“* gefunden werden, jedoch besitzen diese erst wenige 3D-Funktionen. So kennen *„Geotools“* und *„PostGIS“* zwar die Funktion 3D-Intersection, diese prüft jedoch nur ob ein Punkt oder eine Linie innerhalb eines Features liegt, gibt jedoch keine 3D-Schnittpunkte zurück.

Die Funktion, 3D-Intersection von *„PostGIS“* oder *„Geotools“*, hätte zwar ausgereicht, um das betrachtete Gebäude zu bestimmen, allerdings wäre dann ein Vergleich mit der Arbeit von Mosimann (2013, S.47) betreffend dem Genauigkeitsbereich nicht möglich gewesen. Dies weil Mosimann (2013, S.47) zur Bestimmung der Genauigkeit die Abweichung zwischen 3D-Schnittpunkt und 3D-Gebäude berechnete. Daher wurde zur 3D-Verschneidung ebenfalls auf die Werkzeuge von ArcGIS 3D-Analyst zurückgegriffen. Dies führte jedoch zur Anforderung, dass zumindest eine Systemkomponente ohne kostenpflichtige Werkzeuge zu entwickeln ist und daher die GIS-Methoden konsequent von den restlichen getrennt werden.

4.2 Ist–Analyse bestehender Anwendung

Bevor mit dem Entwurf der Systemarchitektur begonnen werden konnte, musste der bestehende – durch Mosimann (2013) entwickelte – lokale Algorithmus analysiert werden. Bei der Entwicklung der bestehenden Anwendung lag der Fokus darauf, die Eye Tracker Daten mit den Smartphone Sensoren zu synchronisieren und das Gesamtsystem so zu kalibrieren, dass daraus überhaupt ein 3D–Blickvektor erzeugt werden konnte. Daher ist es offensichtlich, dass diese Anwendung ein zentraler Bestandteil der vorliegenden Arbeit war. Im Folgenden werden Punkte identifiziert, die zwingend geändert werden mussten, um die an den Prototypen gestellten Anforderungen zu erfüllen.

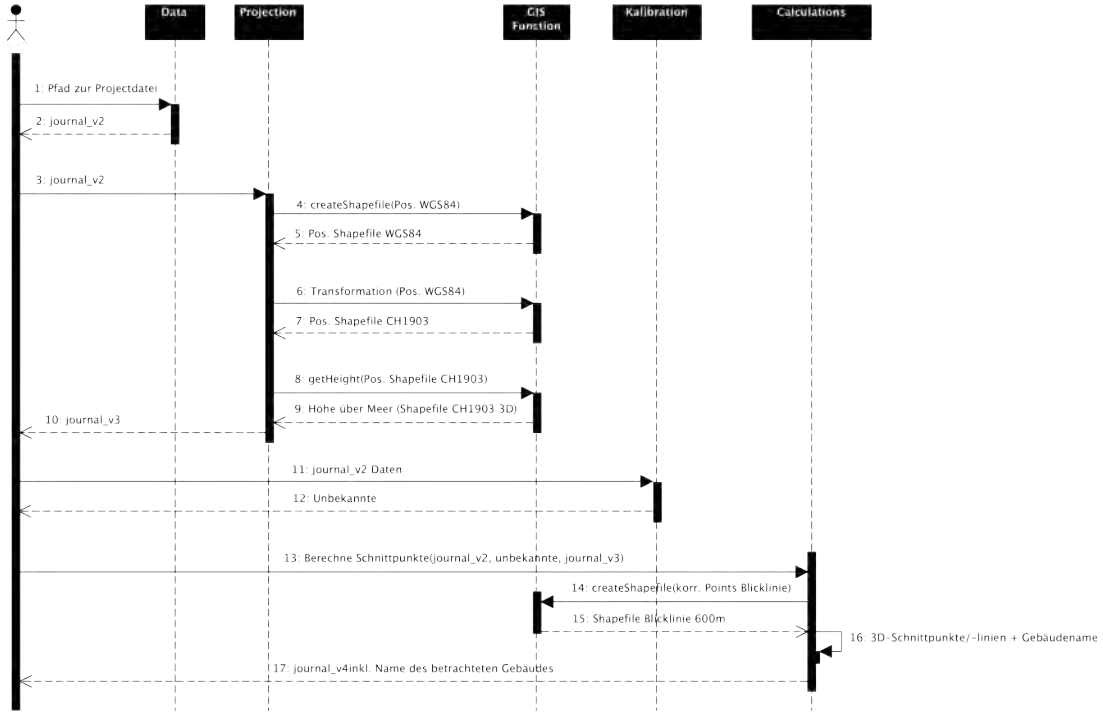

Abbildung 4.1:
Darstellung der Anwendung von Mosimann *(2013) in Form eines UML–Sequenzdiagramms*

Anhand des in Abbildung 4.1 gezeigten UML–Sequenzdiagramms lassen sich die kritischen Punkte sehr anschaulich aufzeigen. Als erstes gibt der Benutzer den Pfad zur Projektdatei an, dadurch kann die Anwendung sämtliche aufgezeichneten Blick– und Sensordaten einlesen, die entsprechenden Daten verknüpfen und in eine neue Textdatei namens *„journal_ v2“* schreiben (vgl. Abb. 4.1 Sequenzen 1, 2 und Kapitel 3).

Diese wird zur Berechnung der in Abbildung 4.1 dargestellten Sequenzen 4 bis 9 genutzt. Dabei wird als erstes die GPS–Position, basierend auf dem WGS84 Referenzsystem, ausgelesen und sogleich als Punkt–Shapefile gespeichert. Der soeben erstellte Punktdatensatz

wird ins nationale Koordinatensystem transformiert und erneut als Shapefile abgelegt. Der transformierte Datensatz wird verwendet, um die Höhenwerte aus dem digitalen Terrainmodell zu extrahieren, damit die durch das GPS ausgegebenen Höhen verbessert werden können (siehe Kapitel 6.1.1). Die so verbesserten Punkte werden anschliessend erneut als Shapefile gespeichert.

Zudem wird der Inhalt der *„journal_ v2“* Datei eingesetzt, um das Gesamtsystem von Eye Tracker und Smartphone zu kalibrieren (vgl. Sequenz 11, 12 und Kapitel 4.3.2). Erst danach kann aufgrund der in der Kalibration ermittelten Korrekturwerte (Unbekannte), der Smartphone– und der Eye Tracker Daten, ein Linien–Shapefile erzeugt werden, welches zur Verschneidung mit einem 3D–Gebäudemodell herangezogen wird (siehe Abb. 4.1 Sequenzen 14 bis 16).

Damit eine blickbasierte Mensch–Computer–Interaktion ermöglicht wird, sind folgende Punkte zu überarbeiten:

- Als erstes fällt auf, dass in den erläuterten Sequenzen jeweils sämtliche aufgezeichneten Blickdaten berücksichtigt werden und die Berechnung seriell erfolgt. Bei einer blickbasierten Steuerung einer Anwendung sind aber die Sequenzen für jeden einzelnen Blick vollständig durchzurechnen − und diese vorzugsweise als nebenläufige Prozesse − bevor mit dem nächsten Blick begonnen wird
- Weiter ist augenfällig, dass nach nahezu jedem Berechnungsschritt ein neues Shapefile geschrieben wird. Die damit verbundenen Speicher– und Lesezugriffe nehmen unnötig viel Berechnungszeit in Anspruch. Damit die in Kapitel 4.1 gestellten Anforderungen an die Performance erfüllt werden können, ist für den Prototypen eine Optimierung anzustreben, indem die Daten vorzugsweise nur im lokalen Arbeitsspeicher zwischengespeichert werden

4.3 Client-Server-Architektur und Aufnahmeprozesse

Für die Umsetzung der service-orientierten Systemarchitektur wurde ein Client-Server-Modell gewählt. Der Eye Tracker und das Smartphone sind direkt mit einem Notebook verbunden, die Software *D-Lab Control* synchronisiert die Daten und speichert diese lokal ab (siehe Kapitel 3). Dieser Vorgang müsste nicht zwingend auf dem Client stattfinden, jedoch müssen die Dateien dem Client zur Verfügung stehen.

Würde *D-Lab-Control* eine Schnittstelle mit den synchronisierten Sensordaten anbieten, so wären die Schritte über die Textdateien und den Aufnahmeprozess *„Create Datastream“* hinfällig (siehe Kapitel 3.3 S. 21). Dies weil der Datenstrom dadurch direkt von den Aufnahmeprozessen *„Start Calibration“* und *„Start Recording“* eingelesen werden könnte.

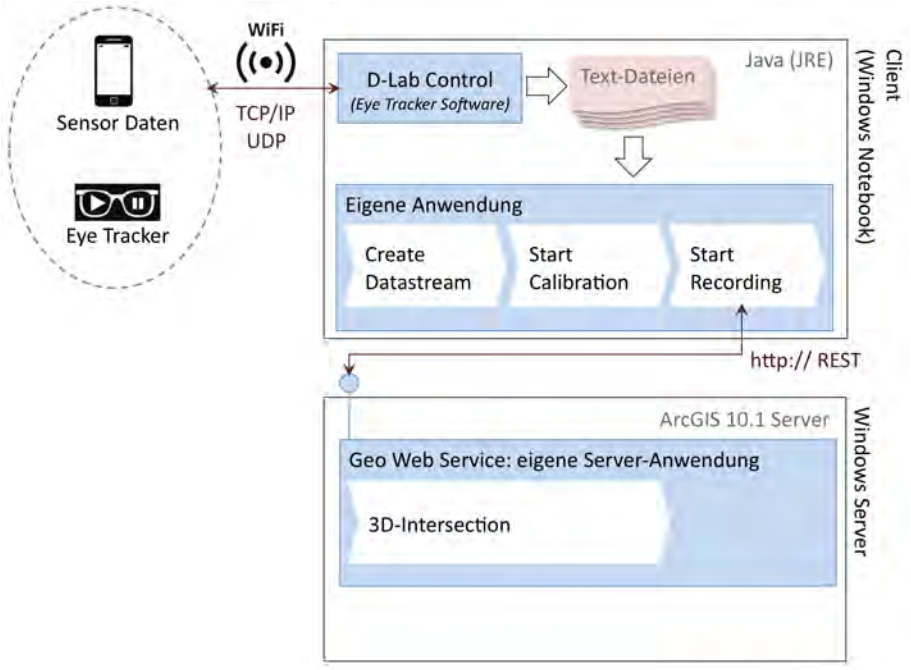

Abbildung 4.2:
In der Abbildung ist die gewählte Systemarchitektur mit den externen Sensoren, dem Client und dem Server ersichtlich. Die auf den Systemkomponenten laufenden Anwendungen sind jeweils mit hellblauer Farbe hinterlegt. Die weissen Pfeile stehen für die Aufnahmeprozesse.

Die Aufgaben zwischen Client und Server sind so verteilt, dass die datenintensiven sowie das Geoprocessing auf dem Server erfolgen. Aufgaben die unmittelbar nach der Datenaufnahme erfolgen können, werden schnellstmöglich auf dem Client berechnet.

Der durch Mosimann (2013) entwickelte lokale Algorithmus – zur Ermittlung des betrachteten Gebäudes – wurde in einem ersten Schritt analysiert und in grob zusammenhängende Aufgabenblöcke eingeteilt. Das Ziel dabei war es, einzelne Aufnahmeprozesse ausfindig zu machen, die vor der eigentlichen Blickerfassung durchgeführt werden können, um so eine zeitliche Entkoppelung zu gewährleisten (vgl. Abb. 4.2). Durch diese Vorgehensweise kann die Verzögerung beim Initiieren der Anwendung bereits reduziert werden, selbst wenn es sich dabei nur um wenige Millisekunden handelt. Im Folgenden werden die ermittelten Aufgabenblöcke beschrieben.

4.3.1 *Create Datastream*

In diesem Schritt werden die durch *Dikablis* und *D-Lab* erzeugten Textdateien gemäss Kapitel 3.3 verknüpft. Als Output resultiert eine „flache“ Textdatei, welche alle benötigten Attribute für den Aufnahmezeitpunkt in einer Zeile enthält. Die letzte Spalte „waitMsec“ enthält die Anzahl Millisekunden, die zwischen den einzelnen Blickaufzeichnungen ver-

strichen sind. Erst durch diese zusätzliche Information wird eine Simulation der Blickaufnahme in realer Aufnahmefrequenz möglich: Die Anwendung ruht für die angegebenen Millisekunden, bevor die nächste Zeile eingelesen wird und die weiteren Berechnungsmethoden aufgerufen werden. Das ist auch eine Voraussetzung, um eine der Kernfragen der Arbeit beantworten zu können, nämlich „wie mit der nicht zu vermeidenden Antwortzeit des erstellten Services umzugehen ist, wenn sich die Blicke sehr schnell bewegen".

4.3.2 *Start Calibration*

Sobald die Testperson mit dem Smartphone und dem Eye Tracker ausgestattet ist, kann das Gesamtsystem kalibriert werden. Das Smartphone ist der Testperson auf einem „Helm" aufgesetzt und der mobile Eye Tracker wird wie eine Brille getragen. Solange sich der Helm oder die Brille nicht verschieben, sind die Abweichung der optischen Achse der Feldkamera und die Ausrichtung des Smartphones relativ konstant zueinander. Sie unterscheiden sich lediglich durch einen unbekannten Roll-, Höhen- und Orientierungswinkel. Der Zweck dieses Aufnahmeschrittes ist es nun, genau diese Unbekannten zu ermitteln. Die während dem Aufnahmeprozess *„Create Datastream"* erzeugte Textdatei wird als Eingangsdatensatz benötigt, als Ausgabe liefert *„Start Calibration"* einen Array mit ermittelten Unbekannten. Die Formeln, welche zur Berechnung der Unbekannten benötigt werden, können in der Arbeit von Mosimann (2013, S. 15ff) nachgeschlagen werden. Voraussetzung zur Anwendung der Formeln ist, dass die Koordinaten des Standortes der Testperson sowie der betrachteten Stelle bekannt sind. Mosimann (2013, S. 21 u. 47) verwendete zur Kalibrierung des Gesamtsystems nur einen Datensatz, in der hier umgesetzten Anwendung werden jedoch die Ergebnisse von 250 Blicken gemittelt.

4.3.3 *Start Recording*

Sobald die Datenaufnahme gestartet wird, werden die 600m langen Blickvektoren aus den Smartphonedaten, den Pixelwerten und den optischen Kennwerten der Feldkamera berechnet. Da bereits vorgängig alle unbekannten Winkel berechnet wurden, können nun die Korrekturen direkt während der Berechnung erfolgen. Anschliessend werden die Blicke an den Server gesendet, wo diese weiterverarbeitet werden. Die Serverantwort wird ebenfalls durch diesen Prozess zur Darstellung an die zuständigen GUI-Komponenten weitergeleitet.

4.3.4 *3D-Intersection*

Auf dem Server selbst finden lediglich die Geoverarbeitungsschritte statt. Der 3D-Blickvektor wird mit dem 3D-Gebäudemodell verschnitten. Falls der Blickvektor auf ein Gebäude gerichtet ist, so dringt der Vektor ungehindert durch alle Aussenwände und es resultieren mehrere Schnittpunkte. Diese sind somit noch zu filtern, weil nur die Gebäudeinformation vom ersten Schnittpunkt benötigt wird.

4.4 Sequenzdiagramm des Prototyps

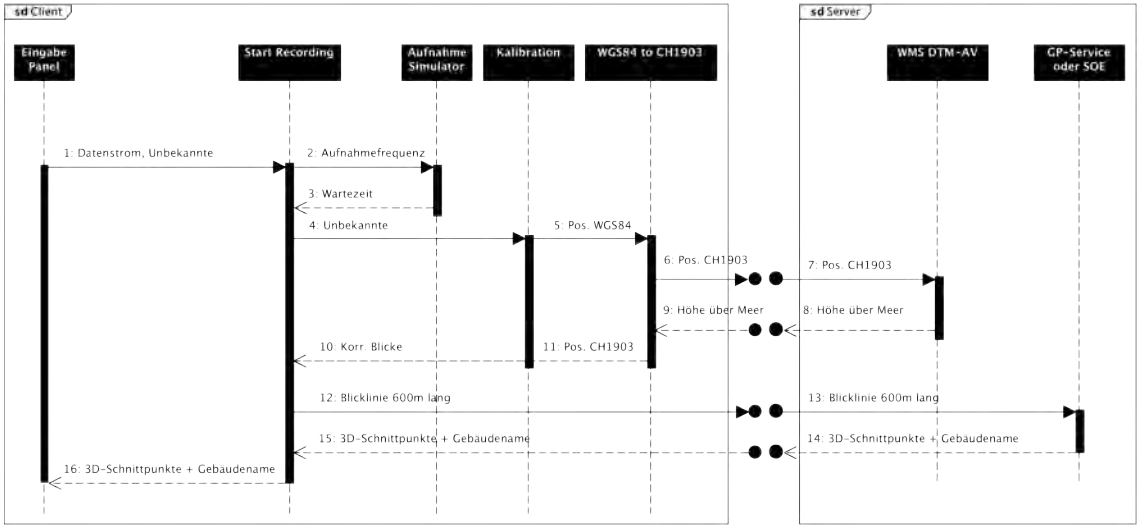

Abbildung 4.3:
In der Abbildung ist das UML–Sequenzdiagramm des zu entwickelnden Prototyps dargestellt

Wie sich die gewählte Systemarchitektur und die an den Prototypen gestellten Anforderungen auf das Zusammenspiel zwischen Client und Server auswirken, ist im Sequenzdiagramm der Abbildung 4.3 ersichtlich. Die in Kapitel 4.3.1 beschriebene „flache“ Datenstrom–Datei enthält sämtliche Blickdaten. Durch das Starten der Anwendung sollen die im Kalibrationsprozess ermittelten Unbekannten abgerufen werden, so dass die Blicke in Echtzeit korrigiert werden können. Aus dem Datenstrom soll jeweils nur eine einzelne Zeile ausgelesen werden, so dass sich beim Durchlaufen der Sequenzen 2 bis 16 stets nur ein einzelner Blick im Berechnungsprozess befindet. Der Aufnahmesimulator hat dabei lediglich die Aufgabe, die Blicke gemäss der tatsächlichen Aufnahmefrequenz in den Berechnungsprozess einzuspeisen.

Die Methoden der Klasse „Kalibration“ und „WGS84 to CH1903“ −zur Korrektur der Blickrichtung beziehungsweise zur Transformation der Koordinatensysteme − müssen gleichzeitig und somit nebenläufig ausgeführt werden können. Erst im Anschluss wird dann ein 600m langer Blickvektor berechnet, der zur Verschneidung mit dem 3D–Gebäudemodell, an den GIS–Server geschickt werden soll.

Im Sequenzdiagramm selbst ist es nicht ersichtlich, aber es sollen keine Daten auf die Festplatte geschrieben werden, sie müssen sich somit stets im Arbeitsspeicher des Clients befinden. Damit die an den Prototypen gestellten Anforderungen eingehalten werden können, ist die Transformation der Koordinatensysteme auf Basis der von der Swisstopo publizierten Formeln eigenständig zu implementieren (Swisstopo, 2008).

Kapitel 5

Verwendete Umgebung

5.1 Integrierte Entwicklungsumgebung

Eine integrierte Entwicklungsumgebung (IDE) beinhaltet alle notwendigen Werkzeuge, die zur Softwareentwicklung benötigt werden. Namentlich sind dies: Ein Texteditor für die Eingabe des durch den Menschen lesbaren Codes, ein Compiler zur Umwandlung des menschenlesbaren Codes in einen maschinenlesbaren Code und ein Debugger zur Fehlerbehebung während der Entwicklung. Für die Entwicklung der Anwendung wurde Eclipse Java EE der Version Juno Service Release 2 als IDE verwendet. Eclipse wurde als Open-Source Projekt entwickelt. Der Kern der IDE kann durch diverse kostenlose und kostenpflichtige Erweiterungen ergänzt werden. Ein weiterer Vorteil ist, dass Eclipse über eine automatische Quelltextformatierung sowie eine Unterstützung zur einfachen Erstellung von grafischen Oberflächen (GUI) verfügt.

5.2 Verwendete Programmiersprache

Die Realisierung der eigenen Anwendung wäre mittels diversen Programmiersprachen wie Python, .Net oder C# möglich gewesen. Die Applikationen von Grob (2012) und Mosimann (2013) – auf welchen diese Arbeit aufbaut– wurden jedoch bereits beide in Java realisiert. Deshalb war es naheliegend, ebenfalls mit Java weiterzufahren und somit wurde Java Development Kit 1.7.0 der 32-bit Version zur Entwicklung benutzt.

5.3 Serverumgebung

Eine serviceorientierte Systemarchitektur ist auf einen Server zur Veröffentlichung von Diensten angewiesen. Dazu wurde ein Windows 7 64-bit Server eingesetzt, mit zwei Intel(R) Xenon(R) CPU E5620 @ 2.40 GHZ Prozessoren, 8.00 GB RAM und bereits installiertem Service Pack 1. Darauf wurde ArcGIS for Server 10.1 installiert, welcher unter anderem der Veröffentlichung von Karten- oder Geoprocessing-Diensten dient.

Kapitel 6

Aufbau Server-Anwendung

Auf dem GIS-Server werden nicht nur die Geoverarbeitungsdienste, sondern sämtliche zur Berechnung benötigten Geodaten veröffentlicht. Der Aufbau der verwendeten Dienste und deren Anwendungszweck wird im Folgenden erläutert.

6.1 Web Map Service

„Web Map Services (WMS)“ dienen dazu, räumliche Informationen übers Internet zu veröffentlichen und so Landkarten einem breiten Publikum zugänglich zu machen. WMS-Dienste können über herkömmliche Browser oder GIS-Software geöffnet und betrachtet werden. Gemäss *„Open Geospatial Consortium (OGC)“* Spezifikation bietet der Service folgende Funktionen an:

- ***GetCapabilities:*** Diese Abfrage liefert die Beschreibung und die Anwendungszwecke des Kartendienstes.
- ***GetFeatureInfo:*** Aufgrund einer Positionsangabe liefert die Abfrage eine Layerinformation des Kartendienstes.
- ***GetMap:*** Dieser Befehl ruft die angebotene Karte auf und stellt diese im entsprechenden WMS-Client dar.

Einige Vorteile von WMS-Diensten sind, dass das Kartenmaterial nicht mehr lokal auf dem Client gespeichert werden muss, sondern zentral auf einem Server verwaltet werden kann, was die Ressourcen des Clients schont. Durch die Verwendung von Kartendiensten wird der Client –abgesehen von der GPS-Position – unabhängig von zusätzlichen Geodaten.

6.1.1 Digitales Terrain Model

Die Smartphonesensoren wurden bereits von Mosimann (2013, S. 47) als Schwachstellen erkannt. Für den GPS-Sensor kann diese Aussage noch etwas differenziert werden. Der Genauigkeitsbereich der X–/Y– Koordinatenausgaben beträgt rund 5m. Für die Höhenausgabe ist dieser Bereich wesentlich ungenauer und die Abweichungen betrugen während

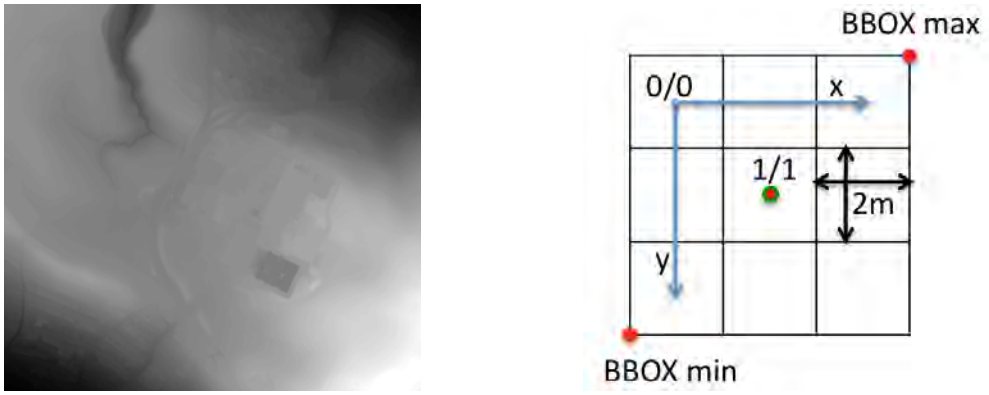

Abbildung 6.1:
Im linken Bild ist ein Auszug des verwendeten DTM abgebildet, die Pixelwerte entsprechen den Geländehöhen. Die rechte Darstellung dient zum besseren Verständnis der „getFeature-Info-Anfrage“. Das abgebildete Quadrat entspricht der „Bounding Box“, welche durch die roten Koordinaten aufgespannt wird. Der Ursprung und die positiven Achsen des Pixel-Koordinatensystems sind in hellblauer Farbe eingezeichnet. Die Position des abgefragten Pixelwertes ist mit einem rot-grünen Punkt markiert.

den Feldtests bis zu 50m, was die Aussage von Mosimann (2013, S. 21) bestätigt. Deshalb wurden anstelle der GPS-Höhenausgabe, die Höhen aus dem DTM-Kartendienst verwendet. Als Grundlage für den DTM-Kartendienst wurde das Höhenmodell *SwissALTI3D* des Bundesamtes für Landestopographie verwendet. Der Rasterdatensatz *SwissALTI3D* lag mit einer Auflösung von 2m und einer Genauigkeit von 0.5m vor. Damit die Client-Anwendung die genauen Höhen erhält, greift sie mittels einer *GetFeatureInfo-Anfrage* auf den Kartendienst zu und erhält als Antwort die Höhe der entsprechenden GPS-Position. Im Folgenden ist die durch den Client verwendete Anfrage aufgeführt:

”http://...WMSServer?SERVICE=WMS&VERSION=1.3.0&REQUEST=GetFeatureInfo&CRS=EPSG:21781...
*...&BBOX=” + **bBOXymin** + ”,” + **bBOXxmin** + ”,” + **bBOXymax** + ”,” + **bBOXxmax** + ...*
*...”&WIDTH=3 &HEIGHT=3&QUERY_LAYERS=0&**X=1**&**Y=1**&SERVICENAME=DTM_WMS ...*
...&INFO_FORMAT=text/plain&”

Die Abfrage besteht aus obligatorischen und optionalen Parametern, die durch den Parameternamen und den zugehörigen Wertebereich definiert werden. Die einzelnen Parameter sind durch das Zeichen „*&*“ voneinander getrennt. Soll die Anwendung beispielsweise den Kartendienst im nationalen Koordinatensystem CH1903 (EPSG Code:21781) aufrufen, so muss dem Parameter namens *CRS* der Wert *EPSG:21781* zugeordnet werden. Die genauen Spezifikationen können auf der Website des *„Open Geospatial Consortium (OGC)“*[1] nachgeschlagen werden, im Folgenden wird nur der essentielle Teil der Anfrage erklärt.

[1] http://www.opengeospatial.org/ zuletzt aufgerufen am 24.11.2013

Der Parameter *BBOX* beschreibt ein umhüllendes Rechteck „*Bounding Box*“, welches anhand der Koordinaten der unteren linken und der oberen rechten Ecke aufgespannt wird (siehe Abb. 6.1). Bei einer *getMap-Anfrage* begrenzt die *BBOX* den zurückgegeben Kartenausschnitt. Bei einer *getFeatureInfo-Anfrage* können nur Werte innerhalb dieses Bereichs abgefragt werden. Zur Formulierung der Anfrage müssen die Eck-Koordinaten im entsprechenden Referenzsystem angegeben werden. Die durch die *Bounding Box* aufgespannte Fläche wird mittels Pixelkoordinatensystem beschrieben und die Auflösung entspricht derjenigen des abzufragenden *Features*. Der Ursprung dieses Referenzsystems liegt in der Mitte des oberen linken Pixels (siehe Abb. 6.1). Gemäss der abgebildeten *getFeatureInfo-Anfrage* auf Seite 32 wird der Wert des Pixels mit den Koordinaten X=1 und Y=1 abgefragt.

Im Client wird die *getFeatureInfo-Anfrage* dynamisch – basierend auf der aktuellen GPS-Position – erzeugt. Die *Bounding Box* wurde deshalb sehr kleinflächig gewählt, damit nur der minimal notwendige Kartenausschnitt geladen wird. Die GPS-Position muss zum Abfragezeitpunkt bereits in schweizerischen Landeskoordinaten vorliegen. Somit können nun aufgrund der GPS-Position die Koordinaten der *Bounding Box (bBOXmin und bBOXmax)* berechnet werden, indem die Position jeweils in der horizontalen und vertikalen um +3m, respektiv −3m verschoben wird. Dies ist notwendig, da die Rasterauflösung des DTM's 2m beträgt und die GPS-Position in der Mitte des zentralen Pixels liegt (siehe Abb. 6.1).

6.1.2 3D–Gebäudemodell

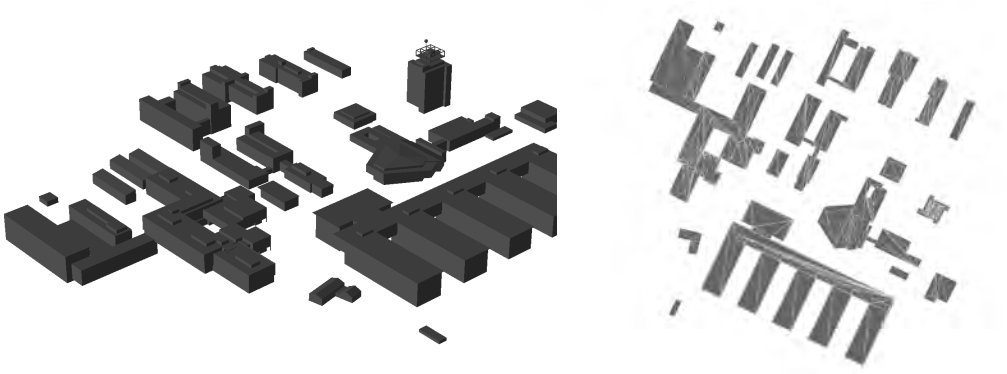

Abbildung 6.2:
Links ist eine 3D Visualisierung des verwendeten Stadtmodells ersichtlich. Rechts ist der veröffentlichte WMS in 2D dargestellt, weil die üblichen WMS-Clients nicht 3D fähig sind.

Eine Kernanforderung der Anwendung besteht darin, den Benutzer über das momentan betrachtete Gebäude zu informieren. Dies wird durch die Verwendung eines 3D-Gebäudemodells möglich. Das Gebäudemodell wurde durch das Institut für Kartographie und Geoinformation zur Verfügung gestellt. Der Geodatensatz basiert auf einem Multipatch-Feature und besitzt lediglich die ID, die Geometrie und die Abkürzung des Gebäudenamens als Attri-

bute. Das Stadtmodell wurde ebenfalls auf dem GIS-Server als Kartendienst veröffentlicht (siehe Abbildung 6.2).

6.2 Geoprocessing-Service

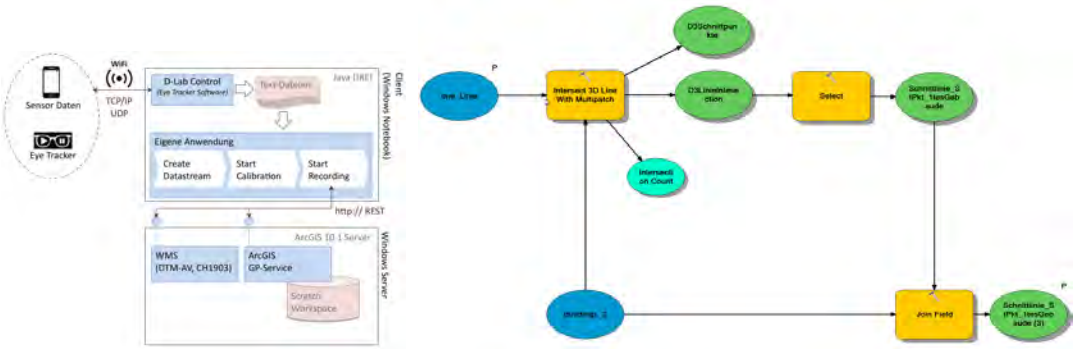

Abbildung 6.3:
In der linken Darstellung ist die Systemarchitektur abgebildet, die hier bei der Verwendung des Geoprocessing-Services eingesetzt wurde. Rechts ist der erstellte Geoprocessing-Service schematisch dargestellt, dieser wurde so auf dem GIS-Server publiziert.

In ArcGIS 10.1 Desktop lassen sich mit dem Model-Builder sehr einfach Geoprocessing-Workflows erstellen, die danach per Knopfdruck ausführbar sind. In der Abbildung 6.3 ist das erstellte 3D–Verschneidungsmodell abgebildet. Die blauen Ellipsen entsprechen den Input-Datensätzen, die orangen Vierecke stellen Geoverabeitungs–Werkzeuge dar, während die grünen Ellipsen für die Output–Datensätze stehen.

Als Input-Datensatz wird eine 3D-Blicklinie und das 3D-Stadtmodell verwendet. Das durch die GIS-Software zur Verfügung gestellte Werkzeug „Intersect 3D-Line with Multipatch“ ist in der Lage die geforderte 3D-Verschneidung der Input-Daten durchzuführen. Als Ergebnis werden drei Ausgaben erzeugt: die Anzahl Schnittpunkte, die realen 3D-Schnittpunkte und die 3D-Liniensegmente. Die zwei zuletzt genannten Ergebnisse werden in Form von *Shapefiles* ausgegeben. Die Liniensegmente entstehen dadurch, dass die Input-3D-Linie an den berechneten Schnittpunkten unterteilt wird. Schneidet beispielsweise eine Linie zwei Gebäudewände, dann resultieren daraus zwei Schnittpunkte und drei Liniensegmente.

Ein Schnittpunkt besitzt die Information des 3D-Objekts, welches vom Blickvektor geschnitten wurde. Die Information liegt in Form einer Gebäude-ID und einer Blicklinien-ID vor. An einem Liniensegment hingegen ist ersichtlich, bei welchem Gebäude sie beginnt und bei welchem sie endet. Diese Angaben liegen ebenfalls als Gebäude-ID's vor und haben grundsätzlich ein positives Vorzeichen. Enthält ein Segment allerdings eine negative Gebäude-ID von „-1“ so bedeutet dies, dass das Segment nicht bei einem Gebäude begann

oder endete. Diese Eigenschaft wurde genutzt, um das vom Benutzer zuerst betrachtete Gebäude zu selektieren, da ein solches Liniensegment stets bei einem „Gebäude“ mit einer negativen ID beginnt, sprich beim Auge des Benutzers (siehe Abb. 6.3 *Select-Abfrage*).

Aufgrund der soeben beschriebenen Selektion ist nun das Liniensegment bekannt, welches zum betrachteten Gebäude führt. Um auch den Gebäudenamen zu identifizieren, ist es erforderlich, eine Verbindung *„Join“* zwischen den Attributen des Stadtmodells und dem selektierten Segment herzustellen. Die Verknüpfung erfolgt mittels der Gebäude-ID und ist im letzten Schritt der Abbildung 6.3 schematisch dargestellt.

Die Vorteile der Verwendung des Model–Builder ist einerseits, dass sich Workflow's sehr schnell erstellen lassen und darüber hinaus sehr einfach auf dem GIS–Server als GP–Service veröffentlicht lassen. Durch die Verwendung des Geoprocessing-Frameworks können zudem die Benutzereingabe validiert werden und vordefinierte Fehlerausgaben eingesetzt werden. Dies sind sehr nützliche Hilfen, um den Entwickler bei der Fehlerfindung zu unterstützen. Zur Speicherung von intermediären Ausgabedateien verwendet der GP-Service einen sogenannten *„Scratch-Workspace“*. Dieser Workspace dient jedoch nicht zur Archivierung von Ausgabedaten, sondern nur zur Speicherung von Daten mit temporärem Charakter.

In der Abbildung 6.3 ist links die erstellte Systemarchitektur ersichtlich. Es ist erkennbar, dass auf dem GIS-Server das *Digitale Gelände Modell* als WMS, sowie der GP-Service veröffentlicht wurden. Auffällig ist, dass das 3D-Stadtmodell nicht als WMS publiziert wurde. Dies ist so, weil das 3D-Stadtmodell als Input-Datensatz im Geoprocessing-Workflow enthalten ist und daher bereits auf dem Server gespeichert wurde.

Die Client Anwendung kommuniziert mit dem erstellten GP-Service durch Methoden der *Representational State Transfer Spezifikationen (REST)*, welche durch die Dissertation von Fielding (2000) geprägt wurde. Dabei wird die Blicklinie in Form eines *JSON-Objekt's* zwischen den Anwendungen transferiert. Die Server-Antwort wird dem Client ebenfalls als *JSON-Objekt* geliefert. Im Anhang A.4 ist eine Blicklinie als JSON-Objekt beigefügt, diese wird in einem *REST-Request* an den Server gesendet.

Die vorgängig geschilderten Vorteile des GP-Framework's führen dazu, dass die Geoverarbeitungsdienste einen *„Overhead“* an Funktionen mitbringen und dadurch den Server-Arbeitsspeicher stark beanspruchen. Zudem muss der GP-Service bei jeder Anfrage eines Clients neu initialisiert werden, dadurch wird die Performance des Service beeinträchtigt [2]. Da bei der vorliegenden Arbeit die Performance eine zentrale Rolle spielt, wurde eine Alternative gesucht und diese wird im folgenden Kapitel vorgestellt.

[2] http://resources.arcgis.com/ zuletzt aufgerufen am 26.11.13, Suchbegriff: Extending Services

6.3 Server Objekt Erweiterung

Der Datenzugriff auf einen ArcGIS Server erfolgt immer über sogenannte Server Objekte. Ein Geocodierungsdienst setzt ein „Geocodierungs–Server–Objekt“ voraus oder ein Kartendienst benötigt ein „Karten–Server–Objekt“. Es sind diese Server Objekte, welche dem Server GIS-Funktionalitäten verleihen. Aus Kosten-Nutzen Überlegungen ist es für den Hersteller nicht notwendig, sämtliche GIS-Funktionen standardmässig auf dem Server zu implementieren (ESRI, 2013a).

Damit auf dem Server dennoch komplexe GIS-Anwendungen veröffentlicht werden können, wird ein umfassendes *Framework* zur Verfügung gestellt. Dieses ermöglicht es, benutzerdefinierte Server Objekt Erweiterungen (SOE) zu entwickeln und so die Grundfunktionen von ArcGIS−Server zu erweitern. SOE's sind Web-Services, welche mittels *REST*– oder *SOAP–Anfragen* nutzbar sind. Auf der *„ArcGIS Online Hilfe“* sind mögliche Entscheidungskriterien aufgelistet, um die Wahl der Serviceart zu erleichtern (ESRI, 2013a).

Im Gegensatz zu Geoprocessing–Diensten können SOE's nicht eigenständig existieren, sondern sie sind immer an einen bestehenden Kartendienst gebunden. Dies bietet den SOE's jedoch die Möglichkeit, direkt auf die Geodaten des entsprechenden Kartendienstes zuzugreifen und diese für den eigenen Algorithmus zu nutzen. Dies ist mitunter ein Grund wieso die SOE sehr schnell ausführbar sind (ESRI, 2013b).

Durch den Einsatz von SOE's stehen dem Programmierer diverse Optionen zur Verfügung um Geodaten, Zwischenprodukte oder Berechnungsergebnisse zu speichern. Für die eigene Anwendung wurden sämtliche Daten im Arbeitsspeicher *(in Memory Workspace)* abgelegt, weil dies einen schnellen Datenzugriff erlaubt (siehe Abb. 6.4 S.37).

6.3.1 Arcobjects Objektmodell

Arcobjects ist eine Sammlung von plattformunabhängigen Softwarekomponenten, welche in C++ geschrieben wurden. Auf dieser Grundlage bauen alle ArcGIS Anwendungen wie ArcMap, ArcCatalog oder ArcGIS Server auf. Um ArcGIS Anwendungen durch benutzerdefinierte Erweiterungen wie Plug–ins, Add–ins oder SOE's zu ergänzen, ist Arcobjects zu verwenden. Damit die Funktionen vom Arcobject Objektmodell genutzt werden können, müssen die entsprechenden Lizenzen auf dem eingesetzten Gerät freigeschaltet sein, nur so kann die Anwendung ausgeführt werden. Für die diversen Programmiersprachen liegt jeweils ein eigenständiges *Software Development Kit* vor, welches den Zugriff auf das erwähnte Objektmodell erlaubt. Da die SOE in Java geschrieben wird, ist Arcobjects zwingend als externe Java-Bibliothek einzubinden (Höck & Manegold, 2010, S. 5).

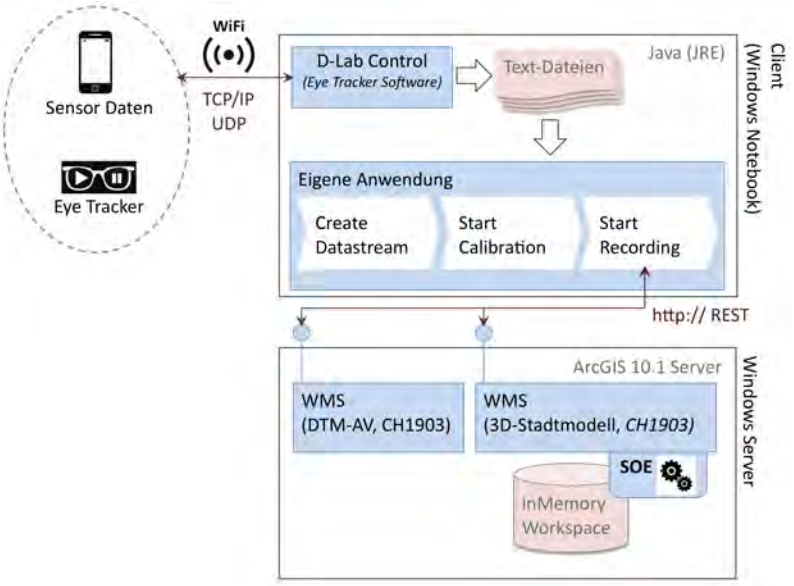

Abbildung 6.4:
Für die Verwendung einer SOE war die Systemarchitektur anzupassen. Damit ein direkter Zugriff auf die Gebäude ermöglicht wurde, musste die SOE an den entsprechenden Kartendienst gekoppelt werden.

6.3.2 Aufbau: Server Objekt Erweiterung

Eine Server Objekt Erweiterung (SOE) wird initialisiert, sobald sie im zugehörigen Kartendienst aktiviert und dieser erneut gestartet wird. Im Verlauf der Instanziierung lassen sich Variablen, Objekte oder Daten im *Cache* abspeichern. Während dem gesamten Lebenszyklus der Server-Objekt-Erweiterung bleiben die Daten im *Cache* erhalten. Diese Eigenschaft kann genutzt werden, um rechenintensive Anwendungsbestandteile bereits während der Initialisierung ausführen und speichern zu lassen. Daher werden die rechenintensiven Prozeduren nur einmal ausgeführt und die Client-Anfragen können schnell mittels den bestehenden Informationen im *Cache* beantwortet werden (ESRI, 2013a).

Durch die Einbindung von Arcobjects in Eclipse (vgl. Kapitel 5.1) kann eine Server-Objekt-Erweiterung basierend auf einer Vorlage generiert werden. Dadurch ist sichergestellt, dass alle zwingend zu implementierenden Methoden und Schnittstellen erstellt werden. Im Folgenden werden nur die für die eigene Anwendung wesentlichen Methoden vorgestellt, für weitergehende Information wird auf die Website von ESRI (2013a) verwiesen.

In Abbildung 6.5 (S. 39) sind die Aktivitäten, die bei der Erstellung des Server-Objektes ablaufen, schematisch dargestellt. Die Methode „init()“ ist eine zwingend erforderliche Funktion und ist für die Erstellung der „noch leeren“ SOE verantwortlich. Durch die Ausführung der Methode wird einerseits auf den *„Logger“* des ArcGIS–Server–Managers zugegriffen, um

den Entwickler bei der Fehlerbehebung zu unterstützen, weil bei SOE ein lokales *„debugging“* nicht möglich ist. Anderseits wird der Zugriff auf den zugehörigen Kartendienst und die darin enthaltenen Geodaten gewährleistet.

Im Gegensatz zur vorgängig beschriebenen Methode ist *construct()* optional und ebenfalls für die Initialisierung des Server Objekts zuständig. Sie wird im Anschluss zum *init()-Aufruf* ausgeführt und dient einerseits dazu, dass die Ausführungs–Parameter jederzeit über den ArcGIS-Server Manager veränderbar bleiben und nicht „hart zu kodieren“ sind. Anderseits wird durch ESRI empfohlen, rechenintensive Abläufe in dieser Methode zu implementieren. Für die eigene Anwendung wurden hier sämtliche Methoden durchgeführt, welche sich von der individuellen Client-Anfrage abspalten lassen, wie beispielsweise die Überprüfung der Serverlizenzen. In den folgenden Abschnitten werden diese ausgelagerten Prozesse kurz erläutert und deren Ausführungsablauf kann in der Abbildung 6.5 (S. 39) nachvollzogen werden.

Damit Geoverarbeitungsmethoden ausgeführt werden können, ist als erstes ein *Geoprocessor-Objekt* zu erstellen. Es dient dazu, auf eine *Feature Class* zuzugreifen oder die Umgebungs- und Ausführungseigenschaften zu definieren. Dazu zählt beispielsweise, ob bereits vorhandene Datensätze überschrieben werden dürfen oder das Koordinatensystems des zu erstellenden Output-Datensatzes selbst definiert werden kann. Dieses Objekt wird nur einmal instanziiert und bei jeder Client-Anfrage wiederverwendet.

Zur Verarbeitung oder Abspeicherung von Geodaten ist ein Workspace zu definieren. Dieser entspricht in etwa einem Ordner bei einem Dateiablagesystem oder dem Pfad zum Speicherort einer Datenbank. Die Methode *createinMemoryWorkspace()* erstellt einen solchen Arbeitsbereich im Arbeitsspeicher und darauf basierend ein *Feature Workspace*.

Die Methode *createFields()* erstellt eine Vorlage der Felder, welche durch einen Feldnamen und einen Felddatentyp definiert werden. Damit die Blicke temporär gespeichert werden können, wird diese Vorlage später genutzt, um eine *Feature Class* effizient zu erstellen. Dies ist relevant, weil bei jeder Client-Anfrage eine *Feature Class* angelegt wird.

Wie in Kapitel 3.3 erwähnt, dient eine *Hashmap* in Java dazu, ein Schlüsselattribut mit einem Wert zu speichern. Durch die Methode *createHashMapBuildingName()* werden die Gebäude-ID und die Gebäudenamen aus dem Kartendienst des 3D–Stadtmodells ausgelesen, um diese anschliessend in einer *Hashmap* abzulegen. Dies ermöglicht es der Anwendung performante Abfragen von Gebäudenamen über die zugehörige ID durchzuführen.
In der letzten Methode, *createtransformParameter()*, werden sowohl die beiden verwendeten räumlichen Referenzsysteme aufgerufen und in Variablen abgelegt, als auch sämtliche

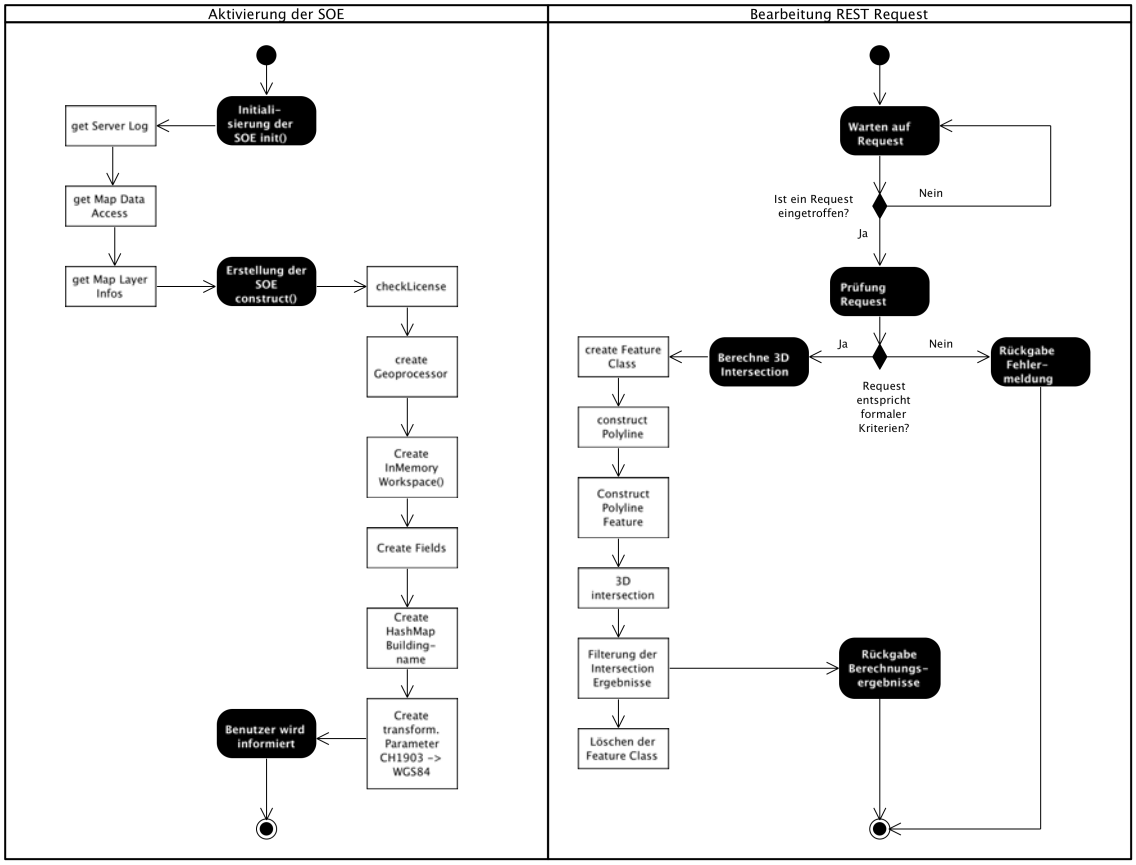

Abbildung 6.5:
In der Abbildung sind zwei Aktivitätsdiagramme dargestellt. Der vollständig ausgefüllte Punkt stellt den Start des Vorgangs dar und der andere das Ende. Die schwarzen Quadrate sind die Aktivitäten, die weissen sind die aufgerufenen Methoden, welche die durch sie erstellten Objekte im Arbeitsspeicher ablegen. In der linken Hälfte ist der Initialisierungsablauf dargestellt, welcher nur einmal während der Veröffentlichung der SOE abläuft. Rechts sind die Aktivitäten, die bei jeder Client-Anfrage wiederholt ablaufen.

Parameter generiert, die zur Koordinatentransformation zwischen den beiden Systemen benötigt werden. Die Transformation ist deshalb angebracht, weil die von Mosimann (2013) erstellten Kalibrationsalgorithmen auf dem schweizerischen Referenzsystem CH1903 beruhen, die GPS-Messungen und die Darstellung in der Karte der Client–Anwendung hingegen auf dem globalen Referenzsystem WGS84.

Nach erfolgreicher Abarbeitung der soeben beschriebenen Methoden, ist die Server Objekt Erweiterung erstellt und einsatzbereit. Diese Information wird dem Server–Manager auf der Administratoren–Website über einen Log-Eintrag mitgeteilt. Im Nachfolgenden wird nun der rechte Teil der Abbildung 6.5 (S. 39) behandelt.

Serveranwendungen verhalten sich in der Regel passiv und warten stets auf das Eintreffen

von Client-Anfragen. Sobald eine Client-Anfrage eintrifft, wird überprüft ob das mitgelieferte JSON-Objekt den formalen Kriterien entspricht (siehe Anhang A.4). Ist dies nicht der Fall, so wird dem Client eine Fehlermeldung zurückgegeben. Ist das JSON-Objekt hingegen vollständig, so kann die 3D-Verschneidung ausgeführt werden.

Dazu wird als erstes eine leere *Feature Class* im Arbeitsspeicher erstellt, dies geschieht mittels der Verwendung der Datenfelder-Vorlage aus der Initialisierungsphase. Anschliessend wird die als JSON-Objekt empfangene Blicklinie deserialisiert, dabei wird als erstes die Geometrie ausgelesen und als Polyline gespeichert. Erst durch den Aufruf der Methode *ConstructPolylineFeature()* wird die Geometrie mit den aus dem JSON-Objekt ausgelesenen Attributen als *Feature* im Arbeitsspeicher abgelegt.

Da die Blicklinie nun als *Feature* vorliegt, kann die 3D-Verschneidung mit dem Stadtmodell berechnet werden. Das Geoverarbeitungswerkzeug wird allerdings so konfiguriert, dass nur die zerschnittenen Blicklinien ausgegeben werden. Dabei ist die Anzahl der Verschneidungen abhängig von der Anzahl der „durchquerten“ Objekte. Auf eine zusätzliche Ausgabe der 3D-Schnittpunkte wird verzichtet. Dies weil, einerseits nur der am nächsten beim Betrachter gelegene Schnittpunkt benötigt wird und nicht sämtliche Schnittpunkte entlang der Sichtlinie und anderseits bei Bedarf die Schnittpunkte aus den *„Vertexen“* der Linienkomponenten hergeleitet werden können. Deswegen kann davon ausgegangen werden, dass durch die beschriebene Konfiguration nur minimale Server-Ressourcen beansprucht werden.

Aus den berechneten Linienkomponenten ist nur diejenige auszuwählen, welche vom Benutzer zum betrachteten Gebäude verläuft. Die Auswahlkriterien entsprechen den Erläuterungen in Kapitel 6.2 (S. 34). Die *Vertexe* der Linienkomponenten werden für beide Koordinatensysteme CH1903 und WGS84 berechnet, hierzu werden die vorprozessierten Transformationsparameter benutzt.

Nach der Selektion und Koordinatentransformation führt die Methode *„Filterung der Intersection Ergebnisse“* allerdings noch weitere Aufgaben aus. Im Falle, dass die Blicklinie ein Gebäude schneidet, wird noch der zugehörige Gebäudenamen ausfindig gemacht. Dies geschieht indem die ID des betrachteten Gebäudes über die bereits im Arbeitsspeicher vorhandene *Hashmap* abgefragt und so der zugehörige Name eruiert wird. Dies ist möglich, weil die selektierte Linie die notwendige Information als Attribut besitzt. Falls keine Gebäude geschnitten wird, wird die empfangene 600m lange Linie in beide Referenzsystem transformiert. Für beide beschriebenen Fälle werden sämtliche Ergebnisse in ein JSON-Objekt gepackt und dem Client zurückgeschickt.

Anschliessend wird die zur Verarbeitung der Blicklinie erstellte *Feature Class* aus dem Ar-

beitsspeicher gelöscht. Für die Erstellung der SOE wurde vorgängig der gesamte beschriebene Workflow in einem lokal auszuführenden Programm getestet. Dabei wurde jeweils nur die erstelle Blicklinie aus der *Feature Class* entfernt. Deshalb bestand die Intention denselben Ablauf in der SOE umzusetzen, jedoch liess sich bei der SOE die einzelne Blicklinie nicht löschen, obwohl dies durch unterschiedliche Varianten getestet wurde. Deswegen musste schliesslich die gesamte *Feature Class* gelöscht werden.

Die soeben beschriebenen Aktivitätsdiagramme verdeutlichen, dass die Server Objekt Erweiterung eine markant feinere Strukturierung des Geoverarbeitungsprozesses zulässt, als dies ein reiner GP-Service ermöglicht (siehe Kapitel 6.2). Sämtliche Eingangsdaten und Verarbeitungsschritte lassen sich detailliert konfigurieren und können darüber hinaus vorprozessiert werden. Im Teil IV dieser Arbeit wird sich zeigen, ob der zeitliche Mehraufwand zur Entwicklung einer Server Objekt Erweiterung gegenüber der Erstellung eines ESRI GP-Service gerechtfertigt war oder nicht.

Kapitel 7

Aufbau: Client-Anwendung

Die Client-Anwendung kann als Bindeglied zwischen dem Nutzer, den externen Sensoren und der Serveranwendung angesehen werden. Der Nutzer gibt die zur Kalibration notwendigen Eingabedaten in der graphischen Benutzerschnittstelle ein und wählt die zu startenden Aufnahmeprozesse aus. Die Client-Anwendung verarbeitet anschliessend die externen Datenströme und geht bei Bedarf eine Verbindung mit der Server-Anwendung ein. In den folgenden Unterkapiteln werden zuerst die zur Entwicklung verwendeten Ressourcen kurz erläutert und erst danach wird die eigentliche Anwendung vorgestellt.

7.1 Verwendete Java-Bibliotheken

Java besitzt von Haus aus eine sehr umfangreiche Programmbibliothek, welche es den Entwicklern ermöglicht, für diverse Anwendungsbereiche lauffähige Applikationen zu erstellen. Dazu können alle grundlegenden Datentypen verwendet, Internetschnittstellen benutzt und unterschiedliche Dateitypen verarbeitet werden. Da Java eine objektorientierte Programmiersprache ist, kann jeder Entwickler eigene Objekte, Methoden oder Datentypen erstellen und diese als Programmkomponenten in separaten Java-Bibliotheken ablegen. Die Programmkomponenten bilden keine lauffähigen Programmbestandteile, können jedoch wiederverwendet oder mit anderen Programmierern ausgetauscht werden. Im Internet kann auf eine umfangreiche Auswahl von quelloffenen Java-Bibliotheken zugegriffen werden, um den Funktionsbereich von Java zu erweitern. Im Folgenden werden die bei der Entwicklung des Clients eingesetzten Java-Bibliotheken vorgestellt.

7.1.1 Java Swing

Swing ist ein Bestandteil der *Java Foundation Classes (JFC)* einer Sammlung von Bibliotheken zur Entwicklung von grafischen Benutzerschnittstellen. Dabei enthält Swing grundlegende Komponenten wie *„Buttons, Radio Buttons oder Textfelder“*, welche zur Erstellung einer Eingabemaske verwendet werden können. Zur einfacheren Verwendung der

Swing Komponenten wurde in *Eclipse* das Plugin *Window Builder* [1] installiert (siehe Kapitel 5.1). Dieses Plugin ermöglicht es in einer graphischen *Layout-Ansicht*, die Komponenten nach dem Prinzip *„What You See Is What You Get“* mittels *„Drag–and–Drop“* anzuordnen. Anschliessend wird der entsprechende Java-Code durch das Plugin automatisch generiert. Danach sind im Code noch die entsprechenden *Event Handler* und *Action Listener* nach der gewünschten Logik zu implementieren.

7.1.2 Java Projekt Swinglabs

Das Java Projekt *Swinglabs*[2] bietet verschiedene Erweiterungen für Java Swing an, wie beispielsweise ein Login-Framework, eine Sortierfunktion oder eine Autovervollständigung. Dabei wird die Entwicklung einzelner Komponenten oftmals in einem Unterprojekt geführt, eines davon war das Projekt SwingX mit den Bibliotheken *„swingx-all“* und *„swingx-ws“*. In einer Desktop-Anwendung lässt sich mit ihnen ein *Open-Street–Map–Kartendienst* anzeigen und auf diesem lassen sich *„Mashups“* oder *„GPS–Tracks“* dynamisch erzeugen. Beim eigenen Client wurden diese beiden Ressourcen genutzt, um die Blickvektoren und die Namen der betrachteten Gebäude anzuzeigen. Auf der offiziellen Projekt–Webseite werden die genutzten Bibliotheken nicht mehr zum Download angeboten, sie können jedoch auf alternativen Websiten[3] heruntergeladen werden.

7.1.3 Jackson

JSON Objekte werden verwendet, um Daten zwischen einem Client und einem Server auszutauschen. Dabei handelt es sich um ein textbasiertes Datenaustauschformat. Zur Verarbeitung der Objekte kann die Java Bibliothek Jackson eingesetzt werden. Diese besteht aus drei Hauptbestandteilen: Der *Data Streaming API*, der *Tree Model API* und der *Databinding API*. Ein einfaches Objekt kann mittels der *Data Streaming API* eingelesen werden. Falls ein in sich verschachteltes Objekt vorliegt, ist die Verwendung der *Tree Model API* angebracht. Soll hingegen aus dem empfangenen Objekt direkt ein Java–Objekt oder ein Datenbank–Objekt erzeugt werden, so ist die *Databinding API* heranzuziehen. Alle drei *API's* ermöglichen nebst dem Einlesen von JSON Objekten auch deren Erstellung. Für die Client Anwendung wurde insbesondere die *Tree Model API* eingesetzt, um einfach auf das einzelne Feature und auf die Geometrie zugreifen zu können[4].

7.2 Strukturierung der Client Anwendung

In den folgenden Unterkapiteln wird die Client Anwendung aus der Anwendersicht beschrieben, wo der Schwerpunkt auf der Interaktion mit der graphischen Benutzerschnittstelle

[1] https://www.eclipse.org/windowbuilder/ zuletzt besucht am 08.12.2013
[2] https://java.net/projects/swinglabs zuletzt besucht am 08.12.2013
[3] http://www.java2s.com/Code/Jar/s/Downloadswingxwsjar.htm zuletzt besucht am 08.12.2013
[4] http://wiki.fasterxml.com/JacksonHome zuletzt besucht am 08.12.2012

liegt. Ebenfalls beschrieben wird die Perspektive des Entwicklers, bei welcher der Fokus hauptsächlich auf der Strukturierung der Java-Klassen liegt.

7.2.1 Clientaufbau aus Anwendersicht

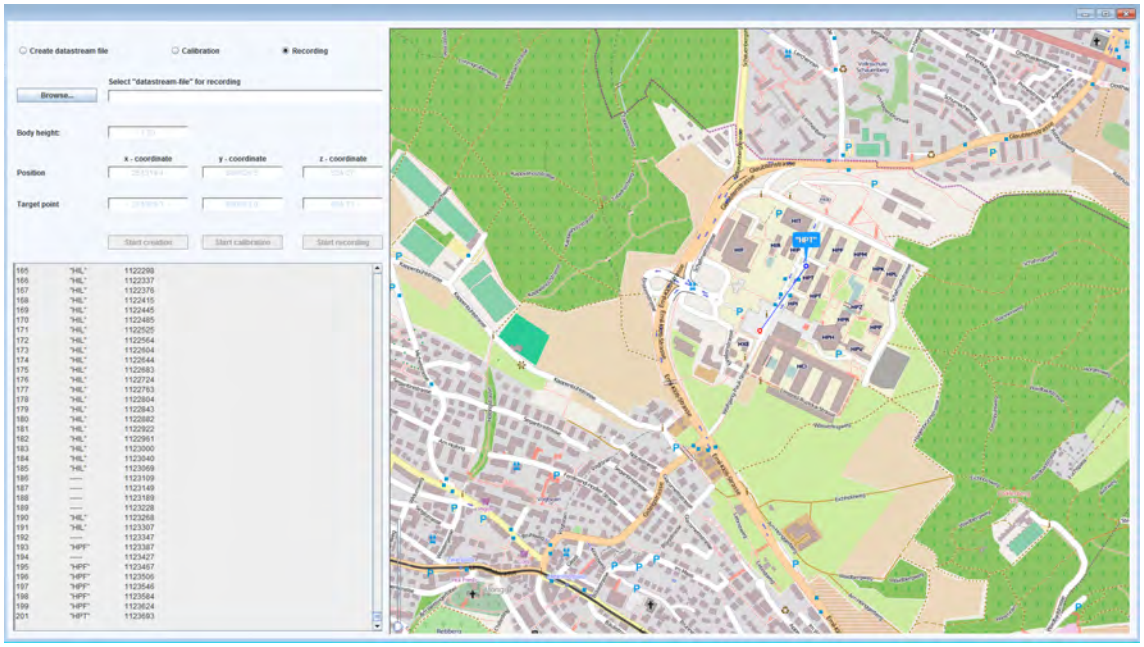

Abbildung 7.1:
Hier ist das GUI des Prototyps mit Eingabe–, Ausgabe– und Kartenfenster abgebildet

Der Benutzer interagiert mit dem Gesamtsystem nur über die graphische Benutzerschnittstelle (GUI) des Clients. Die zur Ausführung der Anwendung benötigten Parameter und Aktionen können über dieses eingegeben oder ausgelöst werden. In Abbildung 7.1 ist das erwähnte GUI ersichtlich. Es besteht aus einem Anwendungsfenster, das in drei Bereiche den Eingabe–, den Ergebnis– und den Kartenbereich unterteilt ist.

Wird die Anwendung gestartet, so ist der *Radio Button „Create datastream file“* aktiviert und der Benutzer wird aufgefordert, die Projektdatei auszuwählen (siehe Kapitel 3.2.2 S. 21). Dies kann er tun, indem er über den *„Browse–Button“* im Datei–Explorer zur gewünschten Datei navigiert. Sobald der Benutzer die Auswahl mit dem *„Start creation Button“* bestätigt, verknüpft die Anwendung die Blickdaten des *Eye Trackers* und die Sensordaten des *Smartphones* miteinander und speichert diese in einer Datei namens *„Datastreamfile“* (siehe Kapitel 3.3 S. 21 und Kapitel 4.3.1 S. 27). Bei erfolgreicher Ausführung wird dies dem Benutzer im Ergebnisbereich mitgeteilt, der *Radio Button „Calibration“* aktiviert und der Pfad der soeben erstellten Datei wird automatisch übernommen und die Felder zur Eingabe der Kalibrationsparameter freigeschaltet.

Im nächsten Schritt hat der Benutzer das Aufnahmesystem von *Eye Tracker* und *Smartphone* zu kalibrieren. Dazu muss er die XYZ–Koordinaten des Standorts und des betrachteten Zielortes eingeben, zudem ist die Körperhöhe respektive die Augenhöhe anzugeben (siehe Kapitel 4.3.2 S. 28). Bestätigt der Benutzer die Eingabeparameter mit dem *„Start calibration" Button* so wird die Berechnung der Höhenwinkel–, der Orientierungswinkel– und der Rollwinkelunbekannten ausgeführt. Im Gegensatz zu der Arbeit von Mosimann (2013, S.47) wird zur Bestimmung der Unbekannten nicht nur der erste valable Blick herangezogen, sondern ganze 250 Blicke berechnet und gemittelt. Die Mittelwerte der Unbekannten werden fortlaufend im Ergebnisfenster ausgegeben. Da die Anwendung erst nach der Datenaufnahme verwendet wird, ist darauf zu achten, dass sowohl für die Kalibrations– wie auch für die Aufnahmedaten ein *„Datastreamfile"* erzeugt wird und anschliessend die Anwendung erneut gestartet wird. Nachdem die Kalibration erfolgt ist, wird der *Radio Button „Recording"* aktiviert und die Felder der Kalibrationsparameter deaktiviert.

Nach der Durchführung der soeben beschriebenen Abläufe ist das System einsatzbereit. Deswegen wird der Benutzer aufgefordert, das *„Datastreamfile"* der Aufnahmedaten auszuwählen und dies mit dem *„Start Recording Button"* zu bestätigten. Anschliessend kann der Standort des Benutzers und der Verlauf der Blicke im Kartenbereich verfolgt werden. Der Personenstandort, respektive die GPS–Position, ist auf der Karte mit einem rot–weissen Punkt dargestellt. Falls der Blick über oder unter dem Gebäude verläuft und somit kein Schnittpunkt resultiert, so wird die Blicklinie hellgrau eingefärbt. Treffen die Blicke jedoch ein Gebäude, so werden ihre Blickvektoren und die Schnittpunkte blau gefärbt. Über den blauen Schnittpunkten wird zusätzlich ein blaues *Label* eingeblendet, darin enthalten ist der Gebäudenamen des aktuell betrachteten Gebäudes (siehe Abbildung 7.1). Im Ergebnisbereich werden die Verschneidungsresultate ebenfalls als Text ausgegeben, eine Zeile enthält die Blick-ID, den Gebäudenamen und den Aufnahmezeitpunkt (siehe Kapitel 3.1 S.19 ff).

7.2.2 Clientaufbau aus Entwicklersicht

Seit den Anfängen der objektorientierten Software-Entwicklung findet das Model–View–Controller (MVC) Paradigma grossen Anklang bei den Softwareprogrammierern (Reenskaug, 1979). Das Paradigma zeichnet sich durch seine Einfachheit und klare Struktur aus. Die Softwarekomponenten werden in drei Kategorien eingeteilt: Die Objektklassen der Kategorie *„Model"* führen die eigentliche Arbeit aus und beinhalten Algorithmen, Datenbanken oder andere komplexe Datentypen. Diese Objektklassen werden strikt von der Kategorie *„View"* getrennt, welche für die Präsentation und das Layout der graphischen Benutzerschnittstelle zuständig ist. Damit die beiden bereits erwähnten Kategorien miteinander interagieren können, ist eine Vermittlungszentrale notwendig. Diese Aufgabe wird durch die

Softwareklassen der Kategorie *„Controller“* übernommen (Krasner & Pope, 1988). Von einer sturen Einhaltung dieses Entwicklungsparadigmas ist abzuraten, da es gegebenenfalls zu einem unnötigen administrativen Mehraufwand führen kann (Veit & Herrmann, 2003). Dies ist mit ein Grund, weswegen das Model–View–Controller Paradigma gelegentlich angepasst wird (Dimov et al., 2005). Es kann somit als Legitimierung betrachtet werden, weshalb im eigenen Code auf eine strikte Einhaltung des MVC–Paradigmas verzichtet wurde. In den folgenden Absätzen werden die Programmstellen explizit ausgewiesen, falls das MVC–Paradigma verletzt wurde.

In Abbildung 7.2 (S. 48) ist die Einteilung der entwickelten Java-Klassen zu den MVC–Kategorien dargestellt. Die Java-Klassen sind durch grau eingefärbte Vierecke und die MVC–Kategorien durch Registersymbole dargestellt, die Kategorien entsprechen in Java den sogenannten *Packages*. *Packages* dienen dazu, funktional zusammengehörende Klassen und Methoden zu ordnen und so den Code übersichtlich zu strukturieren. Im Folgenden werden nicht alle Klassen der Abbildung 7.2 behandelt, da bereits vorgängig einzelne Funktionen beschrieben wurden und daher aufgrund des Klassennamens in diesem Dokument nachgeschlagen werden können(siehe Kapitel 7.2.1 und 6.3).

Die in Kapitel 7.2.1 beschriebene graphische Benutzerschnittstelle ist der Kategorie *„View“* zugeteilt und besteht aus vier Java–Klassen. Das Hauptelement *JFrame* bildet den leeren Rahmen des Anwendungsfensters inklusive dem *„Close“ „Minimize“ und „Maximize“ Button*. Zusätzlich wurde im *JFrame–Element* eine für die Anwendung zentrale Methode, die *„Main Methode“* implementiert. Nur über deren Aufruf lässt sich die Java–Anwendungen erst starten. Das Anwendungsfenster enthält noch weitere Bereiche mit unterschiedlichen Funktionen: Der Eingabebereich *(Input Panel)*, der Kartenbereich *(Map Panel)* und der Ergebnisbereich *(Result Panel)*. Diese Klassen wurden aus *Swing–Elementen* (siehe Kapitel 7.1.1) aufgebaut und enthalten nur den Programm–Code, der das Erscheinungsbild der graphischen Benutzerschnittstelle definiert.

Die Anwendungslogik wird in der Kategorie *„Controller“* umgesetzt. Beim Starten der Anwendung wird durch die Klasse *JFrame* eine Instanz des *„Input Controller “* erzeugt, so dass diese die Koordination zwischen dem Benutzer und der Applikation wahrnehmen kann. Dies kann sie, weil im *„Input Controller “* sämtliche *„Action Listener“* oder *„Event Handler“* implementiert sind. Wenn beispielsweise der Anwender im GUI etwas eingibt und dies durch klicken des entsprechenden *„Button“* bestätigt, so wird dadurch der *„Controller“* aufgerufen. Als erstes prüft dieser, ob alle Eingaben korrekt sind. Im konkreten Fall wäre dies, ob ein Dateipfad zu den Blickdaten ausgewählt wurde oder ob die erforderlichen Koordinaten für die Kalibration angegeben wurden. Ist dem nicht so, so wird der Benutzer – ausgelöst durch den *„Controller“* – mittels einer Mitteilung im GUI auf-

gefordert, die Eingabe zu korrigieren. Erst wenn sämtliche Eingaben den Anforderungen der Anwendungslogik entsprechen, ruft der *„Input Controller "* weitere Java-Klassen oder Methoden aus der Kategorie *„Model"* auf. Hier sei noch angemerkt, dass aus zeitlichen Gründen nur einige mögliche Fehleingaben abgefangen werden und nicht der Anspruch bestand ein marktreifes Produkt zu erstellen.

Die Kategorie *„Controller"* enthält zwei spezielle Klassen des Typs *„Swing Worker"*. Diese waren erforderlich, weil die Anwendung kontinuierlich Datenströme verarbeitet und der Benutzer das System nur über die graphische Oberfläche steuert. Die zwei einfachen Rahmenbedingungen führen dazu, dass die Anwendung mehrere Methoden gleichzeitig ausführen können muss. Wäre die Anwendung dazu nicht in der Lage, würde dies bedeuten, dass das GUI während der Verarbeitung des Datenstroms blockiert ist. Folglich nicht auf einen Mausklick auf den *„Close Button"* reagieren könnte. Erst sobald der Datenstrom unterbrochen würde, wäre die Anwendung erneut empfangsbereit für weitere Befehle. Um diese Erscheinung zu verhindern werden in der Softwareentwicklung *„Threads"* [5] verwendet, welche eine parallele Ausführung von Methoden zulassen. Da die für den Aufbau des GUI's benutzten *Swing–Komponenten* nicht *„Thread–sicher"* sind, würde deren Einsatz unter Umständen zu einer ungewollten Blockade des Gesamtsystems führen[6]. Seit der Java–Version 6 enthält die *Swing-Bibliothek* jedoch sogenannte *„Swing Worker"*, welche ebenfalls eine parallele Ausführung von Funktionen zulässt und bei deren Verwendung die Gefahr einer Systemblockade nicht besteht. Darüber hinaus sind *Swing Worker* in der Lage, Zwischenergebnisse der laufenden Berechnung mitzuteilen. Erst diese Eigenschaft ermöglichte es, die Blickvektoren und ihre Schnittpunkte in „Echtzeit" auf der Karte abzubilden. All dies waren die Gründe, weshalb die Verarbeitung der Datenströme mittels der *Swing–Worker–Klasse* erfolgt[7].

Für das Lesen des Programmcodes ist es sehr hilfreich, einen Überblick des Zusammenhangs zwischen den einzelnen Klassen zu haben. Dieser Aspekt wird in Abbildung 7.2 (S. 48) ersichtlich. Die gestrichelten Linien stellen einen Klassenaufruf dar: Startet der Benutzer beispielsweise eine Kalibration, wählt er im GUI den *„Start Calibration Button"*. Daraufhin prüft der *Controller* die Eingaben und erstellt einen *Calibration Swing Worker*, welcher letztlich die *„Calibration–Klasse"* der Kategorie *„Model"* aufruft. Hier ist noch anzumerken, dass beim Aufnahmeprozess der Kalibration der Programmcode leicht vom MVC Paradigma abweicht dies da der*„Controller"* gleich beim Einlesen der Start- und Zielkoordinaten das Azimut, den Höhenwinkel und die Distanz berechnet. Diese einfachen Berechnungen hätten auch in eine Klasse der Kategorie *„Model"* ausgelagert werden können, was allerdings ein paar zusätzliche Codezeilen erfordert hätte.

[5] http://docs.oracle.com/javase/tutorial/essential/concurrency/threads.html besucht am 08.12.2013
[6] Stichwort: „deadlock"
[7] http://docs.oracle.com/javase/tutorial/uiswing/concurrency/worker.html besucht am 08.12.2013

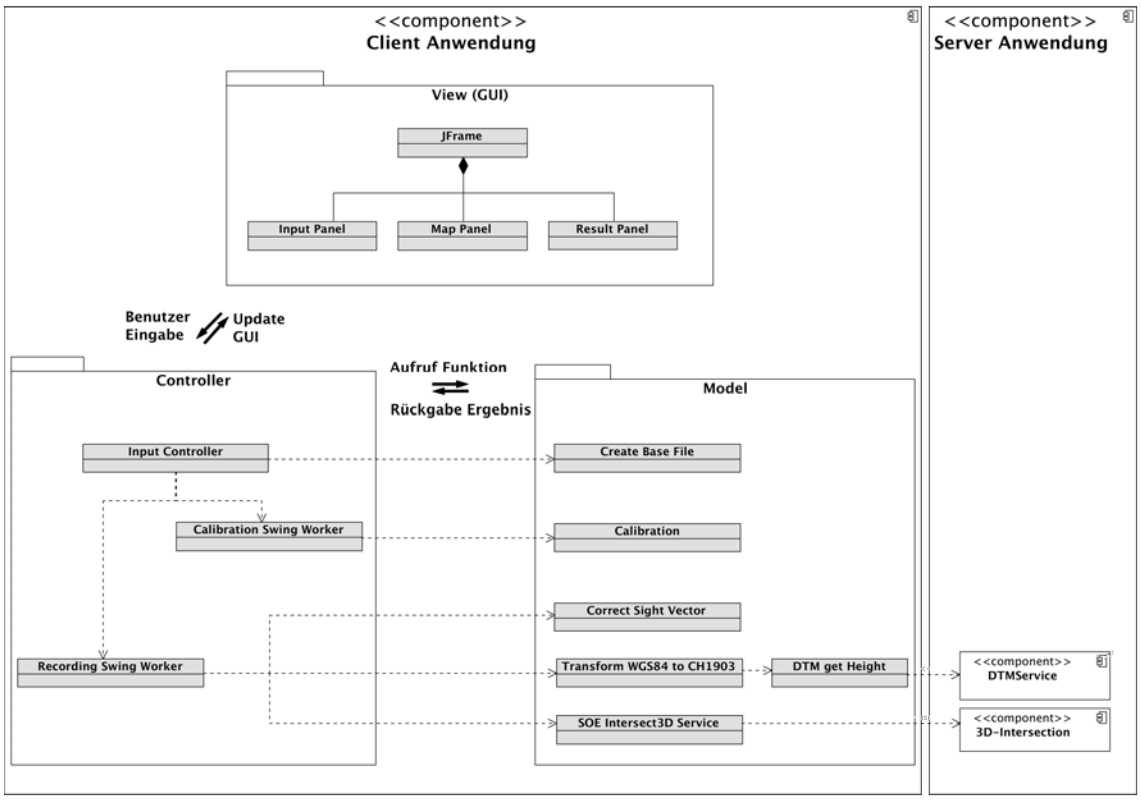

Abbildung 7.2:
Oben ist die Prototypstruktur in Form eines Komponentendiagramms dargestellt

Durch das Starten des Aufnahmeprozesses *„Recording“* greift die Anwendung auf die Klasse *„Correct Sight Vector“* zu. So werden die einzelnen Blickrichtungsvektoren unmittelbar nach deren Aufnahme mit den berechneten Unbekannten des Kalibrationsprozesses korrigiert (siehe Mosimann (2013, S. 19 ff)). Dadurch wird allerdings nur der Richtungsvektor korrigiert, weshalb in einem weiteren Schritt der Anfangspunkt, die Lage des Auges im Koordinatenraum ermittelt werden muss. Dies geschieht, indem die Methoden der Klasse *„Transform WGS84 to CH1903“* ausgeführt werden. Als erstes wird die GPS–Koordinate ins nationale Koordinatensystem transformiert, dies ohne Berücksichtigung der Z–Koordinate. Dazu wurde eine Transformationsroutine gemäss den von Swisstopo publizierten Formeln der strengen Transformation programmiert (Swisstopo, 2008).

Da die durch das GPS gemessene Höhe bis zu 50m von der tatsächlichen Lage abweicht, war es zweckmässig, die Höhe aus dem nationalen digitalen Terrain Modell auszulesen. Deswegen stellt die Klasse *DTM get Height* eine Verbindung zum DTM–Kartendienst her und gewährleistet eine dynamische *getFeatureInfo-Abfrage*. Zur Berechnung der Abfrageparameter wird die soeben transformierte GPS-Koordinate genutzt und gemäss der in Kapitel 6.1.1 (S. 31) geschilderten Methode vorgegangen.

Aufgrund der durchgeführten Berechnungen kann nun anhand der Augenlage und des Blickrichtungsvektors eine 600m lange Linie konstruiert werden. Dazu wird zusätzlich zum Startpunkt der Linie auch deren Endpunkt ermittelt. Die Berechnungen erfolgen durch die Klasse *Recording Swing Worker* ebenfalls entgegen dem MVC Paradigma. Die Berechnungen hätten jedoch ohne weiteres in eine Klasse der Kategorie *„Model“* ausgelagert werden können. Während der Entwicklung brachte diese Aufteilung gewisse Vorteile und wurde anschliessend aus zeitlichen Gründen nicht mehr umgebaut.

Teil IV

Evaluation des Prototyps

Im vierten Teil wird die entwickelte Anwendung durch ausgewählte Versuche geprüft. Die Ergebnisse werden mit den relevanten Arbeiten verglichen und bewertet.

Kapitel 8

Versuchsreihe

In diesem Kapitel werden die zur Bewertung der Anwendung ausgewählten Versuche im Detail vorgestellt. Jeder Testfall wird in einem separaten Unterkapitel abgehandelt inklusive der dabei gesetzten Ziele, der angewandten Analysemethoden und der erwarteten Ergebnisse.

8.1 Versuchsumgebung

In diesem Kapitel werden sämtliche Rahmenbedingungen und eingesetzten Mittel vorgestellt, welche für die geplanten Versuche gelten. Dies sind beispielsweise die verwendete Software und die Begründung zur Auswahl des Testgebiets.

8.1.1 Verwendete Hardware

Zur Messung der Effizienz des Prototyps wird ein ***Desktop*** eingesetzt. Es handelt sich um einen Fujitsu Rechner mit einem 8 Kern Prozessor und den Spezifikationen:

- Windows 7 Enterpise Edition (64−bit, Service Pack 1)
- Intel (R) Xeon(R) CPU E31240 @ 3.30GHz
- RAM 8.00 GB

Die Entscheidung zur Performancemessung anhand eines stationären Arbeitsplatzes lag einerseits darin, dass nur die gewählte Systemarchitektur beurteilt werden sollte und dies möglichst unabhängig von externen Einflussfaktoren. Als externe Faktoren gelten beispielsweise die Netzverfügbarkeit am gewählten Standort, die Abschirmungen durch Gebäude, elektrische Übertragungsleitungen oder die momentane Netzauslastung. Nur schon bei der Evaluierung der Internet−Performance eines Smartphones, müsste eigentlich die Tageszeit berücksichtigt werden, wie dies Huang et al. (2010) aufzeigten. Sie wählten unter anderem die Umsatzrate (RTT) als Messparameter und zeigten, dass diese in Abhängigkeit von der Tageszeit zwischen 300ms bis 700ms schwankt.

Der Ausschluss all dieser Einflüsse kann an einem stationären Arbeitsplatz wesentlich besser kontrolliert werden, als es bei mobilen Anwendungen der Fall ist. Anderseits aber entwickeln sich die mobilen Geräte so rasant, dass deren Leistung in sehr kurzen Zyklen stark verbessert wird. Dabei ist zu erwähnen, dass die benutzte ***Internetverbindung*** an der ETH eine Download–Rate von ca. 25MB/s und eine Upload–Rate von ca. 114MB/s aufwies [1].

Für die Feldaufnahmen ist ein ***Notebook*** erforderlich. Für sämtliche Versuche wurde ein Modell des Herstellers DELL mit einem vier Kernprozessor und den folgenden technischen Spezifikationen genutzt:

- Windows 7 Professional (32−bit, Service Pack 1)
- Intel (R) Core(TM) i5-2520M CPU @ 2.50GHz
- RAM 4.00 GB

Das *Notebook* wurde so konfiguriert, dass nur die essentiellen Applikationen zur Durchführung der Messungen installiert wurden. Dadurch wurden potentielle Fehlerquellen reduziert.

Die Professur für *Geoinformation Engeneering* der ETH Zürich verfügt über einen mobilen ***Eye Tracker*** *(Modell: Dikablis)* der Firma Ergoneers[2]. Das System zeichnet sich dadurch aus, das es dem Tester ermöglicht, das Blickverhalten der Probanden mobil und in Echtzeit zu überwachen. Ein weiterer Vorteil ist, dass sich die Blickdaten mit zusätzlichen externen Datenströmen synchronisieren lassen. Dies ist eine Voraussetzung, um die bereits beschriebenen Anforderungen zu erfüllen.

Die zur Berechnung der 3D−Blickvektoren erforderlichen GPS− und Azimutdaten werden durch ein ***Smartphone*** von Samsung vom Typ Galaxy Nexus mit dem Betriebssystem Android V4.0 ausgegeben. Es verfügt über einen digitalen Kompass und ein Gyroskop. Wie bereits Mosimann (2013, S. 9) erwähnte, ist das Gyroskop nicht physisch vorhanden sondern wird durch den Magnet– und den Beschleunigungssensor virtuell erzeugt.

8.1.2 Verwendete Software

Damit die oben beschriebenen Hardware-Komponenten untereinander kommunizieren und Daten austauschen können, müssen auf den Geräten verschiedene Software-Pakete installiert werden. Die Software ***Dikablis−Record*** ist zur Kalibration der Kameras und zur Aufnahme der reinen Blickdaten notwendig. Die Software gleicht die Blickdaten der Augen–Kamera mit dem Video der Blickfeldkamera ab. Mit anderen Worten sorgt sie unter ande-

[1] Speedtest: http://www.swisscom.ch/de/res/hilfe/loesung/speedtest-fuer-die-dsl-geschwindigkeit.html
[2] http://www.ergoneers.com/de/products/dlab-dikablis/overview.html

rem dafür, dass die Blicke auf dem Video abgebildet werden können.

Eine wichtige Rolle bei den Feldversuchen spielt die Eye Tracker−Software ***D−Lab Control***, die ebenfalls auf dem Notebook installiert wird. Sie dient dazu, die externen Datenströme (GPS-, Azimut- und Rollwinkeldaten) mit den Blickdaten und dem Video zu synchronisieren und zu speichern. Die Feldaufnahmen werden über diese Software kontrolliert und gesteuert.

D−Lab Control verfügt über eine UDP-Netzwerkschnittstelle, welche der exklusive Eingang für externe Datenströme ist. Damit nun das Smartphone an diese Schnittstelle anknüpfen kann, muss auf dem Notebook die Software ***Connectify-Lite*** [3] installiert sein (Grob, 2012, S. 16). Dadurch wird der Rechner zu einem Wi−Fi−Router (Hotspot). Dies erlaubt die einfache Einrichtung eines Netzwerkes, bestehend aus dem Smartphone und dem Notebook.

Die Aufnahmedaten können anschliessend mit einem weiteren Softwarepaket des Eye-Trackers mit ***Dikablis-Analysis***, überarbeitet werden. *Dikablis-Analysis* dient der Nachkalibration der Aufnahmedaten, Verbesserung der Pupillenerkennung, manuellen Auswertung der Blickfilme oder dem Zuschneiden der Videosequenzen.

Eine detaillierte Auswertung der Blickdaten ist mit dem Softwarepaket ***D−Lab*** möglich. Die Software wird dazu verwendet, das Blickverhalten über statistische Kennwerte zu beschreiben und in Grafiken zu präsentieren oder aber Heat-Maps zu erstellen (Hot-Spot-Analysen).

8.1.3 Auswahl des Testgebietes und der Teststandorte

Die Versuche können grundsätzlich in einem beliebigen Gebiet stattfinden. Es ist jedoch von Vorteil, wenn die zur Berechnung der 3D-Verschneidung notwendigen ***Geodaten*** in einer ausreichenden Qualität vorhanden sind − namentlich sind dies: ein 3D-Gebäudemodell und das entsprechende digitale Terrainmodell (DTM).

Einerseits weil die Höhenausgabe des Smartphones bis zu 50m von der exakten Höhe abweichen kann und daher die Höhen aus dem DTM zur Berechnung zu verwenden sind. Je höher die Auflösung der Grundlagedaten ist, desto exakter können die Schnittpunkte berechnet werden. Anderseits weil die eigenständige Anfertigung eines 3D-Stadtmodell's mit Werkzeugen wie beispielsweise „Google Sketch up“ sehr aufwendig und zeitintensiv ist.

Die erwähnten Geodaten standen im ausgewählten Testgebiet für die Feldversuche auf dem

[3] http://www.connectify.me/

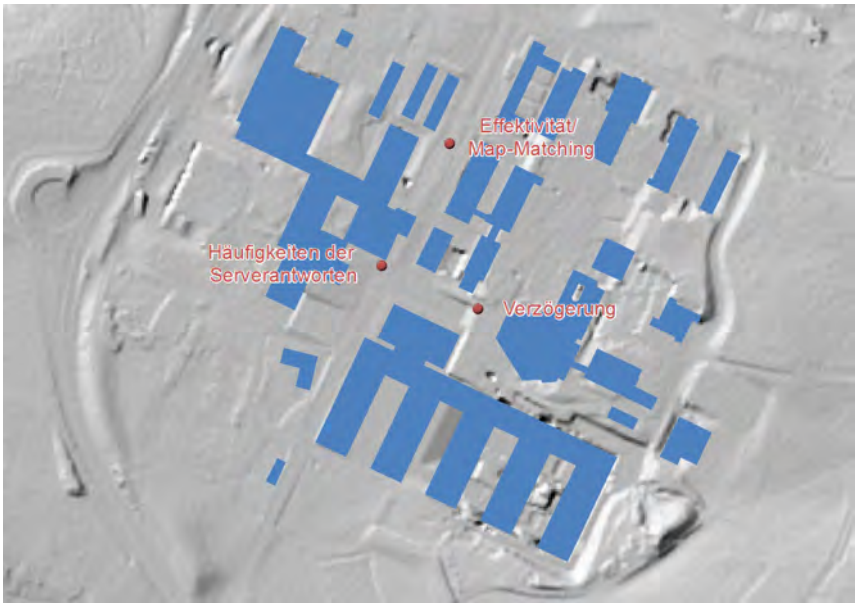

Abbildung 8.1:
ETH Zürich Hönggerberg, die Grundrisse der 3D-Gebäude sind auf dem digitalen Terrainmodell (Hillshade) abgebildet. Die rot eingezeichneten Punkte repräsentieren die Standorte der Testpersonen während den Versuchen.

Hönggerberg des Areals der *ETH* Zürich zur Verfügung (vgl. Abb. 8.1, S. 55) und wurden über Kartendienste veröffentlicht (siehe Kapitel 6.1 S. 31).

Da der verwendete Eye Tracker, die mit dem Smartphone synchronisierte Daten, nicht in Echtzeit als Datenstrom ausgeben kann, werden die erwähnten Geodaten während des Feldtests nicht benötigt. Bei der anschliessenden Prototypauswertung wird jedoch der synchronisierte Datenstrom in der tatsächlichen Aufnahmefrequenz simuliert und so greift der Client frühestens während der Auswertung auf die erstellten Kartendienste zurück.

Erst durch die Darstellung des DTM's als Hillshade werden potentielle Fehlerquellen erkennbar. Beispielsweise kann einfach gezeigt werden, wie wichtig es ist die ***Aktualität der Geodaten*** zu kennen. Dies ist beim südlich liegenden Gebäude mit der grössten zusammenhängenden Grundfläche zu erkennen (vgl. Abb. 8.1 S. 55). Dieses Gebäude wurde erst vor wenigen Jahren erstellt, während der Aufnahme des DTM's war nur die Baugrube zu erkennen und das Erdmaterial war südöstlich deponiert. Heute ist von der Baugrube und der Materialdeponie nichts mehr zu sehen. Daher war es bei der Auswahl der Teststandorte notwendig, solche Fehlerquellen aufgrund von Überprüfungen vor Ort auszuschliessen.

Weiter ist zu beachten, dass bei der Erstellung von DTM jegliche ***Kunstbauten*** von der Erdoberfläche entfernt werden. Daher wäre ein Teststandort auf einer Brücke nicht zuläs-

sig, da dadurch der Stützpunkt des Blickvektors auf Terrainhöhe liegen würde anstatt auf der Höhe der Kunstbaute.

Die Standortauswahl ist ein entscheidender Einflussfaktor für die ***Genauigkeit der GPS- und Azimutmessung***. Die wesentlichen Kriterien, welche es bei präzisen GPS-Messungen zu berücksichtigen gilt, listeten JOSUA & STEFAN (2013, S. 7) in ihrem kürzlich erschienen technischen Bericht anschaulich auf. Bei der Positionswahl ist insbesondere die Reflexion und die Verfügbarkeit der Satellitensignale entscheidend. Je offener das Gelände ist, desto höher ist die Wahrscheinlichkeit, dass ausreichend GPS-Satelliten verfügbar sind und entsprechend weniger Reflexionen der Satellitensignale sind zu erwarten.

Bei der Azimutmessung ist zu beachten, dass die Ausgabewerte durch elektronische Geräte oder Starkstromleitungen stark beeinflusst werden können. Deswegen mussten zumindest die offensichtlichen elektrischen Störquellen ausgeschlossen werden. Wie in diesem Kapitel aufgezeigt wurde, besteht das Gesamtsystem aus vielen elektrischen Komponenten, so dass eine gewisse „ Eigenstörung“ nicht vollkommen auszuschliessen ist.

Dies alles sind Gründe, weshalb die hier erzielten Ergebnisse nur für das vorliegende Testgebiet gelten und nicht ohne weiteres auf beliebige Städte übertragen werden können. Eine Übertragung wäre nur zulässig, falls die Gebäudehöhen, Strassenbreiten und Struktur ähnlich ausgeprägt wären.

8.1.4 Beim Feldtest verwendete Daten

Zur Durchführung der Feldtest's werden keine Daten benötigt. Damit jedoch die Aufnahmen ausgewertet werden können, sind die präzisen 3D–Koordinaten der Personenstandorte, der Kalibrationspunkte sowie die Augenhöhe des Probanden aufzunehmen. Diese Daten können bereits vor der Aufnahme ermittelt werden, in dem die Koordinaten mittels GIS aus den Geodaten herausgelesen werden.

8.2 Effizienz

Dieser Testfall verwendet die Daten, die bei den Feldtests gespeichert wurden. Der Test wird an einem fixen Arbeitsplatz der *ETH* durchgeführt. Dabei handelt es sich um eine einfache Messung der Performance.

8.2.1 Ziel

Das Ziel des Versuches ist es, die Effizienz der Anwendung zu messen und zu eruieren welche Zeit der Berechnungszyklus einer Verschneidung des Blickes mit einem Gebäude beansprucht. Die Berechnungszeiten werden dabei auf die Client- und Server-Anwendung

aufgeteilt. Das dadurch erzielte Ergebnis ist die Voraussetzung, um eine der Kernfragen dieser Arbeit beantworten zu können: Wie mit der nicht zu vermeidenden Antwortzeit des erstellten Services umzugehen ist.

8.2.2 Verwendete Analysemethode

Im Versuch werden die erstellten Varianten: Geoprocessing–Service und Server Objekt Erweiterung beurteilt. Die Zeitmessung wird dabei innerhalb des Programm-Codes implementiert mit Hilfe des Java-Befehls *Current Time Millis*. Dieser Befehl gibt die aktuell vergangene Systemzeit seit dem 1. Januar 1970 (UTC) in Millisekunden aus. Die Berechnungsdauer wird definiert als die Dauer von der clientseitigen Berechnung eines Blickes bis zur Ankunft der Serverantwort beim Client. Dabei wird nicht nur die gesamte Berechnungsdauer ermittelt, sondern auch die einzelnen rechenintensiven Methoden.

Die Bewertung der Varianten auf Effizienz wird mit Hilfe von statistischen Kennwerten erfolgen. Die Zeitdifferenzen zwischen den betrachteten Varianten wird im Anschluss zusätzlich auf Signifikanz überprüft.

8.2.3 Erwartungen

Die Erwartung ist einerseits, dass der Server Objekt Dienst signifikant performanter ist als der entsprechende Geoprocessing–Service. Unter den gegebenen Testbedingungen wird davon ausgegangen, dass keine grosse Streuung um die mittlere Berechnungsdauer auftritt. Anderseits wird erwartet, dass die 3D–Verschneidung der rechenintensivste Programmbestandteil ist, weil dabei Daten übers Internet gesendet werden und relativ aufwendige Berechnungen notwendig sind.

8.2.4 Ergebnisse

Die Evaluation bestätigte die Erwartungen des Verfassers: Mit einer mittleren Berechnungszeit von 112.21 Millisekunden (ms) war die Server Objekt Erweiterung (SOE) rund 35 mal schneller als der Geoprocessing Dienst (GP). Für die Berechnung einer einzelnen 3D–Verschneidung benötigte dieser durchschnittlich 3'744 ms. Der Median lag für die SOE bei 110 und für den GP–Service bei 3'718 ms (siehe Tabelle 8.1). Im günstigsten Fall benötigt der GP–Service 3.2 Sekunden länger als die SOE für dieselbe Berechnung.

Unter Einbezug der mittleren Berechnungszeit kann gesagt werden, dass die Streuung der Messwerte um den Mittelwert für den GP–Service eher klein sind, bei der SOE hingegen relativ gross. Daher konnten diesbezüglich die Erwartungen nicht vollumfänglich bestätigt werden. Die Schwankungen von rund 12 Millisekunden werden allerdings das weitere Vorgehen nicht beeinflussen, da der absolute Betrag doch sehr gering ist.

Tabelle 8.1:
Statistische Kennwerte der Messresultate der untersuchten Verschneidungsdiensten

	GP–Service (ms)	**SOE (ms)**
Mittelwert	3'744.39	112.21
Standardabweichung	156.56	11.86
Median	3'718	110
kleinster Wert	3'524	78
höchster Wert	4'380	281
Spannweite	856	203
Beobachtungen	383	1'263

Im Folgenden werden die Messwerte auf statistische Signifikanz untersucht. Als erstes wird mittels dem Kolmogorov–Smirnov– und dem Shapiro–Wilk–Tests überprüft, ob eine Normalverteilung vorliegt. Voraussetzung für die Tests ist, dass es sich bei den zu untersuchenden Messwerten um kontinuierliche Variablen handelt. Im vorliegenden Fall ist diese Voraussetzung erfüllt, da Berechnungszeiten als kontinuierliche Werte einzustufen sind. Beide Tests nehmen mit ihrer Nullhypothese an, dass die zu bewertenden Daten normalverteilt sind. Die Testresultate sind in der Tabelle 8.2 ersichtlich, wobei „N“ die Anzahl Messwerte ist, „D“ die maximale Distanz zur Normalverteilung in relativen Einheiten und der „P-Wert“ gibt an, wie wahrscheinlich die vorgefundene Abweichung unter Annahme einer normal verteilten Grundgesamtheit ist. Gemäss den Ergebnissen des Kolmogorov–Smirnov– wie auch des Shapiro–Wilk–Tests, muss die Nullhypothese für beide Messreihen verworfen werden, da die P–Werte deutlich kleiner sind als das gewählte Signifikanzniveau $\alpha = 0.05$ und somit keine Normalverteilung vorliegt.

Tabelle 8.2:
In der Tabelle sind die Testergebnisse der Überprüfung auf Normalverteilung aufgeführt.

		Kolmogorov–Smirnov–Test		**Shapiro–Wilk–Test**	
	N	D	P	W	P
SOE	1'263	0.27	1.03E-81	0.69	0
GP	383	0.09	0.0034	0.91	0

Mit dem U–Test kann untersucht werden, ob sich die zwei Messreihen signifikant voneinander unterscheiden. Der Vorteil des U–Test's gegenüber dem T–Test ist, dass dieser keine Normalverteilung der Variablen voraussetzt. Für die Bestimmung der Prüfgrössen werden alle Beobachtungswerte aufsteigend sortiert und jedem Wert ein Rang zugeordnet. Ansch-

liessend wird das Rangmittel der Messreihen berechnet und der U–Wert ermittelt[4]. Das Excel Add–In „WinSTAT“[5] führt die Transformation der U–Werte auf die Abszissenwerte Z einer entsprechenden Normalverteilung automatisch durch und ermittelt die Signifikanz P. Dies bedeutet, dass zur Interpretation der Datensätze der P–Wert mit dem gewählten Signifikanzniveau $\alpha = 0.05$ verglichen werden darf. Ist der P–Wert kleiner als α, so sind die Messreihen signifikant verschieden, ist der P–Wert grösser als α, so kann kein statistisch signifikanter Unterschied ausgemacht werden.

In der Tabelle 8.3 sind die Ergebnisse des U–Tests aufgelistet. Aufgrund des P–Wertes wird deutlich, dass die Messreihen signifikant unterschiedlich sind und dies mit einer Wahrscheinlichkeit von 100%.

Tabelle 8.3:
In der Tabelle sind die ermittelten Prüfwerte des U–Tests dargestellt. Diese dienen zur Überprüfung, ob sich zwei Stichproben statistisch signifikant unterscheiden.

	N	**Rangmittel**	**U–Wert**
Server Objekt Erweiterung	1’263	632	0
Geoprocessing Service	383	1’455	483’729
		Z–Wert	**P–Wert**
		30.11	0

In Tabelle 8.4 (S. 60) sind die Berechnungszeiten ausgewählter Programmbestandteile ersichtlich. Zur Berechnung der Koordinaten des ersten Stützpunktes P1 des Blickes, werden die Java Klassen *„Transform WGS84 to CH1903“* und *„DTM getHeight“* aufgerufen. Aufgrund der gewählten Testumgebung mit sehr performanter Internetverbindung und einem leistungsstarken Desktop, ergab sich ein Berechnungsmittelwert von nur 1.63 Millisekunden. Auffällig an den Werten war, dass nur bei der Berechnung jedes 5–ten bis 10–ten Blickes eine Berechnungsdauer zwischen einer und 62 Millisekunden ausgegeben wurde. Diese Schwankungen könnten eventuell auf die Belegung der Internetbandbreite durch andere Benutzer zurückzuführen sein. Bei der Verwendung von einer leistungsärmeren Testumgebung ist deshalb mit höheren Berechnungszeiten zu rechnen.

Der Punkt „Verschneidung“ der Tabelle 8.4 beinhaltet das Senden der Blicklinie vom Client zum Server, die Berechnung der 3D–Verschneidung und die Rückgabe der Blicklinien–Koordinaten im Referenzsystem WGS84 und CH1903 sowie gegebenenfalls den Namen des betrachteten Gebäudes. Die Berechnungsdauer der „Verschneidung“ nimmt wie vermutet

[4] http://www.methodenberatung.uzh.ch/datenanalyse.html besucht am 30.12.2013
[5] http://www.winstat.de besucht am 30.12.2013

die längste Zeit in Anspruch und bestimmt dadurch die Gesamtzeit des Berechnungszyklus für die Verschneidung einer Blicklinie mit dem 3D–Stadtmodell.

Die Java Swing Worker Klasse ist ein nebenläufiger Programmprozess, ein sogenannter *Thread*, welcher im Hintergrund läuft und die graphische Benutzeroberfläche nicht blockiert. Bei der Verarbeitung der Blickdaten sendet die Hauptmethode *„doInBackground()“* des Swing Workers kontinuierlich Blickdaten an den Server und liefert erst ein *„Return Statement“*, wenn sämtliche Blickdaten abgearbeitet oder der Datenstrom unterbrochen wurde. Während der Ausführung dieser Methode kann von aussen nicht auf darin enthaltene Daten zugegriffen werden. Als Ausweg erlaubt die *„doInBackground()“* mittels der Methode *„process“* und dem Befehl *„publish“*, aktuelle Zwischenergebnisse an die graphische Benutzeroberfläche auszugeben. Die dafür benötigte Zeit ist unter dem Punkt „Update GUI“ in der Tabelle 8.4 angegeben. Sie liefert einen bescheiden Beitrag von durchschnittlich acht Millisekunden an die gesamte Berechnungsdauer.

Die restlichen Programmbestandteile beanspruchten keine in Millisekunden messbaren Beiträge zur Gesamtzeit. Es ist allerdings anzumerken, dass während der Entwicklung des Programmes mehrmals festgestellt wurde, dass teilweise die Initialisierung der Applikation sowie die der *„While–Schlaufe“* innerhalb des Swing Workers eine gewisse Zeit beanspruchten. Weiter konnte festgestellt werden, dass die Berechnung des ersten an den Server gesendeten Blickes mehr Zeit beanspruchte wie die restlichen. Dies könnte damit zu tun haben, dass der Service eine gewisse Aktivierungszeit benötigt.

Auf die letzten zwei Zeilen der Tabelle 8.4 wird erst bei der Besprechung der Ergebnisse des Versuches „Verzögerung“ eingegangen (siehe Kapitel 8.3.5 S. 63).

Tabelle 8.4:
Statistische Kennwerte der Messresultate der untersuchten Programmbestandteile

	Koordinaten P1	Verschneidung	Update GUI	Gesamtzeit
Anzahl Messungen	1’263	1’263	1’263	1’263
Mittelwert (ms)	1.63	112.21	7.47	121.37
Standardabweichung	4.99	11.86	1.67	12.11
Median	0	110	8	118
tiefster Wert	0	78	0	95
höchster Wert	62	281	14	351
3tes Quartil (75%)	0	124	8	132
Quantil bei 150 ms				99.76%

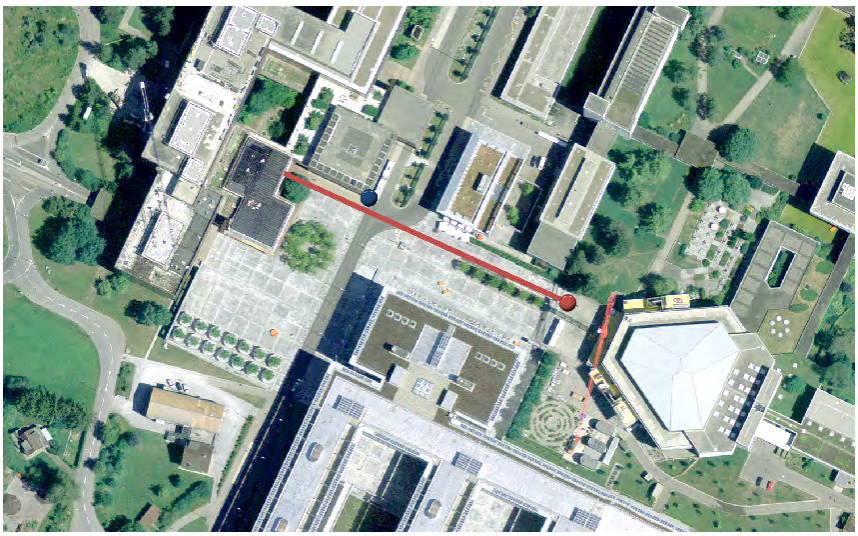

Abbildung 8.2:
roter Punkt: Standort der Testperson / blauer Punkt: Kalibrationspunkt zur Eichung des Gesamtsystems / die Testperson betrachtet das Zentrum des Gebäudes entlang der roten Linie

8.3 Verzögerung

Wie bereits in vorangegangenen Kapitel erwähnt, zeichnet der mobile *Eye Tracker* die Blicke in einer Frequenz von 25Hz auf, d.h. alle 40ms wird ein Blick erfasst. Da für eine 3D–Verschneidung immer eine gewisse Berechnungsdauer beansprucht wird, entsteht alleine durch diese Tatsache eine Verzögerung gegenüber einer Echtzeit–Antwort. In diesem Testfall werden nun genau diese Verzögerungen quantifiziert.

8.3.1 Ziel

Solange die Berechnungsdauer weniger als 40ms andauert, ist die dadurch verursachte Verzögerung nicht gravierend. Werden allerdings die 40ms überschritten, so beginnen sich die Verzögerungen zu kumulieren. Das Ziel ist es, genau dieses Verhalten der SOE zu beziffern.

Kumulationen lassen sich nur vermeiden, wenn man die Berechnungsdauer verkürzt oder aber nicht jeden Blick an den Server schickt. Eine Möglichkeit die Anzahl der gesendeten Blicke zu reduzieren ist, dass nur über eine bestimmte Periode gemittelte Blickdaten – sprich Fixationen – gesendet werden. Folglich soll deshalb auch die minimale Fixationsdauer in Erfahrung gebracht werden bei welcher keine Aufsummierung der Verzögerungen stattfindet.

8.3.2 Standort und Instruktionen

Bei diesem Versuch es ist wichtig, dass die Blicke stets auf ein Gebäude treffen damit die 3D-Verschneidungen kontinuierlich berechnet werden. Daher wurde bei der Auswahl des Zielgebäudes darauf geachtet, dass eine möglichst grossflächige Fassade vorhanden ist.

Für den Standort der Testperson war entscheidend, dass dieser im Feld auffindbar ist und auf dem Orthophoto identifizierbar ist (vgl. Abb. 8.2, S. 61). Für die Auswahl der Kalibrationspunkte galt dasselbe wie für den Standort, zusätzlich müssen diese aber für die Testperson sichtbar und einfach visuell fixierbar sein.

Die Instruktionen für die Testperson sind minimal, sie hat nämlich nur das Gebäudezentrum während einer Minute zu betrachten.

8.3.3 Verwendete Analysemethode

Die Berechnungszeiten gemäss der Beschreibung im Kapitel 8.2.2 werden in einem zweidimensionalen Diagramm eingezeichnet. Auf der X-Achse wird die Aufnahmedauer eingetragen und auf der Y-Achse die kumulierten Verzögerungen jeweils in Millisekunden. Dadurch entsteht eine Punktewolke von Messdaten, welche im Anschluss mittels einer linearen Regressionsanalyse untersucht wird.

Aus den Daten des „DataStreamfiles“ (siehe Kapitel 7.2.1 S. 44) werden die Fixationen mit einem verteilungsbasierten Identifikationsalgorithmus I–DT berechnet und anschliessend die Schwerpunktkoordinaten der Fixationen in eine Datei mit der Struktur analog der Aufnahmedaten abgespeichert (Salvucci & Goldberg, 2000, S. 74). Die Fixationsdauer wird dabei im Attribut „waitMsec“ abgelegt (siehe Kapitel 3.3 S. 21). Es ist zu erwähnen, dass das so erstellte „DataStreamfile“ zur Bestimmung der kumulierten Verzögerungen eingesetzt werden darf, jedoch kann die Anwendung die Fixationsdatei nicht in „echter“ Aufnahmefrequenz simulieren. Dies weil die Fixationsdauer nicht dem Intervall zwischen zwei Fixationen entspricht und somit die Wartezeit gemäss dem Attribut „waitMsec“ zu kurz ist.

Zur Eruierung der Fixationen sind zwei Grenzwerte anzugeben, namentlich den Verteilungsradius *(„dispersion radius“)* und die minimale Fixationsdauer. Der Verteilungsradius wird in Pixel angegeben und entspricht einem Bereich auf dem Video der Feldkamera. Aufgrund der ersten Ergebnisse wird eine erste Fixationsdauer definiert, in den Berechnungsalgorithmus eingesetzt und erneut eine lineare Regressionsanalyse durchgeführt. Dieses Verfahren wird iterativ wiederholt, bis sich die Steigung der Regressionsgerade langsam an 0 annähert. Oder mit anderen Worten, bis keine Kumulation der Verzögerungen mehr auftritt.

8.3.4 Erwartungen

Während der Entwicklung bestand der Anspruch eine äusserst performante Anwendung zu entwickeln. Daher kann nun erwartet werden, dass durch den Einsatz eines ausgewählten Fixationsalgorithmus keine kumulierten Verzögerungen auftreten. Es versteht sich von alleine, dass zur Erkennung der Fixationen die minimale Dauer von zwischen 100 ms und 200 ms eingehalten wird (siehe Kapitel 2.3 S. 11).

8.3.5 Ergebnisse

Bereits aufgrund der in Kapitel 8.2.4 ermittelten Berechnungszeit (121.37 $\pm$12.11ms) war zu erkennen, dass diese rund drei mal länger sind wie die Blickaufnahmefrequenz (40ms). Daher ist es nicht erstaunlich, dass die Regressionsgerade der kumulierten Verzögerungen eine Steigung von rund 300% aufweist. Würde die Aufnahme bereits seit einer Stunde laufen, dann würde der Benutzer die Information über das betrachtete Gebäude erst zwei Stunden später erhalten. Diese Erkenntnis ist natürlich sehr ernüchternd und für eine Anwendung zur blickbasierten Interaktion mit 3D-Geoobjekten nicht geeignet.

Aufgrund der in Abbildung 8.3 (S. 64) enthaltenen Diagramme, der Regressionsgerade und der Histogramme konnten die folgenden Schlüsse gezogen werden: Einerseits ist im Histogramm die Abhängigkeit der gesamten Berechnungszeit von der 3D-Verschneidung nun graphisch feststellbar. Anderseits ist ersichtlich, dass die meisten Blickberechnungen innerhalb von 150 Millisekunden ablaufen. Diese Feststellung kann auch aufgrund der statistischen Kennwerte der Tabelle 8.4 (S. 60) bestätigt werden, denn bereits nach 132 Millisekunden erfolgten 75% der Berechnungen und nach 150 Millisekunden sind dies sogar 99.76%. Daher ist anzunehmen, dass sich mit einer Aufnahmefrequenz von 150 Millisekunden keine Verzögerungen mehr ermitteln lassen und somit kann der erste Parameter für den Fixationsalgorithmus definiert werden.

Der hier gewählte Versuchsaufbau, bei welchem über eine relativ lange Zeit das Zentrum einer Gebäudewand angestarrt wird, entspricht nicht dem natürlichen Blickverhalten des Menschen. Dies hatte zur Folge, dass falls mit dem von Salvucci & Goldberg (2000) empfohlenen Winkel von 0.5° bis 1° gearbeitet wird, aus 2017 Blicken lediglich 50 Fixationen resultieren. Um diese Anzahl etwas zu erhöhen, wurde schliesslich ein Winkel von 0.25° (4Pixel) gewählt und so konnten zumindest 82 etwas kürzer andauernde Fixationen eruiert werden. Die mittlere Fixationsdauer lag dabei bei rund 850 Millisekunden.

Den durch die gewählten Parameter resultierenden Einfluss auf die kumulierte Verzögerung ist in Abbildung 8.4 nachvollziehbar. Es ist wenig erstaunlich, dass nun nach 50 Sekunden Aufnahmedauer nur noch eine aufsummierte Verzögerung von 600 Millisekunden auftritt. Dies, da anstelle von 2017 Blicken nur noch 82 Datensätze berechnet wurden.

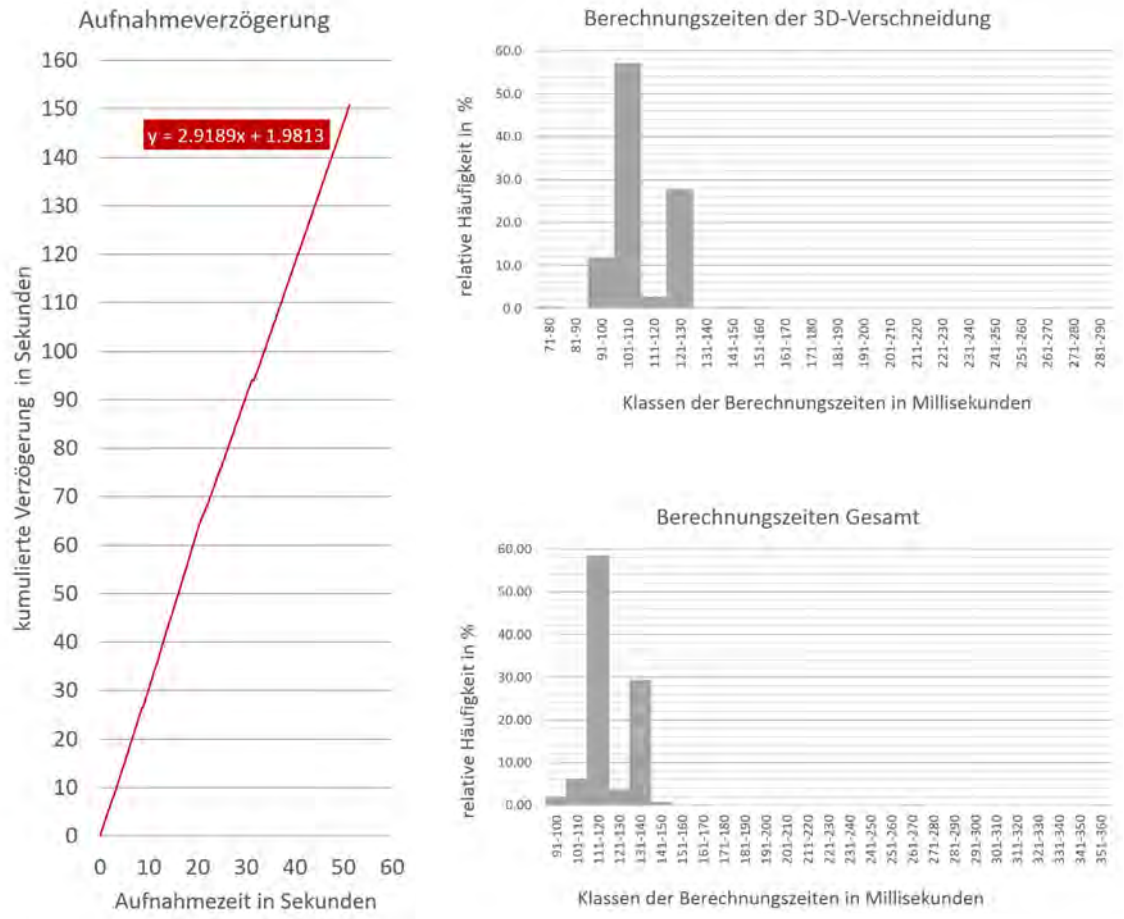

Abbildung 8.3:
Die Abbildung zeigt die auftretende Kumulation der Verzögerungen, falls die 3D-Verschneidung für jeden aufgezeichneten Blick ermittelt wird. In den Histogrammen ist die Häufigkeit der dabei auftretenden Berechnungsdauer dargestellt.

Allerdings ist es dennoch erwähnenswert, dass anhand der roten Kurve in der Abbildung immer noch eine Aufsummierung der Verzögerung feststellbar ist. Betrachtet man die Darstellung noch etwas genauer, so wird man feststellen, dass eine Kumulation immer dann stattfindet, wenn die blaue Kurve der gesamten Berechnungsdauer oberhalb der grauen Balken der Aufnahmeintervalle liegt. Die Aufnahmeintervalle entsprechen mit anderen Worten der Phase von sakkadischer Augenbewegung (siehe Kapitel 2.3 S. 11). Demgegenüber kann die Verzögerung eingeholt werden, sofern die Phase der sakkadischen Augenbewegung länger andauert als die Berechnungsdauer, also der graue Balken oberhalb der blauen Kurve liegt (siehe Abbildung 8.4 S. 65).

Die Erkenntnis aus den Ergebnissen der anfänglich gewählten Fixationsparametern wurde eingesetzt, um eine weitere Abflachung der Regressionsgerade der aufsummierten Verzögerungen zu erzielen. Dies kann nur erreicht werden, indem die Phasen der Sakkaden ver-

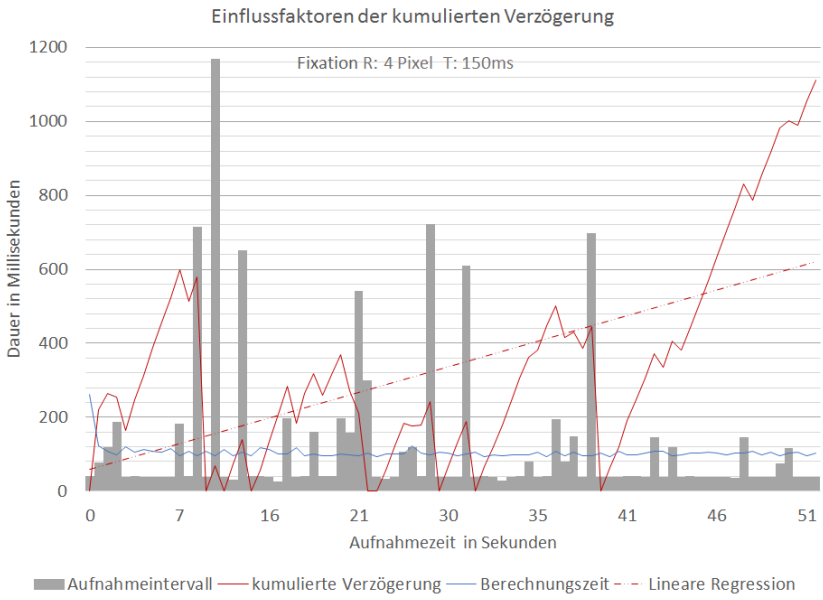

Abbildung 8.4:
Das Diagramm veranschaulicht den Zusammenhang zwischen der kumulierten Aufnahmeverzögerungen, den Aufnahmeintervallen und der gesamten Dauer der Berechnung einer 3D-Verschneidung.

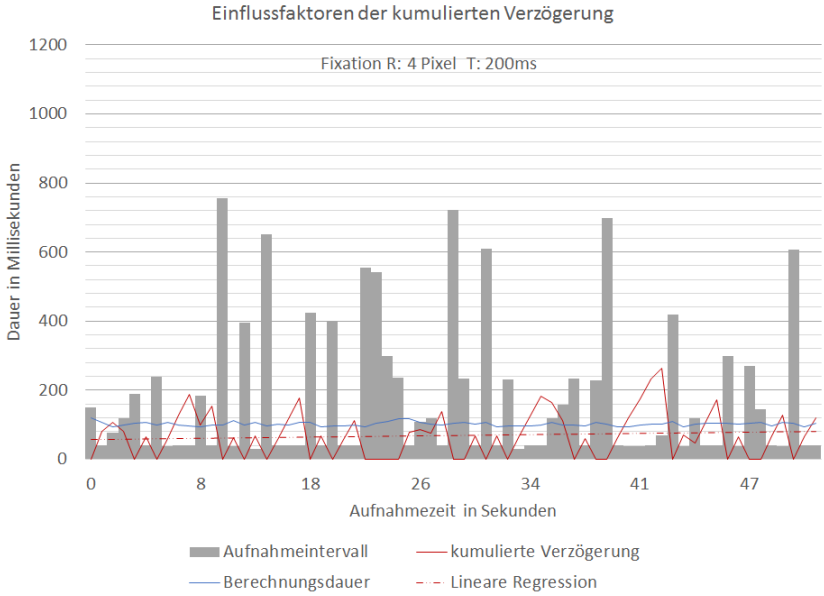

Abbildung 8.5:
Aufgrund der Verschärfungen der Auswahlkriterien der Fixationen, konnte die Steigung der Regressionsgeraden auf annähernd Null reduziert werden.

mehrt auftreten oder länger andauern, mit anderen Worten die Kriterien zur Auswahl der Fixation verschärft werden. Da der Verteilungsparameter basierend auf einem minimalen

Sichtwinkel von 0.25° gewählt wurde, musste der zweite Parameter der Fixationsdauer verschärft werden. Dies erfolgte indem dieser auf 200 Millisekunden erhöht, und anschliessend die Zeitmessung erneut durchgeführt wurde.

Die Auswirkung dieser Verschärfung sind in Abbildung 8.5 sehr anschaulich dargestellt. Einerseits wurde die Anzahl der ermittelten Fixationen von 82 auf 67 reduziert. Anderseits treten nun die Aufnahmeintervalle, welche oberhalb der blauen Kurve der Berechnungsdauer liegen, öfters auf. Tendenziell erhält man den Eindruck, dass nun die Phasen der sakkadischen Augenbewegungen homogener verteilt sind als zuvor. Noch wichtiger ist jedoch die Tatsache, dass nun kaum noch aufsummierte Verzögerungen von mehr als 200 Millisekunden auftreten und die Regressionsgerade horizontal verläuft. Daher kann gesagt werden, dass sämtliche Erwartungen an den Versuch erfüllt wurden und es daher möglich sein muss, eine blickbasierte Anwendung zur Interaktion mit 3D-Geoobjekten zu verwirklichen. Dabei kann die Interaktion annähernd in Echtzeit erfolgen ohne dass nennenswerte Verzögerungen auftreten.

8.4 Effektivität

Der Nutzen der performantesten Anwendung ist jedoch vernachlässigbar, wenn sie ihren Zweck nicht erfüllt. Die „Gebrauchstauglichkeit„ der Anwendung soll nun mit folgendem Test ausgewiesen und eingeordnet werden.

8.4.1 Ziel

Das Ziel ist es, die Effektivität der Anwendung zu evaluieren. Dazu wird ein Zielbereich – eine Art Zielscheibe – definiert, welche durch die Testperson zu betrachten ist. Die Formel (8.1) widerspiegelt das allgemein gültige Verhältnis der hier verwendeten Effektivitätsmessung und gibt die Anzahl Blicke im Zielbereich in Prozent aus.

$$\textit{Effektivität (EK)} = \frac{\textit{Anzahl Blicke im Zielbereich}}{\textit{Gesamtanzahl der erfassten Blicke}} \cdot 100\,\% \tag{8.1}$$

Mittels dem Versuch soll zudem herausgefunden werden, wie stark dieses Verhältnis durch die eingesetzten Sensoren beeinflusst wird. Der Einfluss der bekannten Schwachstellen im System – das GPS und der Orientierungssensor – kann so ebenfalls beziffert werden. Dies kann deswegen geschehen, weil die Koordinaten des Standorts der Testperson und des Ziels bekannt sind (vgl. Kapitel 8.4.2 S. 67). Demnach können entweder die von den Sensoren gemessenen Werte oder die aus den Koordinaten berechneten Werte verwendet werden. Folglich kann das beschriebene Verhältnis als erstes nur für die Video-Aufnahmen allein und danach bei der Hinzunahme jedes weiteren Sensors erneut berechnet werden.

8.4.2 Standort und Instruktionen

Abbildung 8.6:
roter Punkt: Standort der Testperson / blauer Punkt: Kalibrationspunkt (Obere Gebäudeecke) zur Eichung des Gesamtsystems / die Testperson betrachtet die Säule vom Gebäude auf Augenhöhe entlang der roten Linie

Ein Kriterium für die Zielauswahl ist, dass dieses durch die Testperson fixiert werden kann und anschliessend im Video einfach auszuwerten ist. Bei der Standortwahl hingegen war wichtig, dass die Testperson annähernd horizontal auf das Ziel blicken kann (vgl. Kapitel 8.4.3). In der Abbildung 8.6 (S. 67) befindet sich beim blauen Punkt auf Strassenniveau eine Gebäudesäule, welche für die Testperson leicht zu erkennen ist. Als Unterstützung wurde auf der Säule ein A4 Papierblatt auf Augenhöhe der Testperson angeklebt, damit die Blicke möglichst horizontal verlaufen.

Die Umgebungskamera des Eye Trackers ist sehr anfällig auf schnell wechselnde Lichtverhältnisse, wodurch überbelichtete Stellen und Unschärfen im Video entstehen. Damit dieser Einfluss etwas entschärft wird, liegen die Ziel- und Startorte relativ nahe beieinander. Damit soll auch gewährleistet sein, dass das oben erwähnte A4 Papier gut auf dem Video erkennbar ist und eine Nachkalibration und Bearbeitung mit *Dikablis-Analysis* durchgeführt werden kann.

Zur Kalibration des Systems wurde die höchstgelegene Ecke des Gebäudes über der Säule gewählt, in der Abbildung 8.6 ist diese durch einen blauen Punkt markiert. Die Koordinaten der Start–, Ziel– und Kalibrationspunkte wurden anhand der verwendeten GIS-Software aus den Geodaten gelesen. Dies war möglich, weil die erwähnten Punkte so gewählt wurden, dass sie im Feld und auf dem Orthophoto einfach identifizierbar sind. Für die ersten 30 Sekunden musste die Testperson den Kalibrationspunkt fixieren, um das Gesamtsystem zu eichen. Danach wurde sie angewiesen, das angeklebte A4 Papier zu betrachten. Die Aufnahme für den eigentlichen Versuch wurde während zwei Minuten aufgezeichnet.

8.4.3 Verwendete Analysemethode

Wie bereits erwähnt, erhielt die Testperson die Aufgabe eine ausgewählte Stelle −A4 Blatt an der Gebäudesäule− mit ihrem Blick zu fixieren. Dies führt dazu, dass nahezu alle Blicke auf den Zielpunkt fallen oder anders ausgedrückt, dass der Fokus des Probanden konstant auf der Zielstelle liegt.

Wird von einer bekannten Position aus eine Gebäudewand betrachtet, die orthogonal zur Blickachse steht, resultiert als Schnittfläche von Sichtkegel und Gebäudewand ein Kreis. Falls die Blickachse hingegen schief zur Gebäudewand steht, entsteht eine Ellipse oder eine Parabel.

Weil mit der verwendeten Software die Konstruktion eines Kegels (Ellipse) zur weiteren Analyse nicht ohne weiteres möglich war, wurde als Annäherung ein Kreis um den idealen Schnittpunkt der Blicklinie und Gebäudewand erstellt. Dies war jedoch nur zulässig, sofern die Blicke in etwa orthogonal auf die Gebäudewand treffen (vgl. Kapitel 8.4.2).

Fokussiert eine Person etwas, so sieht sie nur in einem Bereich von ca. 1° um die Blickachse scharf (Pfeiffer & Wachsmuth, 2011). Dieser scharfe Sehbereich − Sichtkegel mit einem Öffnungswinkel von 0.5° − wurde herangezogen, um den innersten Bereich der Zielscheibe zu berechnen (vgl. Formel 8.1). Der Kreisradius wurde mittels der Distanz[6] zwischen dem Personenstandort und dem A4 Papier ermittelt. Diese Berechnung wurde für die Kegelöffnungswinkel von 0.5° bis 2.5° wiederholt, so dass ein Zielscheibe entstand. Die Kennwerte der Kreise sind in der Tabelle 8.5 (S. 69) aufgeführt.

Damit das Video mit identischen Kreisradien ausgewertet werden konnte, mussten entsprechende „Area-Of-Interesst AOI“ erstellt werden. Bedauerlicherweise konnten in *D−Lab* nur Polygone als AOI erfasst werden. Daher war es als erstes notwendig − basierend auf den optischen Kennwerten der Eye Tracker Kamera − die Radien in die Pixelwerte des Feldvideos umzurechnen. So wurde es möglich, um das A4 Papier kreisförmige Video–Overlays zu erzeugen. Die Overlays dienten schliesslich als Digitalisierungshilfe zur manuellen Erfassung der AOI–Polygone und konnten so für die Berechnung der Effektivität genutzt werden. Diese wiederum kann bei der Anwendungsauswertung als Nullmessung eingesetzt werden. Es ist hier noch zu erwähnen, dass dabei jegliche Blicke im Zielbereich gezählt werden, selbst wenn diese das Gebäude nicht „wirklich schneiden“. Als Analysesoftware wurde *D−Lab* verwendet (vlg. Kapitel 8.1.2 S. 53).

Die eigentliche Anwendungsauswertung erfolgt in *ArcScene 10.1*, indem um die Koordinate des A4 Blattes die jeweiligen 3D–Buffer erstellt wurden. Die Kugelradien (3D–Buffer)

[6] Die in ArcInfo 10.1 gemessene Entfernung zwischen Zielort und Standort der Testperson beträgt 75.5m

entsprechen dabei den in Tabelle 8.5 aufgelisteten Angaben. Die *Anzahl der Blicke im Zielbereich* sind dabei alle Schnittpunkte zwischen Blicklinien und 3D-Gebäude, welche innerhalb der Kugel liegen. Für die Berechnung der Effektivität ($EK_{Anwendung}$), wurden ebenfalls alle Blicke innerhalb der Kugel verwendet, selbst wenn sie kein Gebäude schnitten. Dazu war es erforderlich, das Zielgebäude im 3D-Stadtmodell dementsprechend zu verfälschen.

Gemäss der Zielsetzung des vorliegenden Versuchs, soll der Beitrag der einzelnen Sensoren zur erzielten Effektivität $EK_{Anwendung}$ ausgewiesen werden. Deswegen werden für die Berechnung des Verhältnisses $EK_{Anwendung\ ohne\ Sensoren}$ die aus den Ziel- und Startkoordinaten erhaltenen Azimut- und Positionswerte eingesetzt. Anschliessend wird das Verhältnis unter der Verwendung der Azimutmessung vom Smartphone $EK_{Anwendung\ mit\ Azimut}$ berechnet, ohne die Verwendung vom GPS zur Positionsermittlung. Zum Schluss erfolgt die Berechnung mit der Positionsangabe vom Smartphone $EK_{Anwendung\ mit\ GPS}$, jedoch ohne die Verwendung von der Azimutmessung.

Tabelle 8.5:
In der Tabelle sind die Kreisgrössen aufgelistet, welche als Zielbereiche in die Effektivitätsberechnung einfliessen.

Parameter der Sichtkegel bei einer Sichtdistanz von 75 Metern					
Öffnungswinkel (°)	0.51	1.03	1.54	2.05	2.56
Durchmesser (m)	1.343	2.686	4.028	5.371	6.714
Durchmesser (Pixel)	8	16	24	32	40

8.4.4 Erwartungen

Falls die Testperson beim Feldversuch konzentriert ist und die Eye Tracker Kalibration gut gelang, sollte beim Eyetracking-Video das Verhältnis EK_{Video} nahezu bei 100% liegen. Das Verhältnis $EK_{Anwendung\ ohne\ Sensoren}$ müsste annähernd identisch sein mit dem Erstgenannten. Abweichungen werden aufgrund der Kopfposition der Testperson und der Erfassungsgenauigkeit der verwendeten Geodaten erwartet. Durch die Hinzunahme weiterer Sensoren müsste der EK-Wert signifikant schlechter ausfallen, d.h. näher bei Null liegen. Durch den Versuch wird eine Bestätigung der Ergebnisse von Mosimann (2013, S. 47) erwartet, welche aussagten, dass das Smartphone die Schwachstelle des Systems ist. Nebst einer Angabe der vertikalen Abweichung ermöglicht die gewählte Versuchsmethode auch eine Angabe über die horizontale Abweichung. Weiter wird vermutet, dass der GPS-Sensor als schwächstes Glied des Gesamtsystems ausgewiesen werden kann.

8.4.5 Ergebnisse

Bevor mit der Besprechung der Ergebnisse begonnen wird, muss erwähnt werden, dass die Kalibration des Gesamtsystems nicht optimal verlief. Die Schwierigkeit war, dass die Auflösung des Feldvideos zu gering war und darüber hinaus die Aufnahme mit Gegenlicht erfolgte. Dadurch konnte das Video nicht nachkalibriert werden, weil der Kalibrationspunkt auf dem Video kaum erkennbar war. Damit das Gesamtsystem dennoch geeicht werden konnte, wurden die Koordinaten des A4 Papiers als Kalibrationspunkt eingesetzt. Dies hatte wahrscheinlich zur Folge, dass die Effektivitätswerte tendenziell zu gut erscheinen.

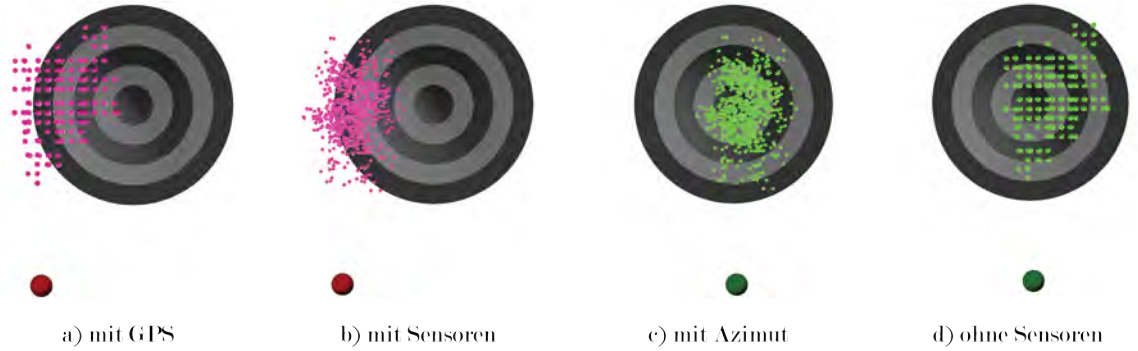

Abbildung 8.7:
In der Abbildung ist der um das A4 Papier erstellte Zielbereich ersichtlich. Der äusserste Kreis weist einen Durchmesser von 6.70m auf, der innerste einen von 1.30m. Weiter ist der Einfluss der verschiedenen Sensoren auf die Lage der Schnittpunkte feststellbar.

In Abbildung 8.7 sind die Blickschnittpunkte auf den verwendeten Zielbereichen dargestellt. Die pinken Punkte unterhalb des Zielbereichs repräsentieren die gemessene GPS–Position, die grünen Punkte den tatsächlichen Standort der Testperson. Es ist sofort ersichtlich, dass unter der Verwendung des GPS nahezu keine Blickpunkte im innersten Zielkreis liegen. Aufgrund der ungenauen GPS–Messung wurde die gesamte Punktwolke um rund 2.50m nach links verschoben.

Als der Höhenwinkel und das Azimut nicht durch das Smartphone ausgegeben wurden, sind die Schnittpunkte gitterartig angeordnet. Dies weil sich dadurch die Winkelausgaben des „Sensors“ nicht mehr ändern, die Winkelunbekannten null betragen und somit nur die Pixelkoordinaten zur Blickrichtungsberechnung genutzt werden. Deswegen ist auf der Zielscheibe die Rasterauflösung des Feldvideos zuerkennen (siehe Abbildung 8.7 a) und d)).

Weiter ist auffällig, dass ohne Einsatz der Winkelmessung die Blicke tendenziell breiter gestreut sind, zumindest in horizontaler Richtung. Diese Vermutung konnte durch statistische Kennwerte bestätigt werden: Ohne die Verwendung von Sensoren betrug die Standardabweichung des Azimuts $\sigma = 0.56°$ und beim Höhenwinkel $\sigma = 0.54°$. Durch den Einsatz der Orientierungssensoren veränderte sich die Standardabweichung des Azimuts auf $\sigma = 0.46°$,

beim Höhenwinkel allerdings auf $\sigma = 0.68°$. Die gemessenen Unterschiede dürfen jedoch nicht auf die Genauigkeit der Sensoren zurückgeführt werden, weil trotz einer Kopfbewegung der Blick weiterhin auf denselben Punkt fixiert bleiben kann. Dies erklärt auch die Tatsache, wieso die Effektivitätswerte $EK_{mit\ AZ}$ in Tabelle 8.6 (S. 71) durchwegs am Besten ausgefallen sind.

Wie in Tabelle 8.6 zu erkennen ist, belegt die Effektivität EK_{Video} entgegen den Erwartungen nicht den ersten Rang. Dies hat damit zu tun, dass die kreisförmigen *„Areas of Interst's"* nur mit ungenügender Qualität digitalisiert werden konnten. Daher sind die auf Basis der 3D–Buffern ermittelten Werte höher ausgefallen.

Tabelle 8.6:
In der Tabelle sind die ermittelten Effektivitätswerte ersichtlich. In den letzten zwei Spalten ist die mittlere horizontale und vertikale Abweichung aufgelistet, dabei ist jedem Mittelwert die entsprechende Standardabweichung angefügt.

	Sichtkegeldurchmesser in Pixel						
	8	**16**	**24**	**32**	**40**	**⌀Δ H**	**⌀Δ V**
$EK_{mit\ AZ}$	25.0	76.6	94.9	99.7	100.0	–0.30° ±0.46°	0.11° ±0.68°
$EK_{ohne\ Sensoren}$	26.0	58.4	92.3	99.3	100.0	–0.58° ±0.56°	–0.22° ±0.54°
EK_{Video}	19.2	45.8	76.9	93.5	97.7		
$EK_{mit\ GPS}$	1.6	5.0	26.6	62.7	84.3	1.79° ±0.56°	–0.19° ±0.54°
$EK_{mit\ Sensoren}$	0.0	0.5	8.6	37.1	76.9	2.08° ±0.46°	0.14° ±0.68°
$EK_{mitS\ Fix8200}$	0.0	0.0	0.6	24.1	71.1	2.14° ±0.41°	0.05° ±0.74°

Zumindest wurde die Erwartung erfüllt, dass durch den Einsatz des GPS–Sensors die Trefferquote der Blicke stark abnimmt. Bereits beim grössten Kreis mit einem Durchmesser von rund 7m, verfehlen 15.7% der berechneten Blicke ihr Ziel. Je kleiner die Zielkreise werden, desto höher fällt tendenziell die Abweichung aus.

Wie bereits erwähnt hat die Winkelmessung einen positiven Einfluss auf die Effektivitätswerte, aber nur sofern der Personenstandort korrekt ausgegeben wird. Ist dies nicht der Fall, kann sich die aufgrund einer Kopfbewegung getätigte Augenbewegung negativ auf die Genauigkeitsmessung auswirken. Weil beispielsweise die Blickrichtung nicht mehr aufs Ziel zeigt, sondern vom Ziel weg. Daher ist die Effektivität mit dem Einsatz aller Sensoren zwischen 8 bis 25% niedriger ausgefallen als die $EK_{mit\ GPS}$ Werte.

Zur Berechnung des letzten Effektivitätswerts wurden die aus den Fixationen ermittelten Sichtvektoren verwendet. Als Fixationskriterium wurde ein Verteilungsradius von 8 Pixel und eine minimale Dauer von 200ms eingesetzt. Damit erhoffte man sich, dass Ausreisser

weniger stark ins Gewicht fallen und sich folglich die Genauigkeit erhöhen würde. Wie sich zeigte war – aufgrund des im vorherigen Abschnitt beschriebenen Effektes – das Gegenteil der Fall.

Die mittleren vertikalen Abweichungen liegen in einer ähnlichen Grössenordnung wie diejenigen von Mosimann (2013, S. 39 ff), welche sich zwischen –2.33° (–0.70°) und 0.21° befanden, mit einer Standardabweichung zwischen 0.41° und 2.04° (0.83°). Wobei der Mittelwert von –2.33 mit der entsprechenden Standardabweichung von 2.04° wurde aus einem Feldtest berechnet, bei welchem stärkere Kopfbewegungen erfolgten. Deswegen sind für einen Vergleich eher die Werte von –0.70° und 0.83° heranzuziehen.

8.5 Map-Matching

Bereits vor der Durchführung der Versuchsreihe wurde angenommen, dass der GPS-Sensor eine zentrale Schwachstelle des Gesamtsystems ist. Mit diesem Versuch soll nun getestet werden, ob die GPS-Genauigkeit durch Methoden von „Map-Matching“ erhöht, und eine signifikante Verbesserung des in Kapitel 8.4 definierten EK-Wertes erzielt werden kann.

Viele Personen kennen Map-Matching-Methoden aus dem Alltag, oftmals ohne sich dessen bewusst zu sein. Diese werden nämlich in jeglichen Navigationsgeräten im Strassenverkehr eingesetzt, dabei wird das GPS-Signal durch verschiedene Methoden korrigiert und auf die gegenwärtig wahrscheinlichste befahrene Strasse gesetzt. Im Versuch werden keine hochstehenden Map–Matching Algorithmen verwendet, sondern die GPS-Position wird im Postprocessing einfach auf die nächstgelegene Strasse gesetzt.

8.5.1 Ziel

Mit dem Versuch soll die Auswirkung der eingesetzten Map-Matching-Methode auf die Effektivität des Systems quantifiziert werden (vgl. Abb. 8.8 S. 73). Die Effektivitätswerte werden dabei für vier Szenarien betrachtet, dabei befindet sich die Testperson[7] ...

1. ... auf der Strassenachse quer zur Blickrichtung
2. ... auf der Strassenachse längs zur Blickrichtung
3. ... am Strassenrand (Strassenbreite $<$ GPS-Genauigkeit)
4. ... am Strassenrand (Strassenbreite $>$ GPS-Genauigkeit)

Für die zwei Standortvarianten am Strassenrand wird jeweils nur von einer Blickrichtung längs zur Strassenachse ausgegangen. Ein Ziel ist es, den möglichen Einsatzbereich der gewählten Map-Matching-Methode auszuweisen und die Grenzen der Methode zu eruieren.

[7] GPS-Genauigkeit: durch das Smartphone ausgegebener Wert der GPS-Accuracy [m]

Abbildung 8.8:
Der rote Punkt ist jeweils die Position der Testperson in beiden Bildern.
Die rote Linie in der linken Abbildung entspricht der Blickrichtung.
Im rechten Bild sind die orangen Linien die hypothetischen Strassenachsen. Bei der Betrachtung der Abbildung ist der Standort der Testperson jeweils mit nur einer einzelnen Strassenachse zu vergleichen. So wird ersichtlich, wie dadurch eine relative Standortveränderung der Testperson herbeigeführt wird: a) auf die Strassenachse mit Blickrichtung quer dazu, b) auf die Strassenachse mit Blickrichtung längs dazu oder c) auf den Strassenrand mit Blickrichtung längs dazu

8.5.2 Standort und Instruktionen

Für den Versuch werden keine eigenständigen Feldaufnahmen durchgeführt, sondern die Daten aus dem Effektivitätstest verwendet. Anstatt den Standort des Probanden zu verändern, werden die hypothetischen Strassenachsen verlegt (vgl. Abb. 8.8 S.73). Daher gelten für den Map-Matching-Versuch dieselben Bedingungen und Instruktionen wie in Kapitel 8.4.2 (S. 67) beschrieben.

8.5.3 Verwendete Analysemethode

Zur Weiterverarbeitung werden die Feldaufnahmedaten in ein Desktop-GIS importiert. Durch die Anwendung spezifischer GIS-Werkzeuge, wird die durch das Smartphone gemessene Position mit der neu berechneten Position ersetzt. Beim gewählten Verfahren bleibt die Datenstruktur erhalten, so dass die Effektivitätswerte im Anschluss anhand der erstellten Anwendung erzeugt werden können. Eine genauere Beschreibung der verwendeten Methode erfolgt im nächsten Abschnitt. Die Effektivitäswerte werden, gemäss Formel 8.1 (S. 66) berechnet.

Zur Berechnung der neuen Position wird das Werkzeug „Near“ der ArcGIS-Toolbox [8] verwendet. Es berechnet in erster Linie die Distanzen aller Objekte einer *Feature Class X* zum nächstgelegenen Objekt einer zweiten *Feature Class Y*. In dem hier beschriebenen Fall wird also die Distanz der GPS-Position des Smartphones zur nächstgelegenen Strassenach-

[8] http://resources.arcgis.com/de/help/main/10.1/

se berechnet. Das erwähnte Geoverarbeitungswerkzeug bietet jedoch noch eine zusätzliche optionale Funktion an, welche hier speziell von Interesse ist: Wird vor der Ausführung des Werkzeugs die Option *Location* markiert, wird neben der Distanz auch die Koordinate des nächstgelegenen Punktes der Strassenachse ausgegeben. Dieser Punkt wird die ursprüngliche GPS-Position des Smartphones ersetzen. Die restlichen Verarbeitungsschritte wie beispielsweise „*DeleteField*“ dienen alleine dazu, die ursprüngliche Datenstruktur wiederherzustellen.

Die Berechnungen werden für die vier definierten Szenarien gemäss Kapitel 8.5.1 wiederholt und folgende Effektivitätswerte unter der Verwendung der Map-Matching-Methode ausgewiesen[9]:

Tabelle 8.7:
In der Tabelle sind die Abkürzungen der Effektivitätswerte je Szenario aufgelistet.

Effektivitätswert	**Bezeichnung**
$EK_{StA\ quer}$	Standort auf Strassenachse mit Blickrichtung quer dazu
$EK_{StA\ laengs}$	Standort auf Strassenachse mit Blickrichtung längs dazu
$EK_{StR\ schmal}$	Standort am Strassenrand auf schmaler Strasse
$EK_{StR\ breit}$	Standort am Strassenrand auf breiter Strasse

8.5.4 Erwartungen

Map-Matching-Methoden können die Effektivitätswerte verbessern, solange die Strassenachse und die Blickrichtung mehr oder weniger parallel verlaufen. Der erwartete Einfluss der Methode ist in der Abbildung 8.9 (S. 75) visualisiert. Je näher der Standort der Testperson auf der Strassenachse liegt – auf welche die gemessenen GPS-Positionen verschoben werden – desto exaktere Effektivitätswerte sind zu erwarten. Aufgrund der Verschiebung der Positionen, wird sich die Streuung der nun korrigierten Messwerte stark verändern. Bei den Testfällen der Abbildung 8.9 a) und b) wird eine höhere Effektivität des Systems erwartet. Je breiter die Strasse, desto geringer ist die Verbesserung der Effektivitätswerte (vgl. Abb. 8.9 c)).

Ist die Blickrichtung hingegen quer zur Strassenachse, ist in der Abbildung 8.9 (d-f) erkennbar, dass die gewählte Map-Matching-Methode keinen Einfluss auf die Effektivität haben wird. Deswegen wird davon ausgegangen, dass bei den Effektivitätswerten und der Streuung der Messungen keine signifikanten Änderungen auftreten. Dies ist der Grund, wieso für die Blickrichtung quer zur Strassenachse nur ein Testfall untersucht wurde.

[9] Strassenbreite ist schmaler/breiter als der Durchmesser des GPS-Genauigkeitskreises

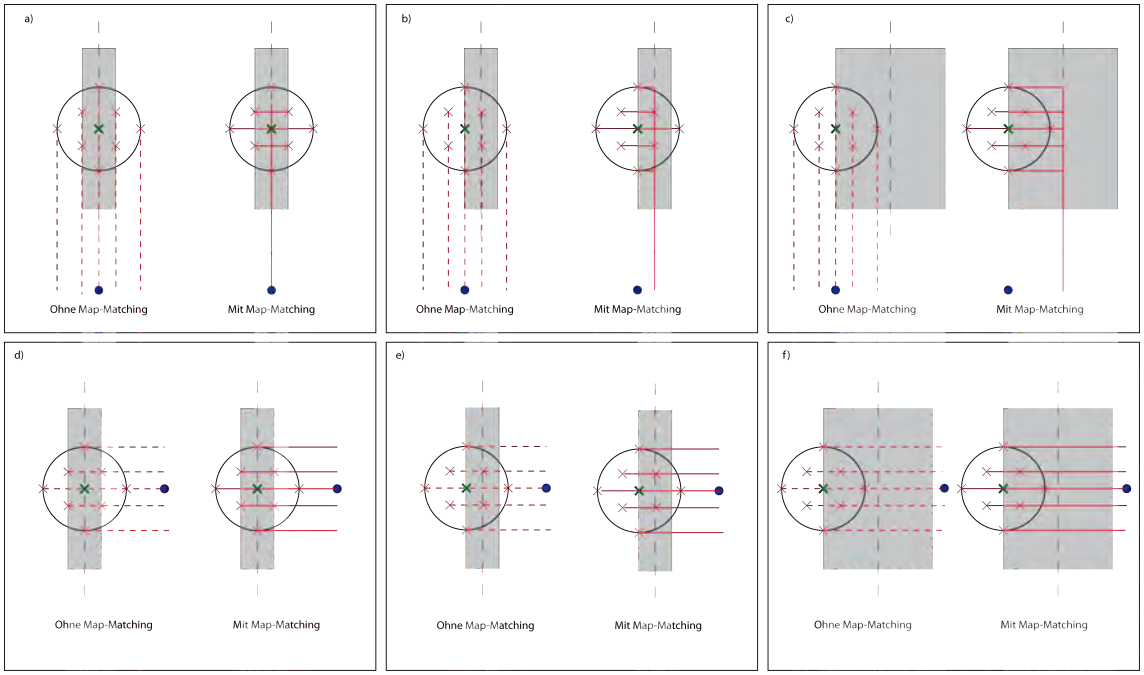

Abbildung 8.9:
Auf den Abbildungen ist jeweils eine graue Strasse und ihre schwarz gestrichelte Achse dargestellt. Das grüne Kreuz steht für den effektiven Standort der Testperson, der blaue Punkt für den betrachteten Zielbereich. Der Kreis repräsentiert den Genauigkeitsbereich der GPS-Messung, die roten Kreuze entsprechen potentiell gemessene GPS-Positionen. Die Darstellungen stellen den Map-Matching Effekt mit einer Blickrichtung längs (a-c) respektive quer (d-f) zur Strassenachse dar.

8.5.5 Ergebnisse

In Abbildung 8.10 ist der Einfluss der Map–Matching Methode ersichtlich. Die Punktwolken wurden dadurch neu über den entsprechenden Strassenachsen ausgerichtet. Die orange Punktwolke entspricht dabei den Schnittpunkten, bei welchen die Blickrichtung quer zur Strassenachse verläuft und beruht somit auf den tatsächlich von den Sensoren gemessenen Werten. Falls sich die Testperson auf der Strassenachse befindet und die Blickrichtung entlang der Strasse verläuft, resultiert die grüne Punktwolke zentral auf dem Zielbereich. Die blauen und braunen Schnittpunkte entstehen, wenn sich die Testperson am Rand einer schmalen, beziehungsweise auf einer breiten Strasse befindet. Dieses Ergebnis deckt sich visuell betrachtet mit den Erwartungen an den Versuch.

Betrachtet man allerdings die Tabelle 8.8 mit den gemessenen Ergebnissen, so wird man feststellen, dass die Erwartungen an den Versuch nur teilweise erfüllt wurden. Durch die Verschiebung der GPS–Position auf die Strassenachse konnte das Ergebnis wie vermutet um 25% verbessert werden. Stand die Testperson jedoch am Rand einer schmalen Stras-

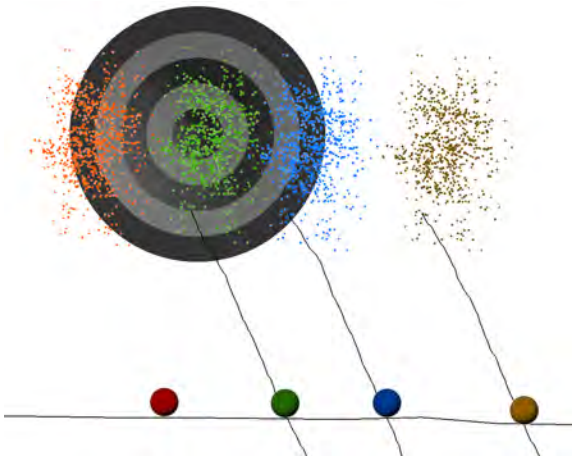

Abbildung 8.10:
Die Abbildung zeigt die aus dem Versuch Map–Matching resultierende Verteilung der Schnittpunktwolke. Der rote Punkt entspricht der gemessenen GPS–Position beziehungsweise dem Blickverlauf quer zur Strassenachse. Beim grünen Punkt stand die Person auf der Strassenachse, beim blauen und braunen Punkt am Strassenrand.

se[10] wurde das Ergebnis allerdings sogar um 10% verschlechtert. Bei einer breiten Strasse[11] lag kein einziger Blick im Zielbereich und dabei betrug die horizontale Abweichung –4.93° ±0.45°. Daher ist der Einfluss von Map–Matching Methoden höchstens in einem schmalen Bereich von kleiner als 2.5m positiv und dies auch nur mit entsprechend ausgerichteter Blickrichtung. In allen andern Fällen hatte die Methode keinen Einfluss oder wirkte sich sogar negativ auf das Ergebnis aus.

Tabelle 8.8:
In der Tabelle sind die ermittelten Effektivtätswerte ersichtlich. In den letzten zwei Spalten ist die mittlere horizontale und vertikale Abweichung aufgelistet, dabei ist jedem Mittelwert die entsprechende Standardabweichung angefügt.

	Sichtkegeldurchmesser in Pixel						
	8	**16**	**24**	**32**	**40**	**⌀Δ H**	**⌀Δ A**
$EK_{StA\ laengs}$	25.68	76.65	95.14	99.70	100.00	–0.28° ±0.46°	0.11° ±0.68°
EK_{Video}	19.16	45.84	76.94	93.49	97.67		
$EK_{StA\ quer}$	0.00	0.55	8.64	37.07	76.87	2.08° ±0.46°	0.15° ±0.68°
$EK_{StR\ schmal}$	0.00	0.00	3.72	24.06	66.12	–2.26° ±0.46°	0.26° ±0.68°
$EK_{StR\ breit}$	0.00	0.00	0.00	0.00	0.00	–4.93° ±0.45°	0.28° ±0.68°

Über die Veränderung der Streuung der GPS–Messwerte, die aufgrund der Map–Matching Methode erwartet wurde, kann hier keine Aussage gemacht werden. Die Begründung liegt

[10] 2.5m Strassenbreite $<$ 5m gemessenen GPS-Genauigkeit
[11] 6.3m Strassenbreite $>$ 5m gemessenen GPS-Genauigkeit

vermutlich in der Applikation, die auf dem Smartphone nicht laufend GPS-Werte ausgab, sondern nur wenn die Bewegungssensoren einen bestimmten Grenzwert überschritten haben. Nur so war es erklärbar, dass über 80 Sekunden lang bis auf die 15te Nachkommastelle identische Positionen ausgegeben wurden. Bei künftigen Versuchen sollte unbedingt darauf geachtet werden, dass das GPS laufend neue Werte ausgibt. Diese Tatsache hat die oben aufgelisteten Ergebnisse damit stark beeinflusst.

8.6 Häufigkeit der Serviceantworten

Die vorangegangen Versuche dienten zur Bewertung der Effektivität und der Effizienz der erstellten Systemarchitektur. Der letzte Versuch zielt auf die Bewertung der *Usability* der Anwendung ab und wird erste Indizien für künftige Anpassungen des Systems liefern.

8.6.1 Ziel

Das Ziel ist es, eine Angabe darüber zu machen, ob die erstellte Anwendung den Benutzer mit ungewollter Information überflutet oder ob er überhaupt die Information erhält, die angefordert wird. Dazu wird eine Gesprächssituation simuliert, bei welcher die Testperson in vorgegebenen Intervallen Informationen über ein Gebäude abrufen muss.

8.6.2 Standort und Instruktionen

Der Standort und die Kalibrationspunkte sind erneut so zu wählen, dass sie auf dem Orthophoto sowie im Felde einfach aufgefunden werden können. Die Testperson hat sechs Gebäude in vordefinierter Reihenfolge zu betrachten (vgl. Abb. 8.11 S.78). Wichtig ist dabei, dass die Gebäude nicht an den äusseren Begrenzungen betrachtet werden, sondern in deren Zentrum, sodass die Anwendung mit grosser Wahrscheinlichkeit die korrekte Antwort liefert.

Während des Versuchs wird die Testperson durch den Versuchsleiter in ein Gespräch verwickelt; dabei hat sie verschiedene, nicht relevante Fragen zu beantworten. Nach 10 Sekunden Gesprächsdauer erhält die Testperson ein akustisches Signal –hier das Wort „bing“ – welches sie anweist, das Gebäude mit der Nummer 1) für eine Sekunde anzuschauen. Danach folgen wieder 10 Sekunden Gesprächsdauer bis das akustische Signal erneut ertönt und das Gebäude mit der Nummer 2) für eine Sekunde betrachtet werden muss. Dies wird solange wiederholt bis alle sechs Gebäude betrachtet wurden. Der beschriebene Zyklus wird insgesamt für folgende Betrachtungszeiten wiederholt: eine Sekunde, drei Sekunden und für sechs Sekunden.

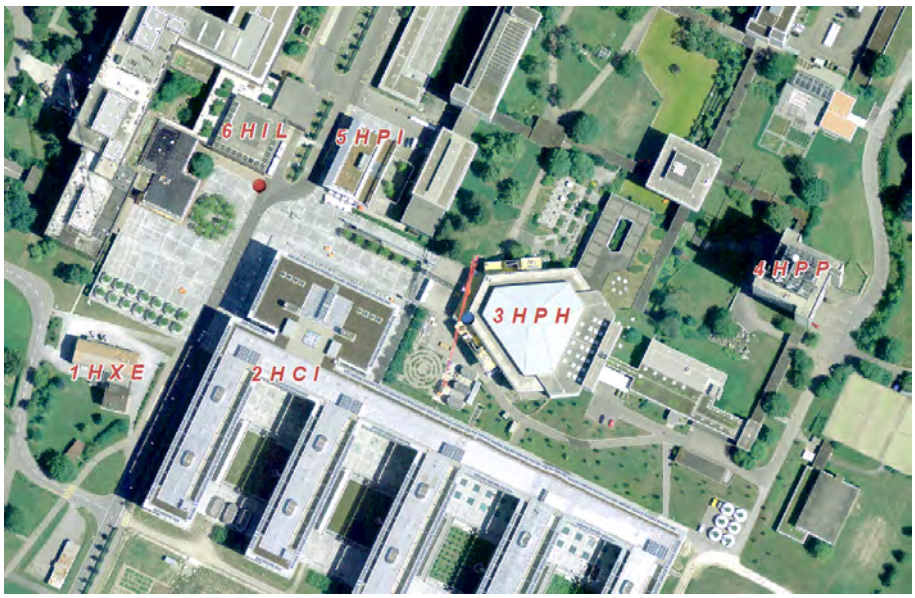

Abbildung 8.11:
Die Testperson befindet sich beim roten Punkt und das Gesamtsystem wird mittels dem blauen Punkt kalibriert. Die beschrifteten Gebäude sind von der Testperson –beim Ertönen des definierten Kommandos– in Reihenfolge aufsteigender Nummerierung zu fixieren.

8.6.3 Verwendete Analysemethode

Mit den aus dem Versuch „Verzögerung“ (S. 61) gewonnenen Erkenntnissen, werden die Selektionskriterien im Fixationsalgorithmus festgelegt. Als minimale Fixationsdauer werden daher 150 Millisekunden gewählt, als Verteilungsradius 16 Pixel. Die Absicht besteht darin, dadurch die Kumulation der Verzögerung zu vermeiden. Die durch den Fixationsalgorithmus ausgegebenen Blickvektoren werden anschliessend als Input in die Client–Anwendung eingegeben. So können die Schnittpunkte und die Gebäudeinformation ermittelt werden.

Die so gewonnenen Daten werden anschliessend in Excel ausgewertet. Dabei werden die Häufigkeiten der angeforderten und somit relevanten Informationen, den nicht angeforderten Meldungen gegenübergestellt. Die Auswertung erfolgt, indem nur Serverantworten berücksichtigt werden, die während einer, drei oder sechs Sekunden unverändert blieben. Als Nebenprodukt des Versuches werden erneut die Verzögerungen gemessen, weil hierbei das Verhalten unter „natürlicheren “ Bedingung ermittelt werden kann.

8.6.4 Erwartungen

Durch den Versuchsaufbau würde der Benutzer idealerweise sechsmal eine Information über das betrachtete Gebäude erhalten, jede zusätzlich erhaltene Information ist zu viel. Durch die simulierte Gesprächssituation der Testperson mit dem Versuchsleiter wird erwartet, dass die Blicke möglicherweise ungewollt auf ein Gebäude treffen und dadurch überflüs-

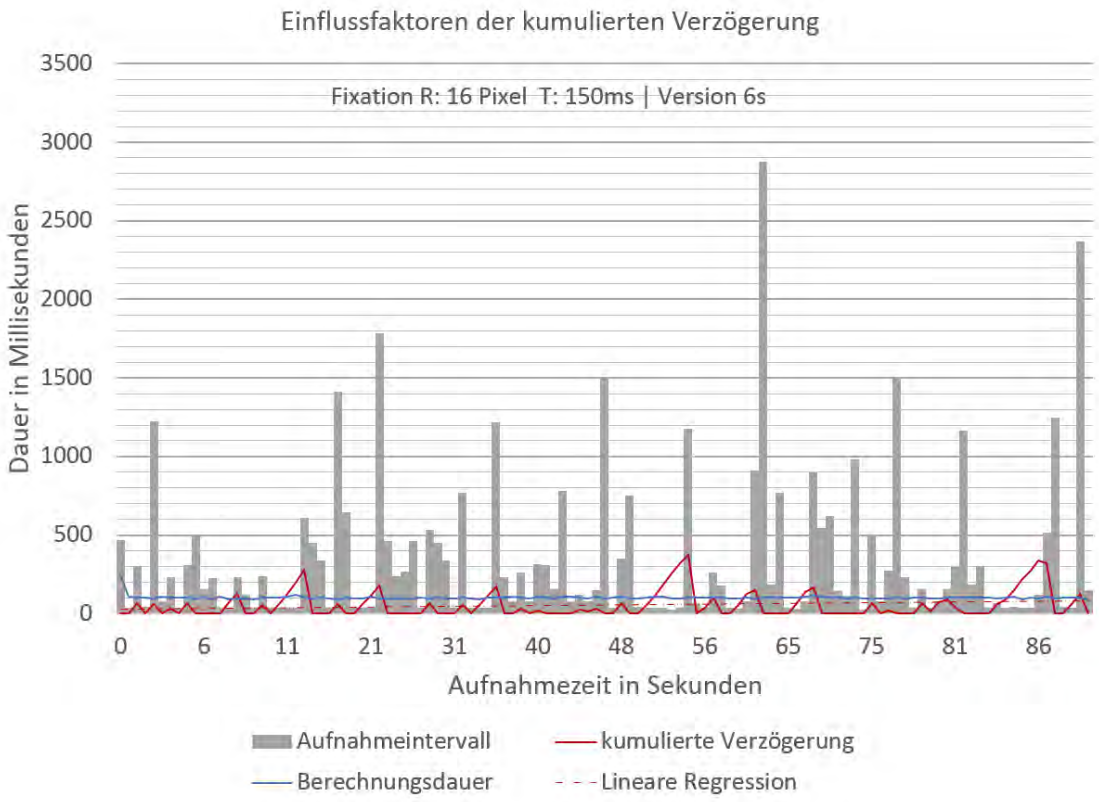

Abbildung 8.12:
In der Abbildung sind die kumulierten Verzögerungen dargestellt, die auftraten während die Gebäude zwischen drei und sechs Sekunden lang betrachtet wurden.

sige Information liefert. Bei den drei Zyklen wird erwartet, dass sich mit zunehmender Betrachtungszeit die Anzahl der Serverantworten der Idealen annähert.

8.6.5 Ergebnisse

Erfreulicherweise traten mit den festgelegten Auswahlkriterien der Fixationen kaum kumulierte Verzögerungen auf. Die Grafik in Abbildung 8.12 unterstützt diese Beurteilung anschaulich. Die These aus den Versuchen in Kapitel 8.3, dass eine Anwendung zur blickbasierten Interaktion mit Geoobjekten in annähernd Echtzeit möglich ist, wird bestätigt. Die maximale Kumulation betrug rund 360 Millisekunden, wobei die lineare Regressionsgerade dennoch horizontal verläuft. In einer bemerkenswerten Regelmässigkeit traten die Aufsummierungen immer dann auf, wenn ein Gebäude während mindestens drei Sekunden im Zentrum betrachtet wurde. Dieses doch sehr unnatürliche Blickverhalten führt dazu, dass eine Fixation nach der andern erkannt wird und die Intervalle dazwischen lediglich 37 bis 45 Millisekunden betrugen und der Aufnahmefrequenz der Feldkamera entsprechen.

In Abbildung 8.13 sind die Häufigkeiten der vom Client erhaltenen Gebäudeinformatio-

nen ersichtlich. Bei der Videoauswertung musste festgestellt werden, dass das menschliche Zeitempfinden subjektiv ist und die Gebäude nie über die vorgeschriebene Dauer betrachtet wurden. Folglich mussten die Grenzwerte der Betrachtungsdauer nach unten korrigiert werden. Dazu wären mehrere Varianten möglich gewesen, beispielsweise hätte jeweils die kürzeste oder die mittlere Betrachtungsdauer eingesetzt werden können. Hier wählte man den Ansatz die Grenzwerte zu halbieren. Dies hatte den Vorteil, dass bis auf eine, sämtliche Betrachtungsphasen oberhalb der Grenzwerte lagen und die Diagramme einfacher zu interpretieren sind.

Im Versuch mit der Betrachtungsdauer von 0.5 Sekunden wurden nur drei von sechs angeforderten Informationen geliefert. Wieso keine Information geliefert wurde, kann mittels den Serverantworten unterhalb der Grenzwerte erkannt werden (siehe Abb. 8.13). Konkret stellt sich die Frage, ob während der Betrachtung des Gebäudes überhaupt eine Fixation erkannt wurde oder ob lediglich die kumulierte Fixationsdauer zu kurz war. Bei der Betrachtung des HPH– und des HPP–Gebäudes wurden zwar Fixationen berechnet, allerdings hat der so ermittelte Blick das Ziel verfehlt und deshalb eine falsche Information geliefert. Bei der Betrachtung des HPI Gebäudes wäre der korrekte Gebäudename geliefert worden, jedoch war die Fixationsdauer unterhalb des Grenzwertes. Insgesamt war das Verhältnis von erwünschten zu nicht angeforderten Antworten 33% zu 66%.

Beim Zyklus mit der Betrachtungsdauer von 1.5 Sekunden wurden von sechs erwarten Informationen fünf korrekt berechnet. Bei der Betrachtung des HPH–Gebäudes wurden drei kumulierte Fixationsblöcke erkannt, zwei davon haben den richtigen Gebäudenamen geliefert und einer hat das Ziel knapp verfehlt. Im konkreten Fall wurden die zwei Fixationsblöcke mit den korrekten Antworten durch denjenigen mit der falschen Antwort unterbrochen weswegen diese den Grenzwert nicht erreichten. Durch den höheren Grenzwert hat sich das Verhältnis angeforderte Information - nicht angeforderte Information verbessert und beträgt nun 56% zu 44%.

Als die Gebäude während 3 Sekunden betrachtet wurden, lieferte der Server fünf Antworten. Davon enthielten vier die gewünschte Information und eine Fixation lieferte keinen Schnittpunkt, da das Gebäude verfehlt wurde. Als der Fokus auf dem HPP-Gebäude lag, wurde ein richtiges Verschneidungsergebnis geliefert aber die Betrachtungsdauer war zu kurz. Das Verhältnis von angeforderten zu überflüssigen Informationen verbesserte sich erneut und liegt bei rund 83% zu 17%. Zusammenfassend kann gesagt werden, dass durch die Erhöhung der Schwellwerte der grösste Teil der nicht angeforderten Informationen ausgeschlossen werden konnte und die Erwartungen an den Versuch erfüllt wurden.

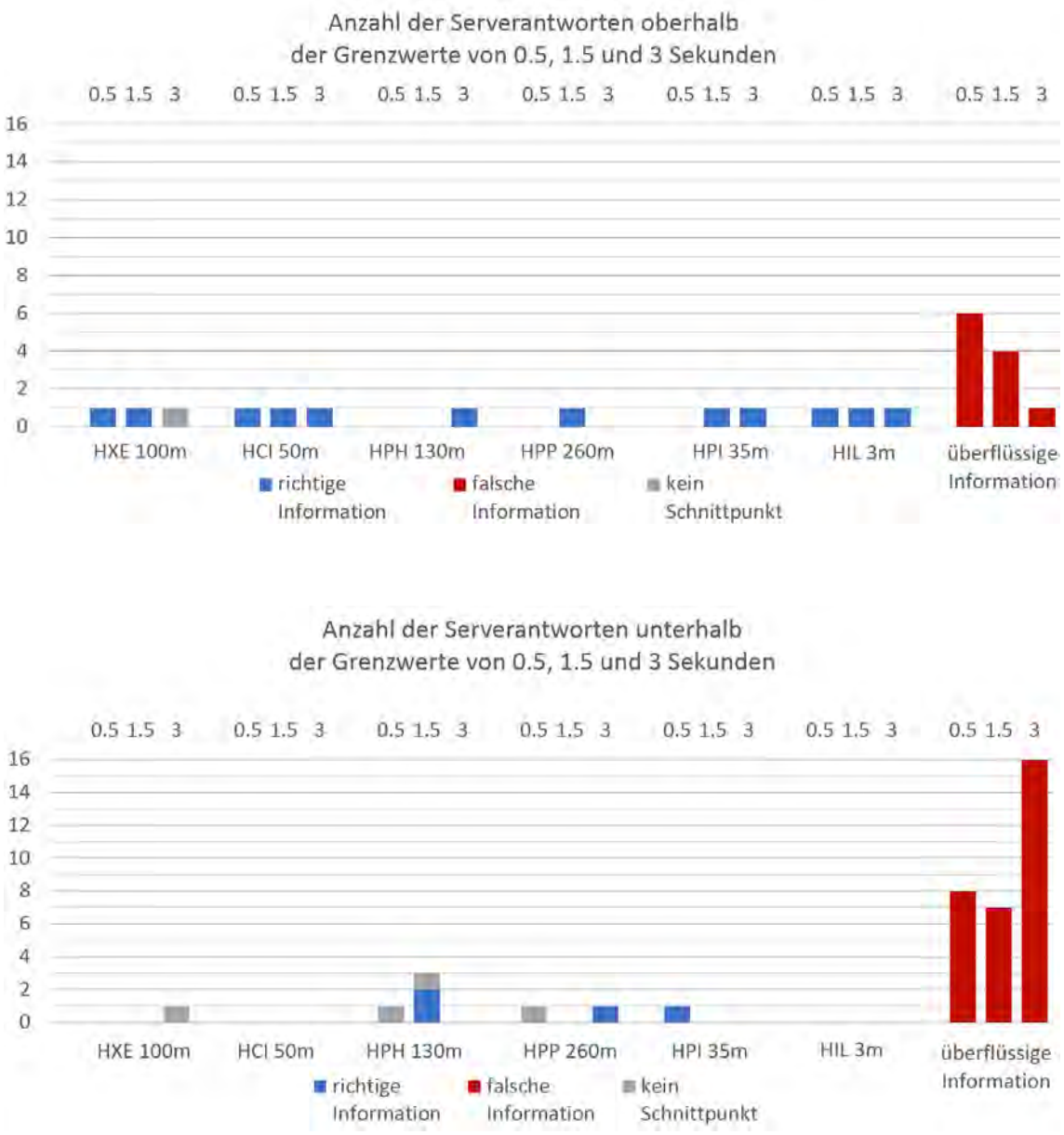

Abbildung 8.13:
In der Graphik ist die Anzahl Serverantworten ober- und unterhalb der Grenzwerte der minimalen Betrachtungsdauer dargestellt. Die Auswertung erfolgte für jede Phase, in der ein Gebäude fixiert wurde, sowie für den entsprechenden Grenzwert. Neben dem Namen des betrachteten Gebäudes ist zusätzliche die horizontale Distanz von der Testperson zum betrachteten Gebäude angegeben.

Kapitel 9

Schlussfolgerungen

9.1 Fazit aus technischer Sicht

Es darf festgehalten werden, dass die aufgrund der Zielsetzung der Arbeit bestehenden Erwartungen erfüllt werden konnten. Auf die technischen Herausforderungen sowie auf die benutzerspezifischen Fragen konnten die entsprechenden Lösungsvorschläge und Antworten erarbeitet werden. Diese werden jedoch erst im Kapitel 9.2 behandelt. In diesem Unterkapitel wird kritisch auf die Qualität der verwendeten Sensoren eingegangen, sowie auf die eingesetzten Applikationen und die daraus resultierenden Daten.

Die eingesetzte Smartphone App zur Ausgabe der GPS–Position, des Azimut's, des Höhen– und Rollwinkels wurde durch den Verfasser nicht näher betrachtet und die bereits existierenden Anwendung übernommen. Zur Prüfung der App bestand aufgrund in der Vergangenheit erzielter Ergebnisse kein Anlass. Erst während des Versuchs in Kapitel 8.4 wurde erkannt, dass sich im Verlaufe von 80 Sekunden die erhaltenen GPS–Positionen bis auf die 15te Nachkommastelle nicht änderten. Deshalb ist davon auszugehen, dass die Applikation nicht laufend Sensoren–Daten ausgibt, sondern diese nach gewissen Kriterien generalisiert. Bevor diese Smartphone App für weitere Testreihen eingesetzt wird, ist deren Funktionsweise genauer zu analysieren und bei Bedarf anzupassen.

Die Version der eingesetzten Eye Tracker Software *D–Lab–Control* enthielt keine Schnittstelle zur Ausgabe eines synchronisierten Datenstroms von Blickdaten und Smartphone–Daten an eine eigenständig entwickelte Anwendung. Folglich konnten die Daten erst nachträglich verwendet werden, was genutzt wurde um die Aufnahmedaten in einem Postprozess nachzukalibrieren. Ab 2014 soll eine überarbeitete Software–Version angeboten werden, welche über eine Schnittstelle zur Ausgabe eines synchronisierten Datenstroms verfügen wird.[1] Ab diesem Zeitpunkt ist das Vorgehen leicht zu modifizieren, indem die Blicke be-

[1] Diese Anmerkung beruht auf einer Aussage des Herstellers des Eye Tracker's Ergoneers GmbH vom Herbst 2013

reits während der Aufnahme nachkalibriert werden. Diese Funktion bietet Ergoneers bereits heute an und ist unter der Bezeichnung „online Kalibrierung“ bekannt Ergoneers (2011, S. 58).

Abschliessend kann gesagt werden, dass zu einer erfolgreichen blickbasierten Mensch–Computer–Interaktion noch weitere technische Verbesserungen an den Sensoren erfolgen müssen. Für den Einsatz des mobilen Eye Trackers unter realen Bedingungen wäre es von Vorteil, wenn die Video–Auflösung verbessert werden könnte, sowie die Kameraeinstellungen für wechselnde Lichtverhältnisse optimiert würden. Unter Umständen könnte dadurch sogar die Pupillenerkennung unter freiem Himmel verbessert werden. Diese Anpassungen würden mit Sicherheit die Kalibrierung und damit die Genauigkeit des Gesamtsystems erhöhen.

9.2 Beantwortung der Forschungsfragen

Wie kann der existierende Algorithmus als Web Service erstellt werden?
Diese Frage muss etwas differenziert beantwortet werden. Es zeigte sich, dass der bestehende Algorithmus in einzelne Aufnahmeprozesse unterteilt werden muss, damit eine zeitliche Entkoppelung zwischen dem Kalibrations– und dem Aufnahmeprozess sichergestellt werden kann. Dadurch wurde erreicht, dass sich die Verzögerungen – die während dem Kalibrationsprozess anfallen – nicht auf den Aufnahmeprozess auswirken.

Im Weiteren ermöglichte erst diese Unterteilung eine serviceorientierte Systemarchitektur mit einer klar definierten Arbeitsteilung zwischen Client und Server. Dadurch konnten die eigentlichen GIS–Funktionen inklusive der dazu benötigten Geodaten – die 3D–Verschneidung eines Blickes mit dem 3D–Gebäudemodell sowie das digitale Terrainmodell – auf den Server ausgelagert werden. Ausserdem musste sich der Client dadurch nur noch um die Blickberechnung aus den Sensordaten ins nationale Koordinatensystems, sowie die Koordination zwischen Service–anfragen und Benutzer-Gesamtsystem-Interaktionen kümmern.

Bereits Mosimann (2013) erkannte, dass die Höhenangaben des GPS–Sensors sehr stark von den tatsächlichen Höhenwerten abweichen und ersetzte die vom Smartphone gemessenen Werte, durch den entsprechenden Höhenwert aus dem lokal gespeicherten digitalen Höhenmodell (DTM) der Schweiz. Dieses Prinzip wurde übernommen indem der Client für jede gemessene GPS–Position den korrekten Höhenwert aus dem DTM abfragte. Dies erfolgte allerdings nicht mehr lokal sondern in Form einer *„getFeature–Abfrage“* des entsprechend veröffentlichten Web–Map–Service.

Zudem musste die Anwendung von Mosimann (2013) so modifiziert werden, dass nicht mehr sämtliche Datensätze auf einmal verarbeitet werden, sondern alle Berechnungsschritte mussten fortlaufend für jeden einzelnen Blick erfolgen. Das heisst, dass die Blicke seriell abgearbeitet werden müssen und so die Verschneidungsergebnisse laufend im Kartenfenster des GUI's ausgegeben werden können.

In dieser Arbeit konnte gezeigt werden, dass die serverseitige Programmierung eines 3D-Verschneidungsdienstes signifikant performanter ist, als ein Geoprocessing–Dienst, der mittels ArcGIS–ModelBuilder erstellt wurde. Die serverseitige Programmierung erfolgte in From einer Server–Objekt–Erweiterung, welche immer an einen Web–Map–Kartendienst gebunden ist.

Wie ist mit den Echtzeit Anforderungen umzugehen?
Es stellte sich heraus, dass nicht jeder aufgezeichnete Blick an den Verschneidungsdienst gesendet werden darf, da sonst – aufgrund der Berechnungszeiten – eine beträchtliche Kumulation der Verzögerungen auftritt. Der menschliche Blickverlauf lässt sich in Phasen von schnellen Augenbewegungen (Sakkaden) und Fixationen unterteilen. Während einer Fixation ruht der Blick für mindestens 100 Millisekunden auf einem Objekt, nur in diesen Phasen ist der Mensch in der Lage Informationen aufzunehmen. Aus den im Verlaufe einer Fixation aufgezeichneten Blicken, lässt sich ein für diesen Moment repräsentativer Blickvektor berechnen. Folglich war es naheliegend, nur diese resultierenden Blickvektoren an den Verschneidungsdienst zu senden. Dazu war die Verwendung eines verteilungsbasierten Identifikationsalgorithmus zielführend.

Es zeigte sich, dass eine Aufsummierung der Verzögerungen nur dann auftritt, wenn die Dauer zwischen zwei Fixationen kürzer ist als die Berechnungszeit des entsprechenden Web–Services. Daher kann gesagt werden, dass die minimal mögliche Aufnahmefrequenz von der Berechnungszeit einer Verschneidung abhängt. Im vorliegenden Fall entsprach dies einer Dauer von 150 Millisekunden oder anders formuliert, einer minimalen Aufnahmefrequenz von 6.6 Hertz.

Mittels einem iterativen Vorgehen konnten die optimalen Auswahlkriterien zu Erkennung der Fixationen festgelegt werden. Unter realen Bedingungen und einem Verteilungsradius von 16 Pixeln sowie einer minimalen Fixationszeit von 150 Millisekunden als Auswahlkriterien, trat keine Aufsummierung der Verzögerungen mehr auf.

Erhält der Nutzer die Angaben, die er wünscht?
In der Arbeit konnte ausgewiesen werden, dass 75% aller Blicke innerhalb eines kegelförmigen Zielbereiches mit einem Öffnungswinkel von 2.5° liegen und das korrekte Ver-

schneidungsergebnis liefern. Der so definierte Zielbereich entspricht in etwa dem zentralen scharfen Sehbereich des menschlichen Auges. Die mittlere horizontale Abweichung vom Ziel betrug 2.08° ±0.46° und die mittlere vertikale Abweichung 0.14° ±0.68°. Falls die GPS–Positionen durch die Koordinaten des tatsächlichen Standortes ersetzt werden, würden allerdings 100% der Blicke den Zielbereich treffen.

Es konnte gezeigt werden, dass ein einfacher Map–Matching–Algorithmus, welcher die GPS–Position auf die Strassenachse versetzt, das Verschneidungsergebnis nur in einem sehr beschränkten Bereich positiv beeinflussen kann. Und dies auch nur, falls die Blickrichtung längs der Strassenachse verläuft und der Personenstandort sich auf der Achse befindet. Sind diese Bedingungen nicht erfüllt, so wird das Ergebnis nicht verbessert sondern gar verschlechtert.

Wenn der Benutzer nur informiert wird, falls das Verschneidungsergebnis während einer kumulierten Fixationsdauer von mindestens drei Sekunden identisch bleibt, erhöht sich die Wahrscheinlichkeit, dass nur angeforderte Informationen ausgegeben werden. In dieser Arbeit konnte gezeigt werden, dass der Benutzer dadurch mindestens 83% für ihn relevante Informationen erhält.

9.3 Ausblick

Damit die Effektivität einer blickbasierten Interaktion erhöht werden kann, sind weitere Anstrengungen zu unternehmen. Dafür muss die Genauigkeit der GPS-Position verbessert werden. Dies könnte entweder durch eine Korrektur der gemessenen Position mit Echtzeitdaten wie swipos–Nav[2] erreicht werden, oder aber durch die Verwendung von komplexeren Map-Matching Algorithmen. Diese berechnen die wahrscheinlichste Position der Person aufgrund der bekannten Strassenbreite, der Fortbewegungsgeschwindigkeit und der vorangegangen Messwerte.

Wie in dieser Arbeit aufgezeigt wurde, ist eine blickbasierte Mensch–Computer–Interaktion unter den aufgezeigten Rahmenbedingungen möglich. Eine zentrale Erkenntnis ist, dass die Berechnungszeit die minimal mögliche Verschneidungsfrequenz bestimmt, ohne dass eine Kumulation der Verzögerungen auftritt. In weiteren Studien – mit mehreren Probanden – ist zu zeigen, welcher Fixationsalgorithmus am besten geeignet ist, um sich dieser minimalen Verschneidungsfrequenz anzunähern. Dabei liegt der Schwerpunkt auf der optimalen Festlegung der Kriterien zur Fixationserkennung. Dazu sind die Algorithmen so in die Anwendung zu integrieren, dass die Fixationen kontinuierlich und in Echtzeit erfolgen können.

[2] http://www.swisstopo.admin.ch/internet/swisstopo/de/home/products/services/swipos/nav.html

In dieser Arbeit wurde zur Reduzierung der überschüssigen Serverantworten eine minimale Betrachtungsdauer von mindestens drei Sekunden ermittelt. Ein weiterer Ansatz wäre, in den Abfolgen von Fixationen und Sakkaden, Suchmuster in Echtzeit zu erkennen. So würde die Anwendung erst eine Anfrage an den Server senden, wenn aufgrund der Blickmuster eine Auskunft erwünscht ist. Nach einer erfolgreichen Integration der Mustererkennung könnte die dadurch erhöhte Benutzerfreundlichkeit anhand einer repräsentativen Feldstudie genauer beziffert werden.

Teil V

Appendix

Anhang A

Dateistrukturen

A.1 Erklärung Journal Datei

Spalte	Name	Wertebereich	Bedeutung
1	index	Aufsteigend Nummeriert	Bildnummer (25 Bilder pro Sekunde), startet bei 0
2	timestamp	Zahl	Aufsteigend in Millisekunden, beginnt wenn Recording Computer gestartet wird
3	eyeroihoriz	0 – 500	Kalibrierungseinstellung
4	eyeroivert	0 – 500	Kalibrierungseinstellung
5	eyeroizoomx	0 – 500	Kalibrierungseinstellung
6	eyeroizoomy	0 – 500	Kalibrierungseinstellung
7	blendmode	0 oder 1	Blending Modus 0: Plus 1: Over
8	blendfactor	0 – 255	Transparenz für Blending Modus
9	onlinecalib	0 oder 1	0: Nicht online Kalibriert 1: Online kalibriert
10	eye_valid	0 oder 1	0: Pupille wurde nicht detektiert 1: Pupille wurde detektiert
11	eye_x	0 – 768	Horizontale Pupillen Koordinate im Video der Augenkamera, sobald durch zwei dividiert
12	eye_y	0 – 576	Vertikale Pupillen Koordinate im Video der Augenkamera, sobald durch zwei dividiert
13	eye_w	Zahl	Breite der Pupille in Pixel
14	eye_h	Zahl	Höhe der Pupille in Pixel
15	eye_a	Zahl	Fläche der Pupille in Pixel
16	field_x	0 – 768	Horizontale Pupillen Koordinate im Video der Feldkamera
17	field_y	0 – 576	Vertikale Pupillen Koordinate im Video der Feldkamera
18	displaymode	0, 1 oder 2	0: Feldkamera aktiv 1: Augenkamera aktiv 2:Blending Modus
19	displaydetection	0 oder 1	0: Pupillendetektion nicht aktiv 1: Pupillendetektion aktiv
20	event	String	Kennzeichnet Events, welche mit den Daten über die Netzwerkschnittstelle aufgenommen werden können und die Task-Intervall von *D-Lab Control*

A.2 Journal Datei

Version	1.08																			
index	timestamp	eyeroihoriz	eyeroivert	eyeroizoomx	eyeroizoomy	blendmode	blendfactor	onlinecalib	eye_valid	eye_x	eye_y	eye_w	eye_h	eye_a	field_x	field_y	displaymode	displayeyedetection	event	
1453	1498312	182	128	223	167	1	128	0	1	328	324	31	26	610	322	281	2	1		
1454	1498352	182	128	223	167	1	128	0	1	328	324	31	27	613	322	281	2	1		
1455	1498393	182	128	223	167	1	128	0	1	328	324	31	27	608	322	281	2	1		
1456	1498432	182	128	223	167	1	128	0	1	328	324	31	27	610	322	281	2	1		
1457	1498471	182	128	223	167	1	128	0	1	328	324	31	27	610	322	281	2	1		
1458	1498510	182	128	223	167	1	128	0	1	328	324	30	27	605	322	281	2	1		
1459	1498550	182	128	223	167	1	128	0	1	328	324	30	27	592	322	281	2	1		
1460	1498590	182	128	223	167	1	128	0	1	328	326	30	27	588	322	281	2	1		
1461	1498631	182	128	223	167	1	128	0	1	328	326	30	27	586	322	277	2	1		
1462	1498671	182	128	223	167	1	128	0	1	328	326	30	26	585	322	277	2	1		
1463	1498710	182	128	223	167	1	128	0	1	330	326	30	26	589	325	277	2	1		
1464	1498749	182	128	223	167	1	128	0	1	330	326	30	27	597	325	277	2	1		
1465	1498788	182	128	223	167	1	128	0	1	330	326	30	27	604	325	277	2	1		
1466	1498828	182	128	223	167	1	128	0	1	330	326	30	27	604	325	277	2	1		
1467	1498858	182	128	223	167	1	128	0	1	328	326	30	26	604	322	277	2	1		
1468	1498896	182	128	223	167	1	128	0	1	330	326	30	27	609	325	277	2	1		
1469	1498936	182	128	223	167	1	128	0	1	330	326	30	27	618	325	277	2	1		
1470	1498975	182	128	223	167	1	128	0	1	328	326	30	27	606	322	277	2	1		
1471	1499014	182	128	223	167	1	128	0	1	330	326	30	28	615	325	277	2	1		
1472	1499053	182	128	223	167	1	128	0	1	328	326	30	28	631	322	277	2	1		
1473	1499093	182	128	223	167	1	128	0	1	328	326	31	28	637	322	277	2	1		
1474	1499132	182	128	223	167	1	128	0	1	330	326	31	28	641	325	277	2	1		
1475	1499173	182	128	223	167	1	128	0	1	328	326	32	28	654	322	277	2	1		
1476	1499211	182	128	223	167	1	128	0	1	328	326	32	28	655	322	277	2	1		
1477	1499250	182	128	223	167	1	128	0	1	328	326	32	28	652	322	277	2	1		
1478	1499288	182	128	223	167	1	128	0	1	330	326	31	28	642	325	277	2	1		
1479	1499325	182	128	223	167	1	128	0	1	328	326	31	28	647	322	277	2	1		
1480	1499366	182	128	223	167	1	128	0	1	328	326	32	28	647	322	277	2	1		
1481	1499405	182	128	223	167	1	128	0	1	328	326	32	28	652	322	277	2	1		
1482	1499442	182	128	223	167	1	128	0	1	328	326	32	28	656	322	277	2	1		
1483	1499471	182	128	223	167	1	128	0	1	328	326	32	28	662	322	277	2	1		
1484	1499510	182	128	223	167	1	128	0	1	328	326	32	28	659	322	277	2	1		
1485	1499549	182	128	223	167	1	128	0	1	328	326	32	28	657	322	277	2	1		
1486	1499588	182	128	223	167	1	128	0	1	328	326	31	28	658	322	277	2	1		
1487	1499626	182	128	223	167	1	128	0	1	328	326	31	28	659	322	277	2	1		
1488	1499664	182	128	223	167	1	128	0	1	328	326	31	28	654	322	277	2	1		
1489	1499703	182	128	223	167	1	128	0	1	328	326	31	28	655	322	277	2	1		
1490	1499741	182	128	223	167	1	128	0	1	328	326	31	28	663	322	277	2	1		
1491	1499780	182	128	223	167	1	128	0	1	328	326	31	28	676	322	277	2	1		
1492	1499819	182	128	223	167	1	128	0	1	328	326	32	28	671	322	277	2	1		
1493	1499858	182	128	223	167	1	128	0	1	328	326	31	28	661	322	277	2	1		
1494	1499896	182	128	223	167	1	128	0	1	328	326	32	28	665	322	277	2	1		
1495	1499935	182	128	223	167	1	128	0	1	328	326	31	28	663	322	277	2	1		
1496	1499974	182	128	223	167	1	128	0	1	328	326	31	28	655	322	277	2	1		
1497	1500013	182	128	223	167	1	128	0	1	328	326	31	28	657	322	277	2	1		
1498	1500051	182	128	223	167	1	128	0	1	328	326	31	28	659	322	277	2	1		
1499	1500079	182	128	223	167	1	128	0	1	328	326	31	28	662	322	277	2	1		
1500	1500118	182	128	223	167	1	128	0	1	328	326	32	29	683	322	277	2	1		
1501	1500157	182	128	223	167	1	128	0	1	328	326	31	28	661	322	277	2	1		

A.3 Dikablis Journal Datei

634909872903O3	2	J2	983291	182	146	196	112	1	128	0	1	342	282	21	18	274	340	293	2	1
63490987290343	3	J3	983328	182	146	196	112	1	128	0	1	342	282	21	18	274	340	293	2	1
63490987290383	4	J4	983367	182	146	196	112	1	128	0	1	340	282	20	18	273	336	293	2	1
63490987290423	5	J5	983404	182	146	196	112	1	128	0	1	342	282	20	18	273	340	293	2	1
63490987290463	6	J6	983443	182	146	196	112	1	128	0	1	340	282	20	18	265	336	293	2	1
63490987290503	7	J7	983472	182	146	196	112	1	128	0	1	342	282	21	18	275	340	293	2	1
63490987290543	8	J8	983511	182	146	196	112	1	128	0	1	340	282	21	18	277	336	293	2	1
63490987290583	9	J9	983549	182	146	196	112	1	128	0	1	342	282	21	18	284	340	293	2	1
		J10	983588	182	146	196	112	1	128	0	1	340	28							
		J12	983666	182	146	196	112	1	128	0	1	340	28							
		J13	983705	182	146	196	112	1	128	0	1	340	28							
		J14	983743	182	146	196	112	1	128	0	1	340	28							
		J15	983781	182	146	196	112	1	128	0	1	340	28							
		J16	983820	182	146	196	112	1	128	0	1	340	28							
		J17	983860	182	146	196	112	1	128	0	1	340	280							
		J18	983898	182	146	196	112	1	128	0	1	340	280	21	17	273	336	298	2	1
		J19	983937	182	146	196	112	1	128	0	1	340	282	21	18	281	336	293	2	1
		J20	983975	182	146	196	112	1	128	0	1	340	282	21	18	287	336	293	2	1
		J21	984013	182	146	196	112	1	128	0	1	340	282	21	18	285	336	293	2	1
		J22	984051	182	146	196	112	1	128	0	1	340	282	21	18	282	336	293	2	1
		J23	984080	182	146	196	112	1	128	0	1	340	282	21	18	275	336	293	2	1
		J24	984119	182	146	196	112	1	128	0	1	340	282	21	18	285	336	293	2	1
		J25	984157	182	146	196	112	1	128	0	1	340	282	21	18	282	336	293	2	1
63490987291223	25	J26	984196	182	146	196	112	1	128	0	1	340	282	21	18	282	336	293	2	1
63490987291263	26	J27	984235	182	146	196	112	1	128	0	1	340	282	21	18	279	336	293	2	1
63490987291303	27	J28	984274	182	146	196	112	1	128	0	1	340	280	20	18	278	336	298	2	1
63490987291343	28	J29	984313	182	146	196	112	1	128	0	1	340	280	21	18	287	336	298	2	1
63490987291383	29	J30	984351	182	146	196	112	1	128	0	1	338	280	21	18	284	333	298	2	1

Durch D-Lab Control zusätzlich erzeugte Attribute:
1. Spalte entspricht Systemzeit

2. Spalte ist ein neuer Index

3. Spalte entspricht dem ursprünglichen Index der Journal-Datei, nur ist der Buchstabe J davor gestellt

Entspricht den Spalten der Journal-Datei, welche durch Dikablis erstellt wurde

A.4 Blicklinie als JSON Objekt


```
{
 "displayFieldName": "",
 "hasZ": true,
 "hasM": true,
 "geometryType": "esriGeometryPolyline",
 "spatialReference": {
  "wkid": 21781,
  "latestWkid": 21781
 },
 "fields": [
  {
   "name": "FID",
   "type": "esriFieldTypeOID",
   "alias": "FID"
  },
  {
   "name": "timestamp",
   "type": "esriFieldTypeInteger",
   "alias": "timestamp"
  },
  {
   "name": "Shape_Length",
   "type": "esriFieldTypeDouble",
   "alias": "Shape_Length"
  }
 ],
 "features" : [
      {
        "attributes" : {
            "FID" : 1,
            "timestamp" : 985879,
            "Shape_Length" : 600
        },
         "geometry" : {
             "hasZ": true,
             "hasM": false,
             "geometryType": "esriGeometryPolyline",
             "spatialReference": { "wkid": 21781},
             "paths" : [
                    [
                     [
                       680627.788,
                       251317.811,
                       524.140
                     ],
                     [
                        680179.592,
                        251716.709,
                        627.405
                     ]
                     ]
            ]
          }
        }
],
 "exceededTransferLimit": false
}
```

Literaturverzeichnis

Alonso, G., Casati, F., Kuno, H. & Machiraju, V. (2010): *Web Services: Concepts, Architectures and Applications.* Springer Berlin Heidelberg.

Baldauf, M., Fröhlich, P. & Hutter, S. (2010): *KIBITZER: A Wearable System for Eye-gaze-based Mobile Urban Exploration.* In: *Proceedings of the 1st Augmented Human International Conference*, 9:1–9:5. ACM, New York, NY, USA.

Büttner, O. B. (2009): *Kognitive Prozesse am Point of Sale: Zur Qualität von Datenerhebungsmethoden der Konsumentenforschung.* Springer Berlin Heidelberg.

Carmigniani, J., Furht, B., Anisetti, M., Ceravolo, P., Damiani, E. & Ivkovic, M. (2011): *Augmented reality technologies, systems and applications.* In: Multimedia Tools and Applications, **51**, 1: 341–377.

Dahm, M. (2006): *Grundlagen der Mensch–Computer–Interaktion.* Pearson Studium.

Dimov, S., Vyas, V., Schardt, M., Faruque, A., Narla, S. & Fu, H. (2005): *Customizing Model-view-controller Design Pattern for Web Controlled Household Appliances and Devices.* In: *Proceedings of the 43rd Annual Southeast Regional Conference - Volume 2*, 343–344. ACM, New York, NY, USA.

Elrad, T., Filman, R. E. & Bader, A. (2001): *Aspect-oriented Programming: Introduction.* In: Commun. ACM, **44**, 10: 29–32.

Ergoneers (2011): *Dikablis - Das Blickerfassungssystem - Benutzerhandbuch.* Ergoneers GmbH, 85077 Manching. Dikablis Versionen 2.0; Erhältlich `http://www.ergoneers.com` besucht am 30. Mai 2012.

ESRI (2013a): *Introduction to server object extensions.* URL `http://resources.arcgis.com/en/help/arcobjects-java/`, letzter Zugriff am 02.12.2013.

ESRI (2013b): *Was ist eine Serverobjekterweiterung?* URL `http://resources.arcgis.com/de/help/main/10.1/index.html#/na/0154000004s5000000/`, letzter Zugriff am 02.12.2013.

Fielding, R. T. (2000): *Architectural styles and the design of network-based software architectures.* Dissertation, University of California.

Geise, S. (2011): *Eyetracking in der Kommunikations- und Medienwissenschaft: Theorie, Methode und kritische Reflexion.* In: SCM Studies in Communication | Media, **Heft 2**.

giswiki (2007): *Open Geospatial Consortium.* URL `http://giswiki.org/wiki/Open_Geospatial_Consortium`, letzter Zugriff am 12.01.2014.

Goldberg, J. H. & Helfman, J. I. (2010): *Visual scanpath representation.* In: *ETRA '10: Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications.* ACM, New York, NY, USA.

Grob, D. (2012): *Konzeption und prototypische Umsetzung eines Frameworks zur ortsbezogenen Erhebung und Visualisierung mobiler Eye-Tracking Daten.* Masterprojektarbeit, ETH Zürich, Schweiz.

Hangel, S., Krieger, A. & Praxl, U. (2012): *Geovisual Analytics im Controlling.* In: *Angewandte Geoinformatik.* Herbert Wichmann Verlag, Berlin/Offenbach, Deutschland.

Höck, M. & Manegold, J. (2010): *ArcMap: Programmierung mit VB.NET.* im Eigenverlag. URL `http://www.arcobjectsbuch.de`.

Hofer, N. & Mayerhofer, W. (2010): *Die Blickregistrierung in der Werbewirkungsforschung: Grundlagen und Ergebnisse.* In: der markt, **49**, 3-4: 143–169.

Huang, J., Xu, Q., Tiwana, B., Mao, Z. M., Zhang, M. & Bahl, P. (2010): *Anatomizing application performance differences on smartphones.* In: *Proceedings of the 8th international conference on Mobile systems, applications, and services*, MobiSys '10, 165–178. ACM, New York, NY, USA.

Jacob, R. J. K. (1990): *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, 11–18. ACM, New York, NY, USA.

Josua, S. & Stefan, K. (2013): *Präzise Positionsbestimmung mit Low-Cost-GPS und Postprocessing.* Technical Report Nr. 1301, HSR Hochschule für Technik Rapperswil, Rapperswil, Schweiz.

Juhlin, O. & Önnevall, E. (2013): *On the Relation of Ordinary Gestures to TV Screens: General Lessons for the Design of Collaborative Interactive Techniques.* In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, 919–930. ACM, New York, NY, USA.

Junglas, I. A. & Watson, R. T. (2008): *Location-based Services.* In: Commun. ACM, **51**, 3: 65–69.

Karslake, J. S. (1940): *The Purdue eye-camera: a practical apparatus for studying the attention value of advertisements.* In: Journal of Applied Psychology, **24**, 4: 417.

Kiefer, P., Giannopoulos, I. & Raubal, M. (2013): *Using Eye Movements to Recognize Activities on Cartographic Maps.* In: *Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL'13, 478–481. ACM, New York, NY, USA.

Kiefer, P., Straub, F. & Raubal, M. (2012): *Towards Location-aware Mobile Eye Tracking.* In: *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, 313–316. ACM, New York, NY, USA.

Krafzig, D., Banke, K. & Slama, D. (2005): *Enterprise SOA: service-oriented architecture best practices.* Pearson Education.

Krasner, G. E. & Pope, S. T. (1988): *A description of the model view controller paradigm in the Smalltalk-80 system.* In: Journal of Object-Oriented Programming, **1**, 3: 26–49.

Laffont, I., Biard, N., Bouteille, J., Pouplin, S., Guillon, B., Bernuz, B. & Rech, C. (2008): *Tétraplégie: solutions technologiques de compensation des incapacités découlant de l'atteinte des membres supérieurs.* In: La Lettre de médecine physique et de réadaptation, **24**, 3: 113–121.

Lukander, K., Jagadeesan, S., Chi, H. & Müller, K. (2013): *OMG!: A New Robust, Wearable and Affordable Open Source Mobile Gaze Tracker.* In: *Proceedings of the 15th International Conference on Human-computer Interaction with Mobile Devices and Services*, MobileHCI '13, 408–411. ACM, New York, NY, USA.

Maglogiannis, I. & Hadjiefthymiades, S. (2007): *EmerLoc: Location-based services for emergency medical incidents.* In: International journal of medical informatics, **76**, 10: 747–759.

Majaranta, P. & Räihä, K.-J. (2002): *Twenty Years of Eye Typing: Systems and Design Issues.* In: *Proceedings of the 2002 Symposium on Eye Tracking Research & Applications*, ETRA '02, 15–22. ACM, New York, NY, USA.

Masak, D. (2007): *Service Oriented System Engineering.* In: *SOA?*, Xpert.press, 191–230. Springer Berlin Heidelberg.

Melzer, I. (2010): *Service-orientierte Architekturen mit Web Services: Konzepte-Standards-Praxis.* Springer Berlin Heidelberg.

Menasce, D. A. & Almeida, V. A. (2002): *Capacity Planning for Web Services: metrics, models, and methods.* Prentice Hall Upper Saddle River.

Miniotas, D., Spakov, O. & Evreinov, G. (2003): *Symbol Creator: An alternative eye-based text entry technique with low demand for screen space.* In: *Proceedings of INTERACT*, Bd. 3, 137–143.

Miniotas, D., Špakov, O., Tugoy, I. & MacKenzie, I. S. (2006): *Speech-augmented Eye Gaze Interaction with Small Closely Spaced Targets.* In: *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, ETRA '06, 67–72. ACM, New York, NY, USA.

Moniri, M. M. & Müller, C. (2012): *Multimodal Reference Resolution for Mobile Spatial Interaction in Urban Environments.* In: *Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, AutomotiveUI '12, 241–248. ACM, New York, NY, USA.

Mosimann, P. (2013): *Abbildung im geographischen Raum erfasster Eye-Tracking Daten und Geosensor-Informationen in einer 3D Umgebung zur Berechnung der Koordinaten der Blickpunkte.* Masterarbeit, ETH Zürich, Schweiz.

OASIS (2004): *UDDI Version 3 Specification.* URL `https://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm`, letzter Zugriff am 10.01.2014.

OGC (2007): *OGC Standards and Supporting Documents.* URL `http://www.opengeospatial.org/standards/`, letzter Zugriff am 12.01.2014.

Paech, B., Dutoit, A. H., Kerkow, D. & Von Knethen, A. (2002): *Functional requirements, non-functional requirements, and architecture should not be separated.* Position paper, Fraunhofer-Institut für Experimentelles Software Engineering (IESE) und Technische Universität München, Kaiserslautern, Deutschland.

Park, H. M., Lee, S. H. & Choi, J. S. (2008): *Wearable Augmented Reality System Using Gaze Interaction.* In: *Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality*, ISMAR '08, 175–176. IEEE Computer Society, Washington, DC, USA.

Pfeiffer, T. & Wachsmuth, I. (2011): *Dreidimensionale Erfassung visueller Aufmerksamkeit für Usability-Bewertungen an virtuellen Prototypen.* In: Gausemeier, J., Grafe, M. & auf der Heide, F. M. [Hrsg.]: *10. Paderborner Workshop: Augmented and Virtual Reality in der Produktentstehung*, 295, 39–51. Heinz Nixdorf Institut, Universität Paderborn.

Rao, B. & Minakakis, L. (2003): *Evolution of Mobile Location-based Services.* In: Commun. ACM, **46**, 12: 61–65.

Reenskaug, T. M. H. (1979): *The original MVC reports.* Techn. Ber., Research report University of Oslo.

Richardson, L. & Ruby, S. (2007): *RESTful web services*. O'Reilly Media.

Salvucci, D. D. & Goldberg, J. H. (2000): *Identifying Fixations and Saccades in Eye-tracking Protocols*. In: *Proceedings of the 2000 Symposium on Eye Tracking Research & Applications*, ETRA '00, 71–78. ACM, New York, NY, USA.

Schlütz, D. & Möhring, W. (2013): *Handbuch standardisierte Erhebungsverfahren in der Kommunikationswissenschaft*. Springer Berlin Heidelberg.

Schönherr, M. (2007): *Enterprise application integration: Flexibilisierung komplexer Unternehmensarchitekturen*, Bd. 1. GITO GmbH Verlag, Berlin.

Swisstopo (2008): *Formeln und Konstanten für die Berechnung der Schweizerischen schiefachsigen Zylinderprojektion und der Transformation zwischen Koordinatensystemen*. URL `http://www.swisstopo.admin.ch/`, letzter Zugriff am 15.12.2013.

Tanriverdi, V. & Jacob, R. J. K. (2000): *Interacting with Eye Movements in Virtual Environments*. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, 265–272. ACM, New York, NY, USA.

Veit, M. & Herrmann, S. (2003): *Model-view-controller and object teams: A perfect match of paradigms*. In: *Proceedings of the 2nd international conference on Aspect-oriented software development*, 140–149. ACM.

Vertegaal, R. (2008): *A Fitts Law Comparison of Eye Tracking and Manual Input in the Selection of Visual Targets*. In: *Proceedings of the 10th International Conference on Multimodal Interfaces*, ICMI '08, 241–248. ACM, New York, NY, USA.

W3C (2001): *Web Services Description Language (WSDL) 1.1*. URL `http://www.w3.org/TR/wsdl`, letzter Zugriff am 10.01.2014.

W3C (2002): *Web Services Description Requirements*. URL `http://www.w3.org/TR/ws-desc-reqs/`, letzter Zugriff am 10.01.2014.

W3C (2004): *Web Services Architecture*. URL `http://www.w3.org/TR/ws-arch/`, letzter Zugriff am 10.01.2014.

Wandmacher, J. (1993): *Software-Ergonomie*, Bd. 2. Walter de Gruyter, Berlin.

Paris Lodron Universität Salzburg
Interfakultärer Fachbereich
Geoinformatik - UNIGIS
Ao. Univ. Prof. Dr. Josef Strobl

Titel der Arbeit:

Erstellung und Evaluation einer service-orientierten Systemarchitektur zur ortsbezogenen blickbasierten Interaktion mit 3D-Geoobjekten

Art der Arbeit und Datum:

Masterarbeit, 28. Februar 2014

Betreuung:

Ioannis Giannopoulos ETH Zürich
Dr. Peter Kiefer ETH Zürich

Student:

Name: Simon Haesler
E-mail: simon.haesler@live.unigis.net
Legi-Nr.: u1552

Erklärung betreffend Plagiaten:

Ich erkläre mit meiner Unterschrift, das Merkblatt Eigenständigkeitserklärung der ETH Zürich zur Kenntnis genommen, die vorliegende Arbeit selbständig verfasst und die im betroffenen Fachgebiet üblichen Zitiervorschriften eingehalten zu haben.

Luzern, 28. 2. 2014: _____