Hochschule für angewandte Wissenschaft und Kunst

Fachhochschule
Hildesheim / Holzminden / Göttingen
Fakultät Naturwissenschaften und Technik

Masterarbeit

Entwicklung einer Klassenbibliothek zur Integration von Blickrichtungsmesssystemen in eine Applikations- und Entwicklungsplattform zur Online-Auswertung und Analyse im Automotivebereich

von

Dipl.-Ing.(FH) Gerald Temme

Masterstudiengang Elektro-/Informationstechnik

Durchgeführt beim

Deutsches Zentrum für Luft- und Raumfahrt e.V.
in der Helmholtz-Gemeinschaft

am Standort Braunschweig
im Institut für Verkehrsführung und Fahrzeugsteuerung (FS)

Erstprüfer : Prof. Dr. -Ing. Bernd Stock (HAWK)
Zweitprüfer : Dr. -Ing. Frank Flemisch (DLR)

vorgelegt am : 24. August 2007

Eidesstattliche Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen verwendet zu haben. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt.

Braunschweig, den 24. August 2007

Gerald Temme

Einverständniserklärung

Ich bin damit einverstanden, dass von meiner Masterarbeit gem. §§ 16 und 17 UrhG Vervielfältigungsstücke erstellt werden können, um sie an Dritte weiterzugeben. Mein Einverständnis erstreckt sich auch darauf, dass die Masterarbeit zu diesem Zweck an Dritte weitergegeben werden kann.

[ ] einverstanden
[ ] nicht einverstanden

Dauer der Sperre:

[ ] Jahre
[ ] unbefristet

Braunschweig, den 24. August 2007

Gerald Temme

Inhaltsverzeichnis

Abkürzungsverzeichnis

2D:	Zweidimensional
3D:	Dreidimensional
AOI:	Area Of Intrest
caSBAro:	Computer aided Situation and Behaviour Analysis replay / online
D:	Dwell
DD:	Dwell Duration
DLR:	Deutsches Zentrum für Luft- und Raumfahrt
EA:	Endlicher Automat
EOG:	Augenbewegungsmessverfahren: Elektro-Okulogramm
FD:	Fixation Duration
FFD:	First Fixation Duration
FS:	Institut für Verkehrsführung und Fahrzeugsteuerung
FSM:	Finite State Machine
GD:	Gaze Duration
GLD:	Glance Duration
HED:	Headmounted Eyetracking Device
HT:	Head Tracking
MMS:	Mensch-Maschine-Schnitstelle
OMG:	Object Management Group
OSG:	Open Scene Graph
RGB:	RGB-Farbraum (Rot Grün Blau)
SMI:	SensoMotoric-Instrumens
SMPL++:	Straigthforward Modular Prototyping Library in C++

SMPLab: Straightforward Modular Prototyping Laboratory

SSM: Smpl Shared Memory

STL: Standard Template Library

SW: Standardwert

UDP: User Datagram Protocol

UML: Unified Modeling Language

VT: Viewing time

XML: Extensible Markup Language

Kapitel 1

Einleitung

Aktuelle Fahrzeugmodelle verfügen schon heute über eine Vielzahl von den Fahrer unterstützenden und schützenden Systemen (ABS, IPA, Airbag,...). Mit dem technischen Fortschritt in den Bereichen Elektronik, Computer sowie Sensor- und Kamerasysteme entwickeln sich diese Systeme weiter.

Die neue Generation der Fahrerassistenzsysteme könnte teilautonom bis autonom in die Steuerung des Fahrzeuges eingreifen, um den Fahrkomfort zu steigern und das Autofahren sicherer zu machen. Dabei sollen sie den Fahrer nicht ersetzen, sondern ihn in Form von Signalen, wie Tönen oder Vibrationen und Bewegungen in den Bedienelementen des Fahrzeuges, auf bestimmte Sachverhalte aufmerksam machen.

Diese Signale stellen die direkte Verbindung (Schnittstelle) zwischen dem Menschen als Fahrer des Fahrzeuges und der Maschine in der Form des Assistenten dar. Für die Entwicklung und das Testen von neuen Konzepten zum intuititiven Informationsaustausch über die Schnittstelle ist eine Analyse und Auswertung dieser Mensch-Maschinen-Schnittstelle (MMS) unumgänglich. Eine Möglichkeit der Auswertung dieser MMS ist die Untersuchung der Reaktionen einer Testperson mittels Blickrichtungsmesssystemen.

Blickrichtungsmesssysteme werden nicht nur zur Analyse der MMS eingesetzt, sondern können auch Eingangsparameter für Fahrerassistenzsysteme zur Verfügung stellen. So

wird zum Beispiel von dem Automobilhersteller Lexus als Eingangsparameter für ein „Pre-Crash Safety System“ des Fahrzeugmodels „LS 460“ eine Kamera eingesetzt, die die Blickrichtung des Fahrers überprüft. Wird ein Hindernis mittels verschiedener Sensoren erkannt, warnt das System den Fahrer, wenn dieser sein Gesicht von der Straße abgewendet hat (siehe Lexus Webseite: „Ein Automobil, das selbst nachts Hindernisse mühelos erkennt“ [Lex07]).

1.1 Aufgabenstellung

Zusammengefasst lautet die Aufgabenstellung dieser Masterarbeit:

Entwicklung einer Klassenbibliothek zur Integration von Blickrichtungsmesssystemen im Allgemeinen und dem iViewX HED/HT System von SensoMotoric-Instrumens (SMI) [SMI07], im Speziellen in die am Institut FS Verwendung findende Applikations- und Entwicklungsplattform „Straigthforward Modular Prototyping Library in C++“ (Smpl++) zur Online-Auswertung, Analyse und Visualisierung der durch das Blickrichtungsmesssystem aufgezeichneten Augenbewegungsmessdaten für den Automotivebereich.

1.2 Zielsetzungen

Die Zielsetzungen innerhalb dieser Masterarbeit in Kürze:

1. Integration von Blickrichtungsmesssystemen (im Speziellen das von SMI) in die Smpl++-Plattform durch Einlesen der Messsystemdaten über eine UDP Remote-Control-Schnittstelle des verwendeten Messsystems. Die wichtigsten Messdaten sind hierbei: Timestamp, Eyeposition, normalized Gazeposition.

2. Echtzeitinterpretation der eingelesenen Messdaten zur Bestimmung einer vorliegenden Fixation oder Sakkade anhand zweier aufeinanderfolgender Messdatensätze. Definieren von Flächen (Planes) und Area Of Interests (AOIs) mit anschließender Echtzeittrefferinterpretation des Blickstrahls. Berechnen verschiedener Blickrichtungsanalyseparameter aus der Literatur z.B. Gaze Duration (definiert durch: Liversedge & Findlay, 2000 [LF00]; Starr & Rayner, 2001 [SR01]).

3. Visualisierung der durch die Analyse bestimmten Parameter in Diagrammen sowie in einem 3D-Modell des Labors zur weiteren Auswertung der Augenbewegungsmessdaten, z.B. für eine Mensch-Maschine-Schnittstellenbeurteilung eines neuen Fahrerassistenzsystemkonzeptes.

4. optional bei genügend Restzeit: Eine einfache Onlineanwendung zur Demonstration der Möglichkeiten der entwickelten Klassenbibliotheken.

5. Dokumentation der Klassenbibliotheken dieser Masterarbeit mittels dem freien Dokumentationstool Doxygen.

1.3 Vorstellung des DLR, des Instituts FS und des SMPLabs

Das DLR ist das Forschungszentrum der Bundesrepublik Deutschland für Luft- und Raumfahrt. Seine umfangreichen Forschungs- und Entwicklungsarbeiten in Luftfahrt, Raumfahrt, Energie und Verkehr sind in nationale und internationale Kooperationen eingebunden. Über die eigene Forschung hinaus ist das DLR als Raumfahrtagentur im Auftrag der Bundesregierung für die Planung und Umsetzung der deutschen Raumfahrtaktivitäten zuständig. [DLR07a]

Der Forschungsbereich des DLR reicht von der Grundlagenforschung bis zur Entwicklung innovativer Anwendungen und Produkten von morgen. Die ca. 5300 Mitarbeiterinnen und Mitarbeiter des DLR verteilen sich auf acht deutschlandweite Standorte. Diese Standorte beherbergen 28 Institute sowie eine Vielzahl von Test- und Betriebseinrichtungen.

Einer dieser Standorte ist Braunschweig (siehe Abbildung 1.1). Die Institute an diesem Standort haben die Forschungsschwerpunkte Luftfahrt und Verkehr. Den in Braunschweig ansässigen Instituten stehen leistungsfähige Fahr- und Flugversuchsträger sowie fliegende Simulatoren, Luftverkehrssimulationsanlagen, Fahrsimulatoren, Windkanäle aus dem europäischen Leistungsverbund DNW (Deutsch-Niederländische Windkanäle), mobile Rotorversuchsstände sowie Prüfstände für die Werkstoff- und Lärmforschung zur Verfügung.

Abbildung 1.1: DLR Braunschweig am Forschungsflughafen Braunschweig [DLR07a]

Der DLR-Standort Braunschweig beherbergt die folgenden fünf Institute:

- Institut für Aerodynamik und Strömungstechnik
- Institut für Faserverbundleichtbau und Adaptronik
- Institut für Flugführung
- Institut für Flugsystemtechnik
- Institut für Verkehrsführung und Fahrzeugsteuerung (FS)

Diese Masterarbeit entstand in Zusammenarbeit mit dem DLR-Institut FS [DLR07b]. Dieses Institut hat seine Forschungsschwerpunkte in den Bereichen:

- Automotive (Entwicklung und Bewertung neuer Assistenzsysteme mit dem Ziel, die Sicherheit und Effizienz im Straßenverkehr zu verbessern)
- Bahnsysteme (Entwicklung neuer Konzepte für eine sichere und wirtschaftliche Betriebsführung)
- Verkehrsmanagement (Entwicklung neuer Konzepte für Organisation und Betrieb von Verkehr mit dem Ziel, die Effizienz im Straßenverkehr zu erhöhen)

Die in dieser Masterarbeit entwickelte Klassenbibiliotek zur Integration von Blickrichtungsmesssystemen in die Applikations- und Entwicklungsplattform Smpl++ wurde für den Forschungsschwerpunkt Automotive Systeme entwickelt.

Das zu integrierende Blickrichtungsmesssystem von SMI befindet sich überwiegend in dem Straightforward Modular Prototyping Laboratory (SMPLab) (siehe Abbildung 1.2), welches als Teamarbeitsraum konzipiert ist. Dieses Labor verfügt neben dem Blickrichtungsmesssystem vor allem über einen Fahrsimulator (SmplSim) sowie mehrere Whitebords, Leinwände und Beamer. Das SMPLab ist optimiert auf Ideenfindung, Brainstorming, Agiles Prototyping & Evaluation sowie Besprechungen und Vorträgen. Die Aufgabenschwerpunkte des Fahrsimulator SmplSim sind dabei:

- Entwicklung von multimodalen (d.h. haptischen, visuellen, akustischen) Fahrerassistenzsystemen
- Entwicklung neuer Fahrerassistenzkonzepte
- Prototypentwurf neuer Fahrerassistenten
- Untersuchungen im Bereich Mensch-Maschine-Schnittstelle (MMS)

Abbildung 1.2: Abbildung zeigt eine 180° Ansicht des SMPLabs in Richtung des Simulators SmplSim [Quelle: DLR Institut FS]

Die Integration von Blickrichtungsmesssystemen in die Smpl++-Plattform sollte insbesondere die Untersuchungen in dem Bereich der MMS unterstützen und nach Möglichkeit vereinfachen.

1.4 Kapitelvorschau

Diese Arbeit ist wie folgt aufgebaut:

Im Kapitel 2 „Grundlagen“ werden sowohl die Grundlagen zum besseren Verständnis der Programmierung, als auch die Grundlagen für das Arbeiten mit Blickrichtungsmesssystemen beschrieben.

Das Kapitel 3 „Softwareentwicklung - Projektmanagement“ beinhaltet das Lastenheft und den Zeitrahmen zur Umsetzung dieser Masterarbeit.

Im Kapitel 4 „Softwareentwicklung - Entwurf“ werden die vor dem Beginn der Implementierung entwickelten Entwürfe der einzelnen Module vorgestellt.

Das Kapitel 5 „Softwareentwicklung - Implementierung & Dokumentation“ beschreibt die umgesetzten Smpl++-EyeTracking-Module zum Empfangen von Blickrichtungsmessdaten, Interpretation der empfangenen Daten und die Visualisierung dieser Daten.

Im Kapitel 6 „Softwareentwicklung - Verifikation“ sind die Tests zur Überprüfung der durch die implementierten Klassen und Module erzeugen Daten beschreiben.

In Kapitel 7 „Zusammenfassung“ werden die erreichten Ziele dieser Arbeit benannt, ein Soll-Ist-Vergleich der Zeitplanung durchgeführt sowie ein Ausblick auf mögliche Verbesserungen und Ergänzungen gegeben.

Den Abschluss dieser Arbeit bildet das Kapitel 8 „Anhang“ gefolgt vom Abbildungs-, Listing- und Literaturverzeichnis.

Kapitel 2

Grundlagen

Im letzten Kapitel wurde einleitend die Zielsetzung dieser Masterarbeit beschrieben. Anhand eines Lastenheftes wurde aus der Zielsetzung eine konkrete Aufgabenstellung entwickelt, welche in den folgenden Kapiteln umgesetzt und dokumentiert wird. Zusätzlich verdeutlicht das letzte Kapitel anhand eines Zeitplans die verschiedenen Arbeitsabschnitte und deren zeitlichen Rahmen innerhalb der Masterarbeit.

Dieses Kapitel beinhaltet die notwendigen Grundlagen zum besseren Verständnis der Masterarbeit. Zu diesen Grundlagen gehört eine Einführung in die Programmierung sowie eine Beschreibung der bestehenden Applikations- und Entwicklungsplattform SMPL++ für Fahrerassistenzsysteme im Automotivebereich. Zusätzlich wird das SMPL++ zugrunde liegende Softwareentwicklungsmodell vorgestellt.

Neben einer Beschreibung von SMPL++ beinhaltet dieses Kapitel die benötigten Grundlagen zum Verständnis der Bewegungsmesssystemparameter. Hier wird insbesondere auf die beiden wichtigsten Begriffe, Sakkade und Fixation, eingegangen. Es folgt eine allgemeine Vorstellung verschiedener Bewegungsmessverfahren. Abschließend werden die am DLR-Standort Braunschweig vorhandenen Bewegungsmesssysteme vorgestellt.

2.1 Programmierung

Zum besseren Verständniss verschiedener Fachbegriffe in der Dokumentation stellt dieses Unterkapitel die benötigten Grundlagen bereit. Insbesondere werden im Abschnit 2.1.7 die bei der Umsetzung der Klassenbibliothek verwendeten Module der Applikations- und Entwicklungsplattform vorgestellt.

2.1.1 Doxygen

Doxygen ist ein Open-Source Software-Dokumentationswerkzeug und wurde von Dimitri van Heesch entwickelt. Durch spezielle Kommentare im Quelltext können Software-Entwickler Erläuterungen zu Programmelementen definieren, aus denen Doxygen eine übersichtliche Dokumentation erstellt. Außerdem ist es möglich, einen zusammenfassenden Überblick über den Aufbau und die Elemente eines bereits existierenden Programms (verwendete Dateien, Funktionen, Variablen sowie deren Rolle im Programmablauf) zu erzeugen.

Doxygen unterstützt die Programmiersprachen C, C++, Java, Python und IDL. Weiterhin werden die Programmiersprachen PHP, C# und D zum größten Teil von Doxygen unterstützt.

Mögliche Ausgabeformate für die durch Doxygen erzeugte Dokumentation sind HTML, CHM, LaTeX, XML, RTF, PostScript, PDF und Man page.

Weitere Informationen und Download von Doxygen gibt es auf der offiziellen Webseite: [Dox07]

2.1.2 UML

Die Unified Modeling Language (UML) ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme. Die UML berücksichtigt die gestiegenen Anforderungen bezüglich der Komplexität heutiger Systeme, deckt ein breites Spektrum von Anwendungsgebieten ab und eignet sich für konkurrierende, verteilte, zeitkritische, sozial eingebettete Systeme. [Oes05]

Alle in dieser Dokumentation verwendeten UML-Diagramme wurden in der Version UML 2.0 erstellt. Die UML2 kennt laut [Oes05] 15 verschiedene Diagrammtypen. Neun dieser Diagramme gehören zur Gruppe der Strukturdiagramme:

1. Klassendiagramm
2. Objektdiagramm
3. Anwendungsfalldiagramm
4. Paketdiagramm
5. Zusammenarbeitsdiagramm
6. Kompositionsstrukturdiagramm
7. Komponentendiagramm
8. Subsystemdiagramm
9. Einsatz- und Verteilungsdiagramm

Die restlichen sechs Diagrammtypen werden der Gruppe von Verhaltensdiagrammen zugeordnet:

1. Aktivitätsdiagramm
2. Zusatandsdiagramm
3. Sequenzdiagramm
4. Kommunikationsdiagramm
5. Zeitdiagramm
6. Interaktionsübersichtsdiagramm

Von dieser Vielzahl von Diagrammtypen werden in dieser Arbeit Klassendiagramme zur Beschreibung der Klassenbeziehungen, ein leicht abgewandeltes Sequenzdiagramm zur Beschreibung des Nachrichtenaustausch zwischen dem Blickrichtungsmesssystem und der Applikations- und Entwicklungsplattform, sowie ein Zustandsdiagramm zur Beschreibung der Berechnung von Analyseparametern mittels eines Zustandsautomaten verwendet.

In den erstellten Klassendiagrammen sind alle innerhalb dieser Arbeit entstandenen Klassen bläulich eingefärbt. Verwendete Klassen aus bestehenden Smpl++-Modulen sind durch eine gelbliche Einfärbung gekennzeichnet. Der jeweils in einem kräftigeren Farbton markierte Abschnitt enthält die `main()` Funktion des in dem Klassendiagramm dargestellten Modules.

Eine genaue Beschreibung der verschiedenen Diagrammtypen befindet sich zum Beispiel in [Oes05] [OMG07] [Wiki07c].

2.1.3 XML

Die Extensible Markup Language[1] (XML) ist eine Metasprache, also eine Sprache zum beschreiben von Sprachen.
Die XML-Spezifikationen werden vom World Wide Web Consortium (W3C) herausgegeben und definieren eine Metasprache als Basis für verschiedene anwendungsspezifische XML-Sprachen. Beispiele für XML-Sprachen sind RSS, MathML, GraphML, XHTML oder XML-Schema.

Für eine Beschreibung der Metasprache und dem Aufbau von XML-Dokumenten siehe [BM07] [Wiki07f].

[1] Extensible Markup Language; engl. für „erweiterbare Auszeichnungssprache“

2.1.4 C++

C++ ist eine standardisierte, höhere Programmiersprache. Sie wurde in den 1980er Jahren von Bjarne Stroustrup bei AT&T als Erweiterung der Programmiersprache C entwickelt. C++ wurde als Mehrzwecksprache konzipiert. Sie unterstützt mehrere Programmierparadigmen, wie die objektorientierte, generische und prozedurale Programmierung, und ermöglicht sowohl die effiziente und maschinennahe, als auch eine Programmierung auf hohem Abstraktionsniveau. [Wiki07g]

Siehe auch [Cpp07], [PK05] oder das Standartwerk [STR00].

2.1.5 OpenSceneGraph

OpenSceneGraph (OSG) ist eine freie 3D-Grafik-Bibliothek zum Entwickeln von leistungsfähigen grafischen Anwendungen wie etwa Flugsimulationen, Spiele sowie Virtual Reality und wissenschaftlichen Visualisierungen. Basierend auf dem Konzept eines Szenengraphen[2] stellt OSG einen objektorientieten Rahmen mittels OpenGL zur Verfügung. OSG verfügt zusätzlich über eine Vielzahl von Dienstprogrammen, die ein schnelles Entwickeln von Grafikanwendungen ermöglichen. [OSG07]

2.1.6 Applikations- und Entwicklungsplattform Smpl++

Die vom DLR verwendete Applikations- und Entwicklungsplattform „Straightforward Modular Prototyping Library in C++“ (Smpl++) entstand ursprünglich am NASA Langley Research Center, beeinflusst und angeregt durch die Erfahrungen aus dem europäischen Prometheus-Projekt2[3] und dem deutschen CAMA-Projekt3[4]. Smpl++ wird aktuell sowohl bei der NASA (Schwerpunkt Luftfahrt) als auch beim DLR (Schwerpunkt Bodenfahrzeuge) eingesetzt und weiterentwickelt.

[2] Ein Szenengraph bzw. Szenengraf ist eine Datenstruktur, die häufig bei der Entwicklung computergrafischer Anwendungen eingesetzt wird. Es handelt sich um eine objektorientierte Datenstruktur, mit der die logische, in vielen Fällen auch die räumliche Anordnung der darzustellenden zwei- oder dreidimensionalen Szene beschrieben wird.

[3] Projektschwerpunkt autonome Fahrzeuge

[4] CAMA: Cockpit Assistant Military Aircraft; Projektschwerpunkt Cockpit-Assistenzsysteme

Der Begriff „**Straightforward**“ im Namen der Entwicklungsplattform bedeutet unter anderem, dass der entwickelte Code selbsterklärend sein muss, dass verwendete Drittsoftware OpenSource ist und die mit Smpl++ entwickelten Simulationen auf einen handelsüblichen PC laufen kann.

„**Modular**“ beschreibt, dass die einzelnen Module der Plattform kleinere Aufgaben erledigen, die in Prozessen ablaufen und über ein Blackboard miteinander kommunizieren (siehe Abschnitt: 2.1.7). Außerdem sind die entwickelten Module weitestgehend plattformunabhängig.

Smpl++ ist demzufolge eine Zusammenstellung von Modulen (siehe 2.1.7) die auf C++-Bibliotheken und -Prozessen beruhen. Die Weiterentwicklung von Smpl++ geschieht beim DLR auf Basis eines Agilen Softwareentwicklungsmodells[5] . Einige der Prinzipien der Agilen Softwareentwicklung sind:

- Personen und Kommunikation sind wichtiger als Prozesse und Tools
- lauffähige Software ist wichtiger als eine umfassende Dokumentation
- Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen
- Reaktion auf Veränderungen ist wichtiger als strikte Planbefolgung
- für weitere Prinzipien des „Agile Manifesto“ siehe [Agil07]

Der gering bürokratische Ansatz des Agilen Softwareentwicklungsmodells findet sich unter anderem in den überwiegend öffentlich (= public) gehaltenen Variablen und Methoden der Klassen der Smpl++ Module wieder. Die wenigen Zugriffsbeschränkungen fördern ein schnelles, zielorientiertes Programmieren sowie Wiederverwenden von Klassen und Modulen innerhalb der Smpl++ Plattform. Dies ist möglich, da die Smpl++ Applikations- und Entwicklungsplattform nur im sicherheitsunkritischen, internen Bereich des DLR verwendet wird, und so keine Gefahr droht durch Dritte „gehackt“ oder „missbraucht“ zu werden.

[5] Agile Softwareentwicklung ist der Oberbegriff für den Einsatz von Agilität in der Softwareentwicklung. Agile Softwareentwicklung zeichnet sich durch geringen bürokratischen Aufwand und wenige, flexible Regeln aus. [Wiki07b]

Das Ziel der Smpl++ Plattform ist es nicht durch einen perfekt durchstrukturierten und dokumentierten Code zu glänzen, sondern möglichst schnell, einfach und effektiv entwickelte Fahrerassistenzkonzepte in Module umszusetzen, diese in der Praxis zu erproben (Simulator, Testfahrzeuge) und während der Erprobung aufgezeichnete Daten zu analysieren und zu dokumentieren.

Die mittels der Smpl++ Plattform umgesetzten Fahrerassistenzkonzepte stellen somit die ersten Prototypen bei der Entwicklung eines neuen Fahrerassistenten dar.

2.1.7 Smpl++-Module

Die Applikations- und Entwicklungsplattform Smpl++ setzt sich aus einer Vielzahl von Modulen zusammen. Die meisten dieser Module decken einen speziellen Aufgabenbereich ab. Ohne dass es eine offizielle Zuordnung der bestehenden Smpl++ Module zu einer bestimmten Kategorie gäbe, kann man die Module vereinfacht drei verschiedenen Kategorien zuordnen (siehe Abbildung: 2.1).

Die unterste Kategorie bilden **Basismodule**, die vor allem häufig durch andere Module verwendete Funktionalitäten bereitstellen. Das Modul „`SmplEthernet`“ oder auch „`SmplSharedMemory`“ (siehe Modulbeschreibungen weiter unten) gehören zu dieser Kategorie.

Die mittlere Kategorie bilden **Datenverarbeitungsmodule**. Darunter fallen z.B. Module, die die Fahrphysik der simulierten Fahrzeuge bestimmen oder die das Verhalten eines Assistenten bei bestimmten Ereignissen festlegen. Das Modul „`SmplEyeTrackingDataInterpretation`“ ist ein Beispiel für ein Modul dieser Kategorie.

Die oberste Kategorie ist die der **Interaktionsmodule**, die uns eine Kommunikation mit den in Smpl++ erstellten Automationen ermöglichen. Hierunter fallen Module zur Steuerung bzw. zum Auslesen der angeschlossenen aktiven haptischen Systeme, Module

zum Darstellen der 3D-Modelle wie das SMPLab oder der zu befahrenden Teststrecke „`SmplFSViewer`“ oder auch das Modul „`SmplcaSBAro`“ zum Speichern, Analysieren und Dokumentieren des „Shared-Memorys“.

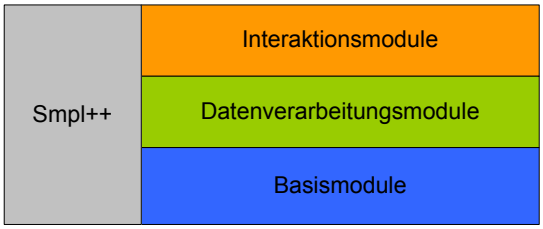


Abbildung 2.1: Grobeinteilung der Smpl++ Module in drei Kategorien

Die Folgenden Abschnitte stellen die innerhalb dieser Arbeit verwendeten Smpl++ Module kurz vor.

SmplEthernet

Das Modul „`SmplEthernet`“ beinhaltet die Klassen „`SmplTcpip`“ und „`SmplUDP`“, die Methoden zum Empfangen und Senden von Datenpaketen über das Ethernet mit den Protokollen TCP/IP (Transmission Control Protocol/Internet Protocol) [Wiki07e] und UDP (User Datagram Protocol) [Wiki07d] zur Verfügung stellen. Das in dieser Arbeit entwickelte Modul „`SmplEyeTrackingRemoteControl`“ zur Kommunikation mit dem Blickrichtungsmessystem (siehe Kapitel 5.1 Datenerfassungsmodul) verwendet für diese Kommunikation die Klasse „`SmplUDP`“.

SmplSharedMemory

Das Modul „`SmplSharedMemory`(SSM)“ beschreibt den Datenaustausch über ein Shared-Memory und liefert damit die Implementierung einer speziellen Blackboardarchitektur. Diese Architektur ermöglicht einen schnellen Wechsel von Modulen mit ähnlicher Funktionalität. So ist es zum Beispiel möglich,zur Laufzeit eines Versuches unkompliziert die Steuerung des Fahrzeugs von einem Sidestick auf ein Lenkrad zu wechseln (siehe Abbildung: 2.2).

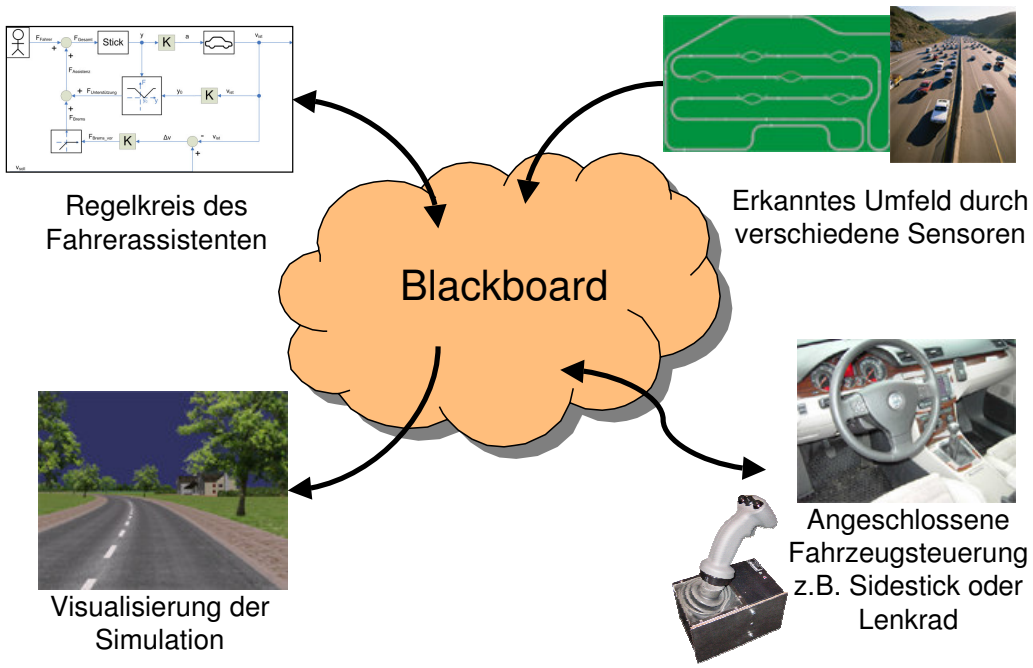


Abbildung 2.2: Blackboardarchitektur [Quelle: DLR-FS]

Das Modul „`SmplSharedMemory`“ verfügt zur Realisierung des Shared-Memorys über eine Vielzahl von Klassen und Komponenten. So stellt das „SSM“ zum Beispiel einen Timer „`SmplSharedTimer`“ zur Verfügung, welcher allen beteiligten Modulen eine einheitliche Zeit vorgibt und durch Beeinflussung dieses Timers auch ein Zeitraffer oder ein Einfrieren der Module ermöglicht. Ausserdem besitzt das „SSM“ Modul Klassen zur Administration und Zugriffssteuerung des Shared-Memorys, sowie Klassen, die verschiedene Teile der „Standard Template Library (STL)“ für den „SSM“ verwendbar machen. Dies erfolgt durch Vorgabe fester Längen. So ermöglicht die Klasse „`SmplString50`“ das Speichern von Strings mit maximal 50 Zeichen Länge im „SSM“.

Eine detailliertere Beschreibung der einzelnen Klassen und Komponenten des Moduls „SSM“ befindet sich in der Diplomarbeit [Ju06].

SmplcaSBAro

Das Modul „`SmplcaSBAro`“ (computer aided Situation and Behaviour Analysis replay/online) stellt der Smpl++ Plattform ein Online-Analyse-, Aufzeichnungs- und Replay-Tool zur Verfügung. „`SmplcaSBAro`“ ermöglicht ein Aufzeichnen des gesamten Inhalts des Blackboards und somit der gesammten Kommunikation zwischen allen aktiven Modulen. Zusätzlich ermöglicht „`SmplcaSBAro`“ das datensynchrone Aufzeichnen von mehreren Videoströmen, sodass sowohl das Zusammenspiel der verschiedenen Module einer Automation als auch das Zusammenspiel zwischen Mensch und Technik genau analysiert und wiedergegeben werden kann. Eine Visualisierung der Modulkommunikation kann durch „`SmplcaSBAro`“ in Form von Diagrammen erfolgen. Ausserdem ermöglicht „`SmplcaSBAro`“ ein Replay der aufgezeichneten Modulkommunikation. Innerhalb dieses Replays kann jede aufgezeichnete Situation zu jedem Zeitpunkt sofort wieder hergestellt werden. Wird zusätzlich zu der Wiedergabe der aufgezeichneten Daten zum Beispiel das Modul „`SmplFSViewer`“ gestartet, können unter anderem die genauen Fahrzeugpositionen und Geschwindigkeiten der aufgezeichneten Situationen in der 3D-Visualisierung des Simulators erneut dargestellt werden. Dies ermöglicht eine genaue Analyse der Einzelsituationen (siehe Abbildung: 2.3).

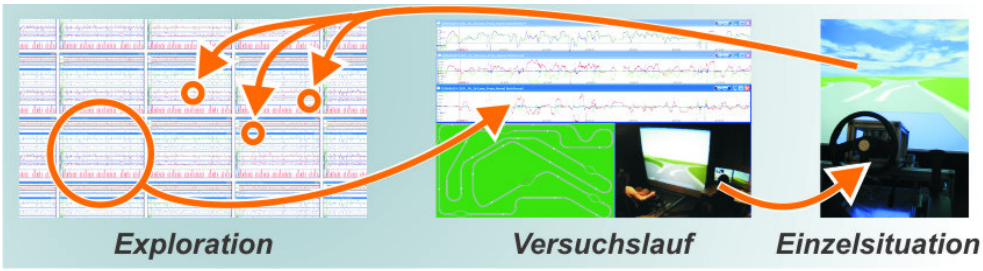


Abbildung 2.3: Darstellung des Smpl++ Moduls „SmplcaSBAro“ [Quelle: DLR-FS]

SmplFSViewer

Das Modul „`SmplFSViewer`“ ermöglicht eine 3D-Visualisierung der zu befahrenden Teststrecke, Hindernissen, Trajektorien, Head-Up-Displays (Geschwindigkeitsanzeigen etc.) und einiges mehr sowie nach Abschluss dieser Arbeit der aufgenommen Blick-

richtungsmessdaten in einem 3D-Modell (siehe Kapitel: 5.3.3) mittels der freien 3D-Bibliothek OpenSceneGraph (OSG) [OSG07] (siehe Abbildung: 2.4).

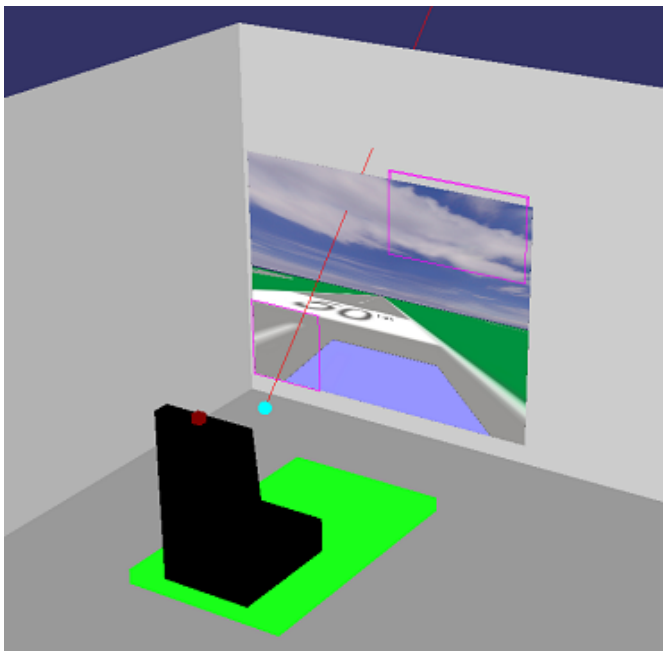


Abbildung 2.4: Visualisierung des SMPLabs als 3D-Modell mit dargestelltem Blickstrahl durch das Smpl++-Modul „SmplFSViewer“

Die Ursprungsversion dieses Viewers ist eine auf OSG basierende Entwicklung des DLR für einen anderen Simulator am Standort Braunschweig. Diese Urversion wurde Ende 2006 umgebaut und an die Smpl++-Plattform angepasst.

2.2 Augenbewegungsmesssystem

Die visuelle Wahrnehmung (Sehen) ist der wichtigsten Sinn des Menschen zum Erfassen seiner Umgebung. Die Augen versetzen uns in die Lage, Farben, Formen und Bewegungen zu erkennen, und geben uns die Möglichkeit der Orientierung im Raum. Mit dem Auge alleine können wir allerdings noch gar nichts wahrnehmen. Das Auge selbst ist vielmehr eine hochentwickelte Kamera mit einer ausgeklügelten Optik, welche ihre Bildinformationen als elektrische Signale an das Gehirn weitersendet. Aus dem Zusammenspiel von Auge und Gehirn entsteht erst unsere Wahrnehmung.

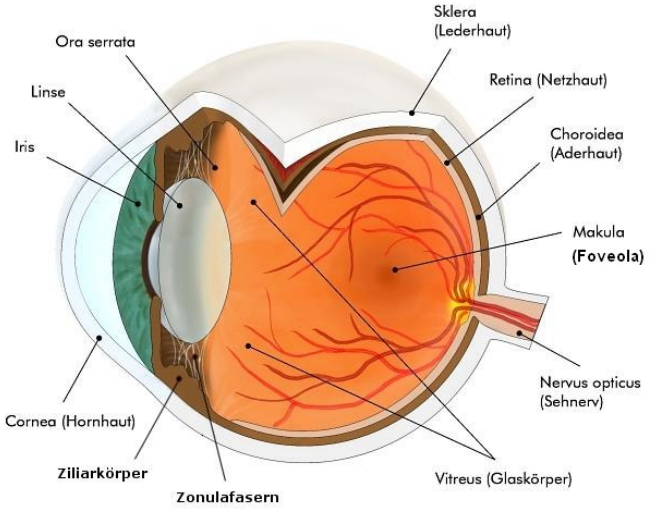


Abbildung 2.5: Grafische Darstellung des menschlichen Auges
[Quelle: http://www.ocunet.de/patienten/auge.html]

Im Auge (siehe Abbildung 2.5) werden die eintreffenden Lichtstrahlen, welche von den Objekten im Sichtfeld reflektiert werden, von den Stäbchen und Zäpfchen auf der Netzhaut in elektrische Signale umgewandelt. Dabei nehmen wir die betrachteten Objekte nicht in allen Bereichen der Netzhaut gleich scharf wahr. Es gibt vielmehr einen Punkt des schärfsten Sehens. Dieser Punkt wird als Foveola[6] bezeichnet. Die Foveola befindet sich in der Mitte der Netzhaut. Wenn wir ein bestimmtes Objekt betrachten wollen, fixieren wir dieses mit der Foveola ([Fi99] S.42-44).

[6] Weitere Bezeichnungen für Foveola: Gelber Fleck, Makula

Wenn wir ein Objekt willentlich oder ausgelöst durch einen Reiz fixieren, so verweilen wir eine gewisse Zeit mit der Foveola auf diesem Punkt. Dies bezeichnet man auch als Fixation.

Die von der Foveola ausgehenden elektrischen Signale der Netzhaut werden vom Gehirn bevorzugt verarbeitet. Die restlichen von der Netzhaut weitergegebenen Signale werden untergeordnet verarbeitet. Mit letzteren wird vor allem die Position des als nächstes zu betrachtenden Objektes in der Foveola ermittelt. Dieser kann willentlich festgelegt, aber auch durch einen Reiz automatisch bestimmt werden.

Wurde ein neuer Punkt für die nächste Fixation bestimmt, erfolgt eine schnelle Augenbewegung auf die berechneten Zielkoordinaten. Diese schnelle Bewegung der Augen nennt man Sakkade. Sie wird durch die Augenmuskeln ausgeführt. Die Steuerung der Augenmuskeln (siehe Abbildung 2.6) übernimmt hierbei das Gehirn unter Berücksichtigung vieler Variablen wie der eigenen Bewegung des Körpers, der Bewegung des zu betrachtenden Objektes und einiger anderer Parameter.

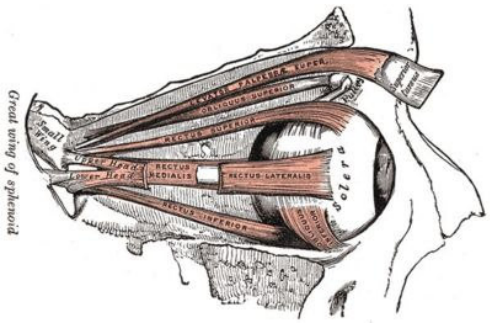


Abbildung 2.6: Grafische Darstellung der Augenmuskeln
[Quelle: http://de.wikipedia.org/wiki/Bild:Eyemuscles.jpg]

Während des natürlichen Sehens vollführen wir ständig Fixationen zum Betrachten ausgewählter Punkte, gefolgt von Sakkaden, die die Foveola auf den nächsten zu betrachtenden Fixationspunkt bewegen. Es gibt eine Vielzahl von unterschiedlicher Sakkaden für die verschiedensten Formen der Bewegung des Auges (siehe Abbildung 2.7). Nachfolgend werden die für diese Arbeit wichtigsten Augenbewegungsformen beschrieben.

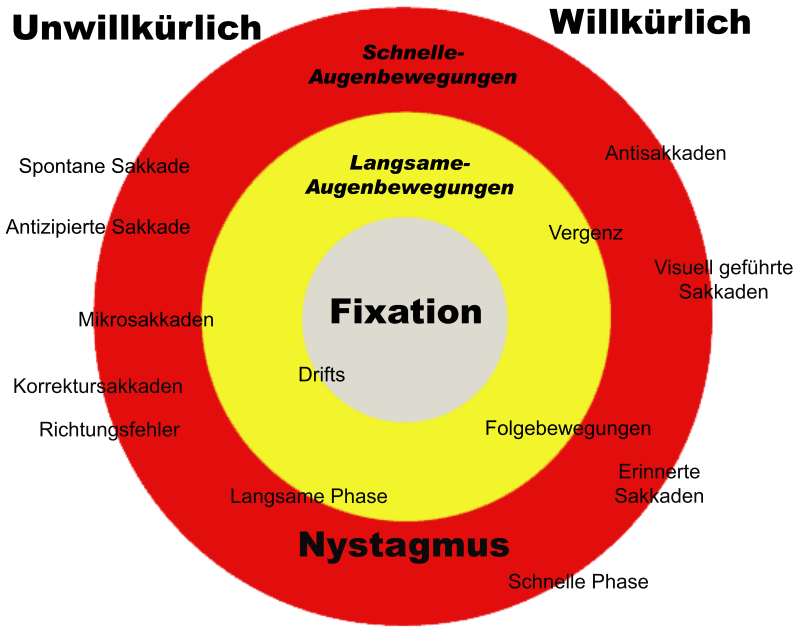


Abbildung 2.7: Grafische Darstellung der verschiedenen Augenbewegungen Quelle: [Fi99]

Die Abbildung 2.7 stellt die verschiedenen willkürlichen und unwillkürlichen Augenbewegungen grafisch dar. Je näher die Bewegungsform an der Fixation liegt, um so langsamer findet sie statt. Die schnellsten Bewegungen können hierbei Geschwindigkeiten bis zu 500 Grad pro Sekunde erreichen.

2.2.1 Fixation

Als Fixation wird in der Augenheilkunde das Betrachten (Fixieren) eines Objektes im Punkte der höchsten Netzhautauflösung der Foveola bezeichnet. Die zeitliche Dauer einer Fixation liegt in der Größenordnung von 100 bis 300 ms. Fixationen der Länge 170 ms treten hierbei bevorzugt auf. Der fixierte Punkt im Raum wird als Fixationspunkt bezeichnet ([Fi99] S.128).

2.2.2 Drift, Nystagmus und Vergenz

Während einer Fixation steht das Auge keinesfalls vollkommen still. Es treten permanent kleinere Augenbewegungen auf [JRV07]:

- Um die Funktionen der visuellen Rezeptoren der Netzhaut aufrecht zu erhalten, vollführt das Auge eine beständige leichte Zitterbewegung, welche als Nystagmus bezeichnet wird.

- Es kommt gelegentlich zu einem Driften des Auges aus dem Fixationspunkt. Verursacht wird diese langsame, ungewollte Bewegung durch eine ungenügende Kontrolle der Augenmuskeln.

- Die Vergenz bezeichnet das Gegeneinanderschieben der Blickachse, um Objekte unterschiedlicher Entfernungen zu fixieren. Dies ist eine bewusste Augenbewegung und dient zum Halten eines Fixationspunktes, wenn der Kopf oder auch der ganze Körper sich beispielsweise davon wegdrehen. Ein Beispiel für den Einsatz der Vergenz bietet das betrachten eines Objektes in der Ferne durch eine verschmutzte Glasscheibe hindurch. In dieser Situation ist es möglich, zuerst das Objekt in der Ferne und anschließend, ohne den Fixationspunkt zu verändern, einen Dreckspritzer direkt vor uns deutlich zu betrachten (scharf zu stellen).

2.2.3 Sakkaden

Wird ein neuer Fixationspunkt bestimmt, so erfolgt ein ruckhafter Sprung auf diesen Punkt. Diesen schnellen Blicksprung bezeichnet man als eine Sakkade. Während des natürlichen Sehens erfolgt 3-5 mal in der Sekunde eine solche Sakkade. Die Größenordnung dieser Blicksprünge beträgt dabei 1 bis 10 Grad. Die Augen werden während einer Sakkade etwa 30 ms lang bewegt. Während der Durchführung dieser Bewegung nimmt das Auge keine visuellen Informationen auf. Neben dieser Standardform der Sakkaden gibt es verschiedene Spezialsakkaden. Die wichtigsten Spezialsakkaden sind hierbei ([Fi99] S.120,ff):

Mikrosakkaden

Zum Ausgleich von ungewollten Drifts werden Mikrosakkaden vollzogen. Diese Sakkaden sind wegen ihrer geringen Größe nicht sehr schnell (betrachtet im Verhältnis zu einer normalen Sakkade), aber schneller als das Driften. Mikrosakkaden nehmen wir in der Regel[7] genau wie den Drift nicht wahr.

Korrektursakkaden

Die Sakkaden treffen regelmässig nicht den anvisierten Fixationspunkt. Es kommt zu sogenannten *overshoots* oder *undershoots*. In diesen Fällen erfolgt eine Korrektursakkade. Warum es zu Abweichungen vom Ziel kommt, ist zurzeit noch nicht geklärt und Gegenstand von Theorien und Forschung.

2.2.4 Folgebewegung

Sakkaden werden vor allem beim Betrachten von ruhenden Objekten oder auch beim Lesen eingesetzt. Für das Beobachten sich bewegender Objekte gibt es ein eigenes Augenfolgesystem. Es ermöglicht den Fixationspunkt nahe bei dem sich bewegenden Objekt zu halten. Dabei können die Augen bei Geschwindigkeiten von Objekten von bis zu 50 °/s in einer glatten Bewegung folgen. Wird das bewegte Objekt jedoch noch schneller, bleibt das Auge bzw. der Fixationspunkt hinter dem sich bewegenden Objekt zurück. In diesem Fall erfolgt immer wieder eine mittelschnelle Sakkade (ca. 250 °/s, siehe Folgebewegung in Abbildung 2.7), um mit dem Fixationspunkt zurück auf das Objekt zu springen. Man unterscheidet hierbei grundsätzlich zwei verschiedene Bewegungsformen, welche reflexgesteuert ablaufen ([Fi99] S.123-125):

- Optokinetik:
  Beim Auto- oder Zugfahren werden wir bewegt und die Bilder ziehen an uns vorbei. Unser Sehsystem geht davon aus, dass wir uns selbst in Ruhe befinden und versucht, die vorbeiziehenden Bilder auf unserer Netzhaut zu stabilisieren.

[7] Ausnahme: In einer dunklen Umgebung scheint sich ein fixierter heller, kleiner Punkt langsam hin und her zu bewegen.

Es erfolgt eine Augenbewegung, die dem fixierten Objekt folgt. Das verfolgen des Objektes ist allerdings nur begrenzt möglich. So kommt es zu einer reflexhaften Folge von langsamen Folgebewegungen (Stabilisierungen), gepaart mit schnelleren Rückbewegungen. Dieser Reflex kann zum Beispiel während des Zugfahrens beim Fixieren der Schwellen des benachbarten Gleises beobachtet werden. Um eine solche Zitterbewegung zu vermeiden, muss ein relativ stationäres Objekt (Strommast am Horizont) im Verhältnis zur eigenen Position im fahrenden Zug fixiert werden.

- vestibuläre Kompensation:
  Wenn wir uns selbst bewegen, sorgt ein eigenes System dafür, dass das Bild auf der Netzhaut stabil bleibt. Gesteuert wird dies durch den Gleichgewichtsapparat des inneren Ohres. Die Beschleunigungen des Kopfes werden dort erfasst und so an die Augenmuskeln weitergegeben, so dass die Augen die Bewegungen des Kopfes kompensieren können.

Es kann in besonderen Situationen im Zusammenhang mit den Folgebewegungen zu gefährlichen Wahrnehmungstäuschungen kommen. Bei Bewegungen in erhöhter Geschwindigkeit über einen längeren Zeitraum hinweg, zum Beispiel auf der Autobahn, kommt es zu einer Gewöhnung an den konstant andauernden, großflächigen Bewegungsreiz der vorbeiziehenden Bilder auf der Netzhaut. Die Folge ist, dass wir unsere eigene Geschwindigkeit unterschätzen.
Sehen wir im umgekehrten Fall großflächig ein konstantes Bild, welches sich auf einmal in Bewegung setzt (etwa der abfahrende Zug auf dem Nachtbarkleis am Bahnhof), gewinnen wir den Eindruck, wir würden uns selbst in Bewegung setzen. Dieser Effekt kann auch beim Autofahren, etwa durch Nebel, hervorgerufen werden und falsche Steuerbewegungen provozieren.

2.2.5 Gängige Augenbewegungsmessverfahren

Zum Messen der Bewegungen des Auges existieren verschiedene Messverfahren. Die wichtigsten Parameter dieser Verfahren sind die Genauigkeit mit welcher die Drehbewegung des Auges in Grad bestimmt werden kann, sowie die zeitliche Auflösung des

Messverfahrens. Ideal wäre ein Messverfahren mit der Genauigkeit von einer Bogenminute[8] und einer zeitlichen Auflösung von einer Millisekunde ([Fi99] dort Kapitel 5 oder [JRV07]). Nicht jedes Messverfahren kann dieses Ideal erreichen. Die folgenden Verfahren sind die zur Zeit verbreitetsten und haben jeweils ihre Vor- und Nachteile:

Elektro-Okulogramm

Das Auge als Ganzes stellt einen elektrischen Dipol dar, der je nach Drehstellung des Auges zu verschiedenen kleinen Strömen zwischen zwei Elektroden führt. Diese Eigenschaft nutzt die Elektro-Okulogramm (EOG)- Messmethode. Es werden Metalplättchen als Elektroden neben die Augen geklebt. Anhand des gemessenen Stromes kann eine zeitlich sehr genaue Messung vorgenommen werden. Allerdings ist die örtliche Auflösung dieser Methode erst ab 1 bis 2 Grad zuverlässig. Ein Vorteil ist, dass das Vefahren auch bei geschlossenen Augen Messdaten liefern kann. [Fi99]

Magnetische Induktionsmethode

Bei diesem Verfahren wird das Auge künstlich mit elektromagnetischen Eigenschaften in From von Kontaktlinsen mit eingebetteter Minidrahtspule ausgestattet. Um den Kopf herum wird anschließend ein senkrecht stehendes Magnetfeld erzeugt. Die in der Spule entstehenden Induktionsströme sind ein Maß für den Winkel der Spule zum Magnetfeld. Man erreicht eine sehr gute zeitliche, sowie örtliche Auflösung der Messdaten. Dies ist die zur Zeit genauste verfügbare Messmethode. Allerdings birgt die Nutzung der Kontaktlinsen einige Risiken[9] und eignet sich nicht für längere Messungen. [Fi99]

Infrarot-Reflexmethode

Das Auge reflektiert einen großen Teil des Infrarotlichtes, welches auf seine Oberfläche trifft. Bei einer Beleuchtung des Auges mit Infrarotlicht aus etwa 2cm Entfernung reflektiert dieses das Licht je nach Augenstellung unterschiedlich. Dies gelingt nur, da das Auge keine homogene Kugel ist. Die reflektierten Infrarotstrahlen werden von Photo-

[8] Die Bogenminute (auch Winkelminute oder Minute, engl. arcmin) ist der 60ste Teil eines Winkelgrads
[9] z.B. Schädigung der Hornhaut möglich

zellen links und rechts neben dem Auge aufgefangen und in elektrische Signale umgewandelt. Aus der Differenz der Signale lässt sich der Drehwinkel des Auges bestimmen. Die Infrarot-Reflexmethode erlaubt eine zeitliche Auflösung von einer Millisekunde und eine Winkelauflösung von ein bis zwei Zehntel Grad. Die Einsatzdauer dieser Methode ist allerdings auf wenige Sekunden begrenzt, denn bei längerem Einsatz wird das Verfahren aufgrund von langsamen Drifts der Signale instabil. [Fi99]

Video-Methode

Mit steigender Leistungsfähigkeit der Computer gewann die Video-Methode an Möglichkeiten. Anhand der Position fester Merkmale im Auge, wie etwa der Pupille, berechnet der Computer die Augenbewegungen. Für eine gleichmäßigen Ausleutung des Auges, ohne das dieses geblendet wird, wird Infrarotlicht verwendet. Aufgenommen wird das Auge aus geringer Entfernung zumeist mit einer kleinen infrarotempfindlichen Kamera. Aufgrund der Bildwechselfrequenz der eingesetzten Kameras von 50 - 60 Hz (etwa 20 ms) ist die zeitliche Auflösung sehr schlecht. Es gibt Kameras, die eine Auflösung von 2 bis 4 ms ermöglichen, diese sind aber auch dementsprechend teuer. Die örtliche Auflösung ist abhängig von den eingesetzten Algorithmen der Analysesoftware bei der Berechnung der Augenbewegungen. Die Berechnung der Augenbewegungen erfolgt anhand von Merkmalen wie der Pupille und der direkten Spiegelung der Infrarotlampe auf der Hornhaut aufgenommenen Bild. Der Vorteil dieser Methode ist die Möglichkeit der Augenbewegungsmessung über einen längeren Zeitraum bei nur minimaler bis keiner Sichtbehinderung. Die Bestrahlung des Auges mit Infrarotlicht kann jedoch zu einem schnelleren Austrocknen des Auges führen. [Fi99]

2.2.6 Vorhandene Augenbewegungsmesssysteme am Standort

Die zuletzt vorgestellte Video-Methode eignet sich hervorragend zur Erforschung der Mensch-Maschine-Schnittstelle (MMS). Es ist ebenfalls denkbar, dass zukünftige Fahrzeuge mit Kameras ausgestattet werden, die es erlauben,die Augenbewegungen des Fahrers zu analysieren um Rückschlüsse als Eingangsparameter für Fahrerassistenzsysteme zu gewinnen. Am DLR Standort in Braunschweig stehen dem Institut FS zwei

Augenbewegungsmesssysteme unterschiedlicher Hersteller zur Verfügung.

FaceLAB von Seeing Machines

Das von der Firma Seeing Machines aus Australien entwickelte System faceLAB^{tm} (siehe Abbildung: 2.8) ermöglicht die Berechnung verschiedener Parameter der visuellen menschlichen Wahrnehmung wie Kopfposition, Augenposition oder Blickrichtung.

Es handelt sich um ein passives Video-Messverfahren, welches ohne einschränkende Apparaturen, beispielsweise durch den Einsatz von Helmkameras (verglichen mit SMI-System siehe 2.2.6) auskommt. Dies ermöglicht eine freie Bewegung des Kopfes innerhalb des Erfassungsbereiches der Kameras.

Abbildung 2.8: Das Foto zeigt ein faceLAB^{tm} v4 System
[Quelle: http://www.seeingmachines.com/facelab.htm]

Ermöglicht wird die Berechnung der Parameter nach dem Prinzip des Stereo-Sehens mittels zweier Kameras, die in einem festen Abstand zueinander angeordnet sind und auf den selben Fokusierungspunkt ausgerichtet werden. Der Nachteil dieses Verfahrens ist eine geringe zeitliche Auflösung aufgrund der eingesetzten Kameras mit einer Bildwechselfrequenz von 60 Hz. Zusätzlich ermöglicht das Verfahren nur eine geringe örtliche Auflösung, welche eine Genauigkeit von etwa 5° bietet. Das faceLAB^{tm} System wird häufig in Versuchsfahrzeugen eingesetzt, da mit dem Prinzip des Stereo-Sehen die Verletzungsgefahr der Testperson bei einem möglichen Unfall geringer ist als im Vergleich zu einem helmbasierten System wie dem von SMI (siehe nächsten Abschnitt

2.2.6). Für eine detalierte Beschreibung des faceLAB^{tm} Systems siehe [Schl07].

iViewX HED/HT von SensoMotoric-Instrumens

Das zweite am DLR Standort Braunschweig vorhandene Augenbewegungsmesssystem ist von der Firma SensoMotoric-Instrumens (SMI). Hierbei handelt es sich um ein Headmounted Eyetracking Device (HED) (siehe Abbildung 2.9) mit der Bezeichnung iViewX. Das System besitzt zwei an einem Helm angebrachte Kameras. Die erste Kamera ist in Blickrichtung des Helmträgers ausgerichtet und nimmt das Sichtfeld auf (Szenenkamera). Bei der zweiten Kamera handelt es sich um eine infrarotempfindliche Kamera, welche das linke oder rechte Auge des Trägers über einen ebenfalls am Helm montierten Spiegel aufzeichnet. Beide Kameras besitzen eine Bildwechselfrequenz von 60 Hz. Dies ermöglicht eine Bestimmung der Augenbewegungen anhand von 60 Bildern pro Sekunde bzw. mittels eines Bildes alle 0,016667 Sekunden, was der maximalen zeitlichen Auflösung des SMI-Systems entspricht.

Abbildung 2.9: Foto zeigt das im Labor vorhandene SMI iViewX HED System [Quelle: http://www.smi.de/]

Die Bewegungen des Kopfes werden durch ein magnetisches Bewegungserfassungssystem (iView HT) bestimmt. Mit dem HT-System (HeadTracking) wird über dem Einsatzort des HED ein Magnetfeld erzeugt. Auf dem HED befindet sich eine Messeinheit (Prinzip wie bei der magnetischen Induktionsmethode), welche die Bewegungen des HED in elektische Signale umwandelt. Mit den hieraus ermittelten Daten ist eine rela-

tiv genaue Berechnung der Augenbewegungen möglich, ohne dass die Bewegungsfreiheit des Kopfes eingeschänkt werden muss (Alternative: starre Fixierung des Kopfes an dem Versuchsaufbau).

Der Helm ist über ein Kabel mit einem Computer verbunden. Dieser berechnet anhand einer Analysesoftware aus den Bildern der auf das Auge gerichteten Kamera und den Bewegungsdaten des Kopfes verschiedene Parameter (für eine detailierte Parameterübersicht siehe 2.2.6). Die Berechnung der Parameter erfolgt in Echtzeit durch die Analysesoftware, welche Augenbewegungen von kleiner als 0,1 Grad bestimmen und den aktuellen Blickpunkt in der betrachteten Szene mit einer Genauigkeit von kleiner als 0,5 - 1 Grad berechnen kann [SMI07].

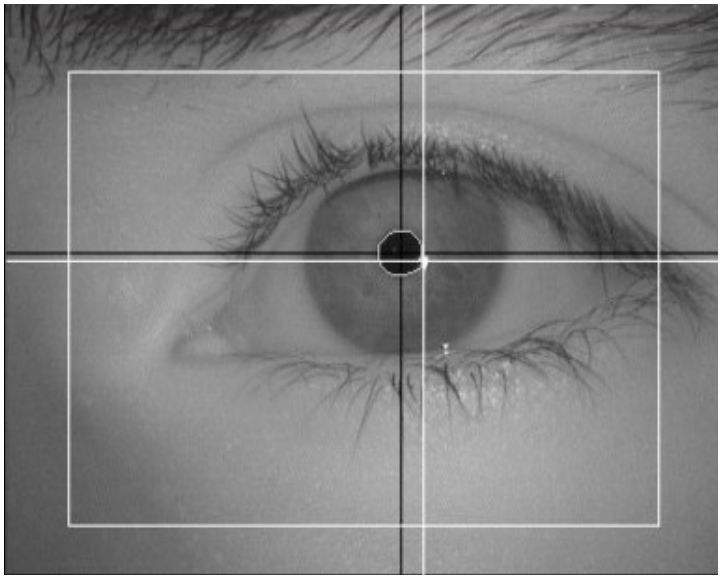


Abbildung 2.10: Foto zeigt ein Bild der Infrarotkamera, in welchem durch die SMI Analysesoftware die Pupillenposition und der corneale Reflex des Auges bestimmt wurde [Quelle: http://www.smi.de/]

Bei der Analyse der Kamerabilder wird die Ausrichtung des Auges über die Pupille und die direkte Spiegelung der Infrarotlampe auf der Hornhaut (cornealer Reflex) als Referenzpunkte bestimmt (siehe Abbildung 2.10). Dies ist nur nach einer zuvor erfolgten Kalibierung des Systems möglich. Bei dieser Kalibierung muss die Testperson verschiedene Punkte im Sichtfeld fixieren und einen Moment lang halten.

Die in Echtzeit berechneten Parameter und Videos werden auf dem Analyserechner grafisch dargestellt und gespeichert. Zusätzliche ist ein Versenden der berechneten Parameter über Ethernet (UDP) oder die serielle Schnittstelle möglich.

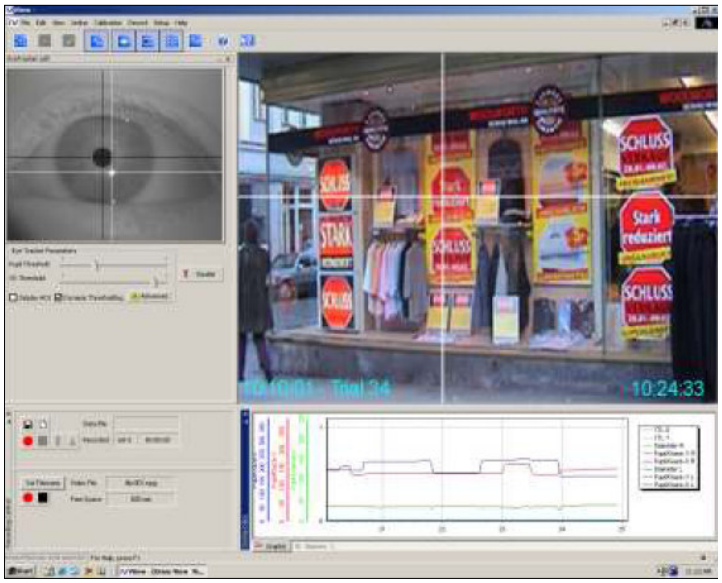


Abbildung 2.11: Das Foto zeigt einen Ausschnitt der SMI Analysesoftwareoberfläche [Quelle: http://www.smi.de/]

Die Abbildung 2.11 zeigt die Benutzeroberfläche der SMI Analysesoftware. In dem Bild befinden sich im oberen linken Bereich die Darstellung des Infrarot-Kamerabildes als Grundlage für die Berechnung der Ausrichtung des Auges. Rechts daneben wird die nach vorne gerichtete Szenenkamera dargestellt. Dort wird anhand des weißen Makierungskreuzes der aktuelle Fixierungspunkt eingeblendet. Im unteren rechten Bereich befindet sich eine Darstellung der berechneten Parameter in Diagrammform. Mit den Bedienelementen im unteren linken Bereich der Oberfläche kann der Viedeodatenstrom der Szenenkamera für eine spätere Analyse oder zur Dokumentation gespeichert werden.

SMI iViewX HED Parameterübersicht

Die Analysesoftware der Firma SMI bestimmt anhand der aufgenommenen Kamerabilder sowie der ermittelten Kopfposition verschiedene Parameter in Echtzeit. Diese Parameter können durch die Analysesoftware sowohl angezeigt als auch gespeichert werden. Zusätzlich ist über verschiedene Schnittstellen ein Streamen der Daten an an-

dere Systeme möglich. Nachfolgend werden die verschiedenen von dem SMI System zur Verfügung gestellten Parameter kurz vorgestellt:

- timestamp:
  Zeitstempel der Daten. Beginn bei Zeitmessung (bei Null) wenn die Analysesoftware von SMI gestartet wird. Maßeinheit: [ms]

- scene counter:
  Nummerierung der mit den Kameras aufgenommenen Bilder in Form einer fortlaufenden Zahl

- viewing plane:
  Zu Beginn eines Versuches werden bestimmte Bereiche (Ebenen) von besonderem Interesse bestimmt. Dies kann zum Beispiel die Windschutzscheibe eines Autos sein. Blickt man während des Versuches in eine dieser Ebenen wird die Nummer dieser Ebene zurückgegeben. Wenn der Blick außerhalb aller definierten Ebenen ruht, wird -1 zurückgegeben.

- pupil diameter X,Y:
  Pupillendurchmesser in X- bzw. Y-Richtung. Maßeinheit: [Pixel]

- pupil position X,Y:
  Pupillenposition in X bzw. Y im Bild der Infrarotkamera. Maßeinheit: [Pixel]

- corneal reflex position X,Y:
  Position der direkten Spiegelung der Infrarotlampe in X bzw. Y auf der Hornhaut im Bild der Infrarotkamera. Maßeinheit: [Pixel]

- gaze position X,Y:
  Stellt die Blickpostion des Auges in X- bzw. Y-Koordinaten im Bild der Sichtfeldkamera dar. Maßeinheit: [Pixel]

- normalized gaze vector X,Y,Z:
  Normalisierte Darstellung der Blickrichtung des Auges als dreidimensionaler Vektor. Einheitslos [+/- 1,000]

- eye position X,Y,Z :

  Beschreibt die X- bzw. Y- und Z-Position des Auges im Verhältnis zum Ursprung des magnetischen Feldes (iViewX HT). Maßeinheit [mm]

- head position X,Y,Z:

  Gibt die Postion des Kopfes im Verhältnis zum Sender des magnetischen Feldes (iView HT) an. Maßeinheit: [mm]

- head rotation (A, E, R):

  Beinhaltet die Rotationsbewegungen des Kopfes in einem Koordinatensystem mit Azimut (A), Elevation (E) und Roll (R) mit Bezug auf den Ursprung des magnetischen Feldes (iView HT). Maßeinheit: [Grad]

Kapitel 3

Softwareentwicklung - Projektmanagement

Im vorangegangenen Kapitel wurden die für die Programmierung notwendigen Grundlagen beschrieben. Unter anderem wurden das Softwareentwicklungsmodell und die in dieser Arbeit verwendeten Module der Applikations- und Entwicklungsplattform SMPL++ vorgestellt. Im letzten Kapitel wurden ebenfalls die im Bereich der Blickrichtungsmesssysteme verwendete Begriffe wie Sakkade oder Fixation erklärt. Zusätzlich wurde ein Überblick über die am DLR Standort Braunschweig vorhandenen Blickrichtungsmesssysteme gegeben.

Dieses Kapitel stellt das in der Planungsphase dieser Masterarbeit definierte Lastenheft vor. Zusätzlich wird in diesem Kapitel der zeitliche Rahmen für die Implementierung der Smpl++-Plattform Klassenbibiliotheken (EyeTracking-Module) durch eine Meilensteinübersicht vorgestellt.

3.1 Lastenheft

Vor Beginn der eigentlichen Umsetzung der vorgestellten Aufgabenstellung (siehe 1.1) wurde das folgende Lastenheft[1] erstellt. Dieses Heft entstand in der Planungsphase der Masterarbeit und beschreibt die durch diese Masterarbeit und dem DLR Institut FS definierten Anforderungen an die zu entwickelnde Smpl++-EyeTracking-Module für die Integration von Blickrichtungsmesssystemen.

Allgemeine Anforderungen an die Implementierung

- Dokumentation der Smpl++-Module mittels UML-Klassendiagrammen
- Doxygen-konforme Kommentierung des Quellcodes

Vorgaben zur Entwicklung der EyeTracking-Module

- **Datenerfassung** zum Einlesen der Messdaten von Blickrichtungsmesssystemen
  - Generalisierte Schnittstelle für verschiedene Blickrichtungsmesssysteme
  - Spezielle Schnittstelle für Aufnahmen vom Blickrichtungsmesssystem von SMI
  - Implementierung eines Prozesses für die Verwaltung der empfangenen Daten
  - Aufnehmen von Blickrichtungsmessdaten über die SMI-RemoteControl-Schnittstelle (Blickwinkel, Gaze-Rotation, Head-Rotation, Plane, Zeitstempel, Gaze-Position)
  - Rohdaten in „`SmplSharedMemory`“ (SSM siehe Abschnitt: 2.1.7) schreiben
- **Rohdateninterpretation** zur Analyse der eingelesenen Blickrichtungsmessdaten
  - Analyse der gängigen Parameter (Sakkaden, Fixationen, Viewing Time, Viewing Duration etc.)
  - Definition von Areas Of Interests (AOIs)

[1] Ein Lastenheft (auch Anforderungsspezifikation oder Requirement Specification) beschreibt die unmittelbaren Anforderungen, Erwartungen und Wünsche an ein geplantes Produkt.

  - Festlegung der mathematischen Methoden und Schwellwerten zur Interpretation der Blickrichtungsmessdaten
  - Implementierung eines Prozesses zur Interpretation der Rohdaten
  - Rohdaten aus SSM lesen
  - Interpretierte Daten in SSM schreiben
  - Testing, verifizieren der interpretierten Daten

- **Visualisierung der Daten** zur besseren Auswertung der interpretierten und gemessenen Blickrichtungsmesssystemdaten
  - Bestehenden „`SmplFSViewer`“(siehe Abschnitt: 2.1.7) ergänzen um Funktionalität zur Darstellung von Rohdaten und interpretierten Daten
  - Weiterhin „`SmplFSViewer`“ um Perspektive auf Versuchssetup „Straightforward Modular Prototyping Laboratory“ (SMPLab) ergänzen
  - Erstellung eines SBA-Take-Files und Implementierung der zugehörigen Daten-Extraktoren zur Visualisierung in Diagrammform mit „`SmplcaSBAro`“ (siehe Abschnitt: 2.1.7)

- **Beispielhafte Online-Applikation** (optional bei genügend verbleibender Restzeit innerhalb der Masterarbeit)
  - Mögliche Aufgabenstellung: ”Wenn Blick nicht nach vorn, dann Vibration auf den Sidestick geben”
  - Auswertung und Analyse der Ergebnisse der Online-Applikation

3.2 Meilensteinübersicht

Anhand des beschriebenen Lastenheftes sowie dem für die Umsetzung zur Verfügung stehenden Zeitrahmens wurde die folgende Meilensteinübersicht (siehe Abbildung 3.1) in der Planungsphase entworfen. Diese Übersicht beschreibt die geplanten Meilensteine sowie die für die einzelnen Arbeitsabschnitte vorgesehenen Entwicklungszeiten innerhalb dieser Masterarbeit.

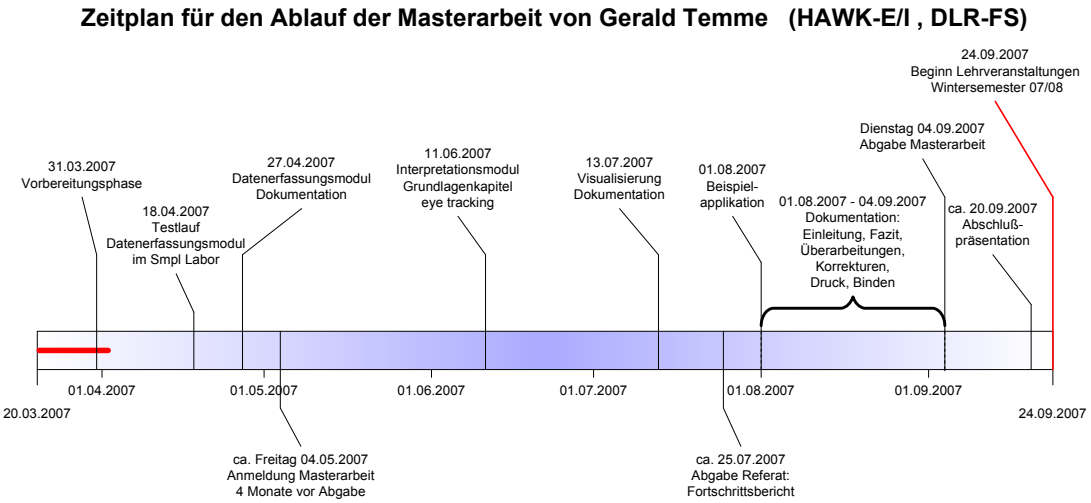


Abbildung 3.1: Zeitplan zur Umsetzung der Masterarbeit, Meilensteinübersicht

Kapitel 4

Softwareentwicklung - Entwurf

Das vorherige Kapitel stellte das in der Planungsphase entwickelte Lastenheft und Meilensteine in Form eines Zeitplans zur Umsetzung dieser Masterarbeit vor.

Dieses Kapitel stellt die ersten Entwürfe, welche vorbereitend für die Implementierung der EyeTracking-Klassen und Module in die Smpl++-Umgebung während der Planungsphase entwicklet wurden, vor. Durch einen Vergleich dieser ersten Entwürfe der Module mit den Diagrammen der finalen Implementierung (siehe Kapitel 5) läßt sich ein Eindruck von dem Mehraufwand der zusätzlichen Funktionalitätenanforderungen an die Module gewinnen, welche erst zur Umsetzung der Module bekannt geworden sind.

4.1 Datenerfassungsmodul

Die Folgende Abbildung 4.1 zeigt den Entwurf des Datenerfassungsmodules vor der Implementierung.

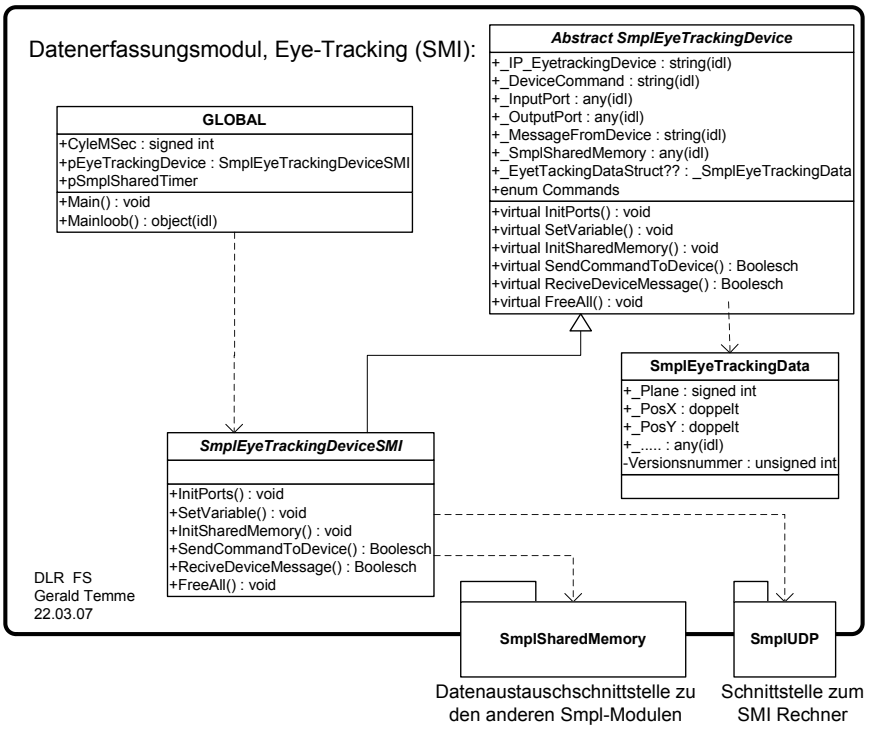


Abbildung 4.1: Entwurf des Datenerfassungsmoduls mit spezieller Klasse zur Integration des Blickrichtungsmesssystems von SMI (diese Abbildung wurde als eine an ein UML Klassendiamm angelehnte Skizze entworfen)

Das Datenerfassungsmodul besteht im ersten Entwurf aus einer Abstracten Basisklasse „`SmplEyeTrackingDevice`“, die den speziellen Blickrichtungsmesssystemklassen wie „`SmplEyeTrackingDeviceSMI`“ Funktionalitäten in Form von virtuellen Methoden vordefiniert. Die Datenklasse „`SmplEyeTrackingData`“ ermöglicht zusammen mit dem Modul SSM die Weitergabe der über UDP erhaltenen Blickrichtungsmesssystemparameter an die Module der Smpl++-Platform.

4.2 Interpretationsmodul

Die Abbildung 4.2 beschreibt den ersten Entwurf des Interpretationsmodules.

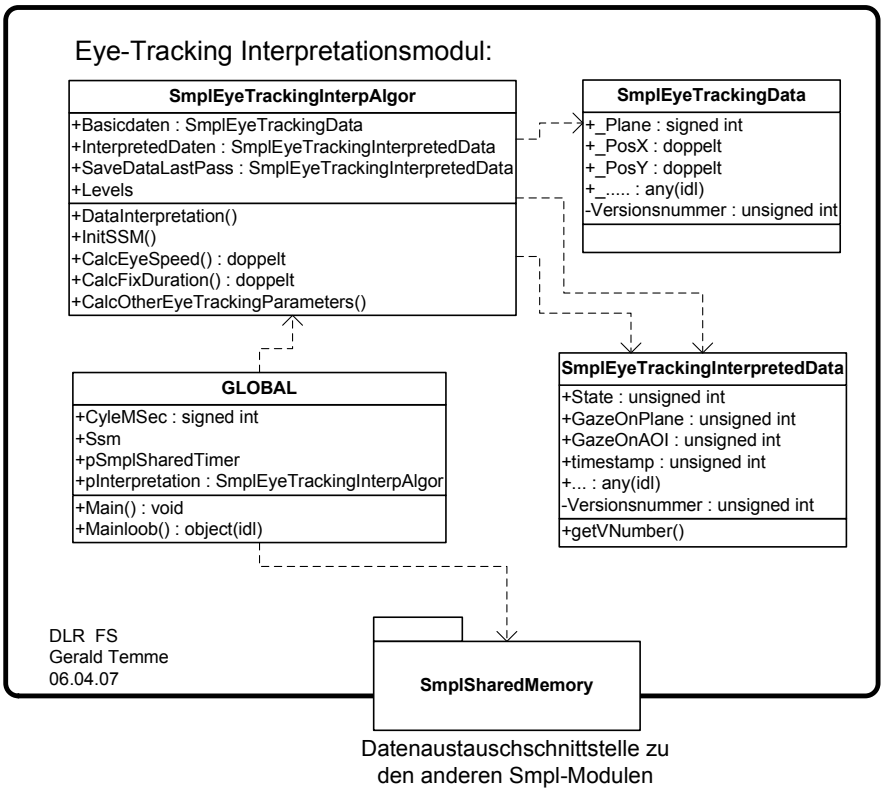


Abbildung 4.2: Entwurf des Interpretationsmoduls zur Bestimmung von Fixationen und Sakkaden anhand der durch das Blickrichtungsmesssystem zur Verfügung gestelten Parameter (diese Abbildung wurde als eine an ein UML Klassendiamm angelehnte Skizze entworfen)

Der Entwurf zeigt eine Interpretationsklasse „`SmplEyeTrackingInterpAlgor`“, welche für die Parameter der Blickrichtungsmesssysteme „`SmplEyeTrackingData`“ bestimmt, ob das Auge ein Objekt fixiert oder ob es eine Sakkade ausführt. Das Ergebnis dieser Berechnung sowie einige Überträge aus den Parametern der Rohdaten werden in der Datenklasse „`SmplEyeTrackingInterpretedData`“ gespeichert und über das Modul SSM allen Smpl++-Modulen zur Verfügung gestellt.

Dieser Entwurf wurde während der Implementierung des Moduls stark erweitert/überarbeitet, da sich herrausstellte, dass die durch das Blickrichtungsmesssystem bestimmten Parameter nicht allen Anforderungen des DLR zur Analyse der Augenbewegungen entsprechen (eine Visualisierung von Blickpunkten ausserhalb der Planes ist durch das Blickrichtungsmesssystem von SMI nicht vorgesehen).

4.3 Datenvisualisierung

Der erste Entwurf der Datenvisualiserung sah zweidimensionale Bildoverlays zur Darstellung von Fixationen und Sakkaden vor (siehe Abbildung 4.3).

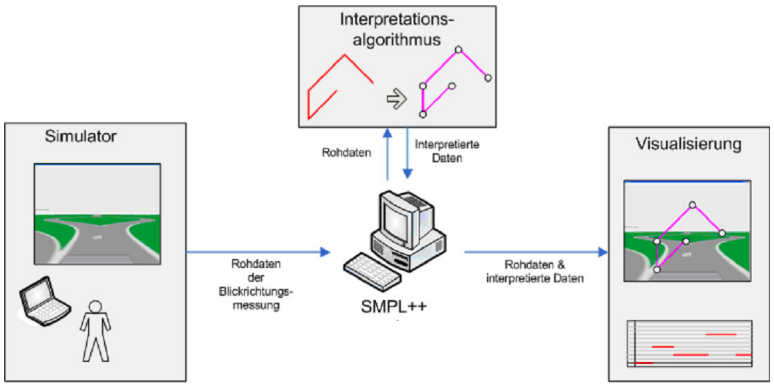


Abbildung 4.3: Skizze zur Visualisierung der interpretierten Daten aus dem Expose' zu dieser Masterarbeit [Quelle: DLR-FS]

Diese Datenvisualisierung wurde während der Masterarbeit um die dritte Dimension erweitert (siehe Kapitel 5.3). Die folgende Abbildung 4.4 zeigt eine erste Skizze der Visualisierung des SMPLabs als 3D-Modell.

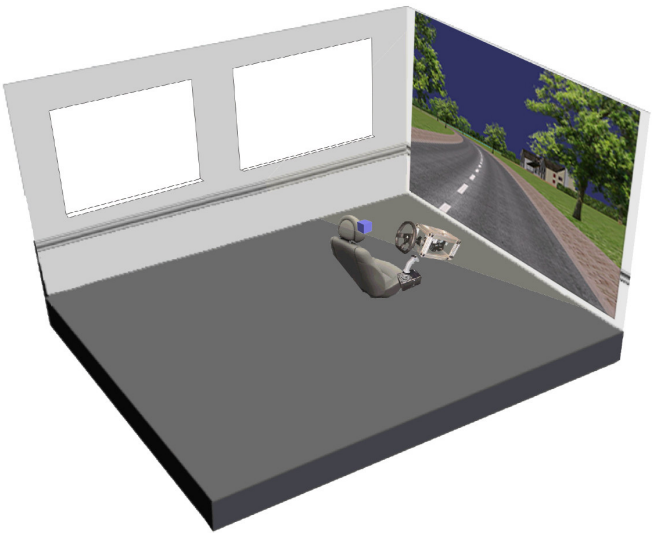


Abbildung 4.4: Skizze des SMPLabs als 3D-Modell

Nach der Modellierung eines 3D-Modells zur Darstellung des Versuchsaufbaus wurden die Darstellungsformen für Fixationen und Sakkaden entworfen. Der Entwurf der Darstellungsformen orientiert sich im wesentlichen an dem Dokument [Fle00] (siehe z.B. Abbildung 5.13) . Die folgende Abbildung 4.5 zeigt einen Entwurf der Darstellung von Fixationen und Sakkaden mittels eines Whitebords im SMPLab.

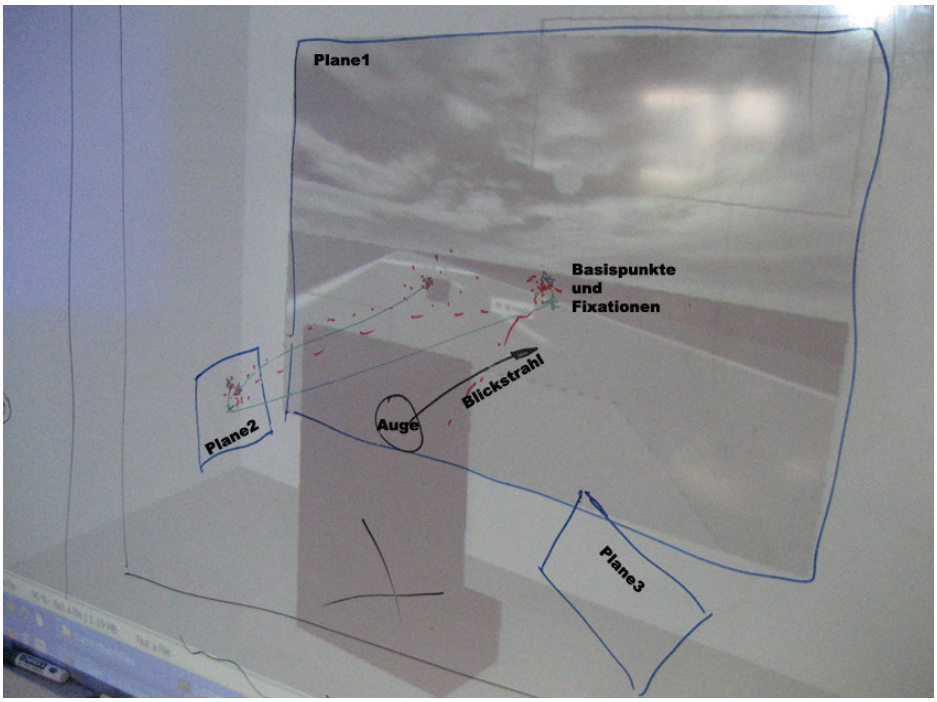


Abbildung 4.5: Foto zum Entwurf der Visualisierung von Fixationen und Sakkaden

Kapitel 5

Softwareentwicklung - Implementierung & Dokumentation

Im letzten Kapitel wurden die Entwürfe zur Umsetzung der folgenden Implementierungen der EyeTracking-Module vorgestellt.

Dieses Kapitel beschreibt und dokumentiert die innerhalb der Masterarbeit entstandenen Klassen, welche nach Abschluss dieser Arbeit als Module für die Smpl++-Umgebung zur Verfügung stehen. Zusammengenommen bilden diese Module eine Verarbeitungskette, beginnend bei der Rohdatenerfassung[1] des verwendeten Blickrichtungsmesssystems mittels einer Ethernetschnittstelle, über die Interpretation der erfassten Rohdaten in annähernd Echtzeit bis hin zur Visualisierung in Form von Diagrammen und einem 3D-Modell des Versuchsaufbaus, sowohl in annährend Echtzeit als auch in Form von Replays.

[1] Die Dokumentation der Rohdatenerfassung beschreibt die generische für alle Blickrichtungsmesssysteme entwickelte Basisklasse, sowie die innerhalb dieser Masterarbeit umgesetzte spezielle Klasse für das Blickrichtungsmesssystem von SMI.

5.1 Eye Tracking Datenerfassungsmodul

Die allgemeine Aufgabe des Datenerfassungsmodules ist die Kommunikation mit Blickrichtungsmesssystemen wie faceLAB von Seeing Machines oder IViewX von SMI bzw. die Steuerung der Systeme. Die Kommunikation zwischen dem in die Smpl++-Umgebung integrierten Datenerfassungsmodul und dem angeschlossenen Blickrichtungsmesssystem erfolgt über das Ethernet mittels des UDP.

Das Modul ermöglicht hierbei das Senden einfacher „remote commands“ wie START, STOP und PAUSE. Daneben nimmt er eingehende Datenpakete in Empfang. Diese Datenpakete enthalten die gemessenen Rohdaten des Blickrichtungsmesssystems. Da sich die übertragenen Daten in ihren Bezeichnungen, Maximal- und Minimalwerten je nach eingesetztem Blickrichtungsmesssystem unterscheiden können, müssen sie zunächst generalisiert werden. Danach stehen sie über das „`SmplSharedMemory`“ (SSM) allen anderen Smpl++-Modulen zur Verfügung. Dieses Modul ermöglicht ebenfalls ein Starten bzw. Stoppen der Viedeoaufzeichnung des Versuches über das Online-Analyse-, Aufzeichnungs- und Replay-Tool „`SmplcaSBAro`“ der Smpl++-Umgebung (siehe Kapitel 2.1.7).

5.1.1 UML-Klassendarstellung des Modules

Da neben den Rohdaten, auch die Form der „remote commands“ abhängig von dem eingesetzten Blickrichtungsmesssystem ist, ist es nicht möglich, alle Systeme mittels einer Datenerfassungsklasse zu betreiben. Das folgende UML Klassendiagramm (siehe Abbildung 5.1) zeigt die Umsetzung des Datenerfassungsmodules „`SmplEyeTrackingRemoteControl`“.

Eine Schnittstellenklasse „`SmplEyeTrackingDevice`“ stellt für das Datenerfassungsmodul Funktionalitäten, die für möglichst viele Blickrichtungsmesssysteme gültig sind, zur Verfügung. Dazu gehört die Vorbereitung der Kommunikation des Datenerfassungsmo-

dules mit dem Blickrichtungsmessystem. Diese erfolgt über UDP unter Zuhilfenahme des Smpl++-Modules „`SmplEthernet`“ und im Speziellen der Klasse „`SmplUDP`“ dieses Modules. Diese UDP Klasse stellt die notwendigen Hilfsmittel zum Erstellen, Verwalten und Empfangen von UDP Nachrichten über das Ethernet zur Verfügung. Ausserdem sorgt die Klasse „`SmplEyeTrackingDevice`“ für die Initialisierung des „SSM“ und bereitet den speziellen Datentyp „`SmplEyeTrackingData`“ zur späteren Verwendung vor.

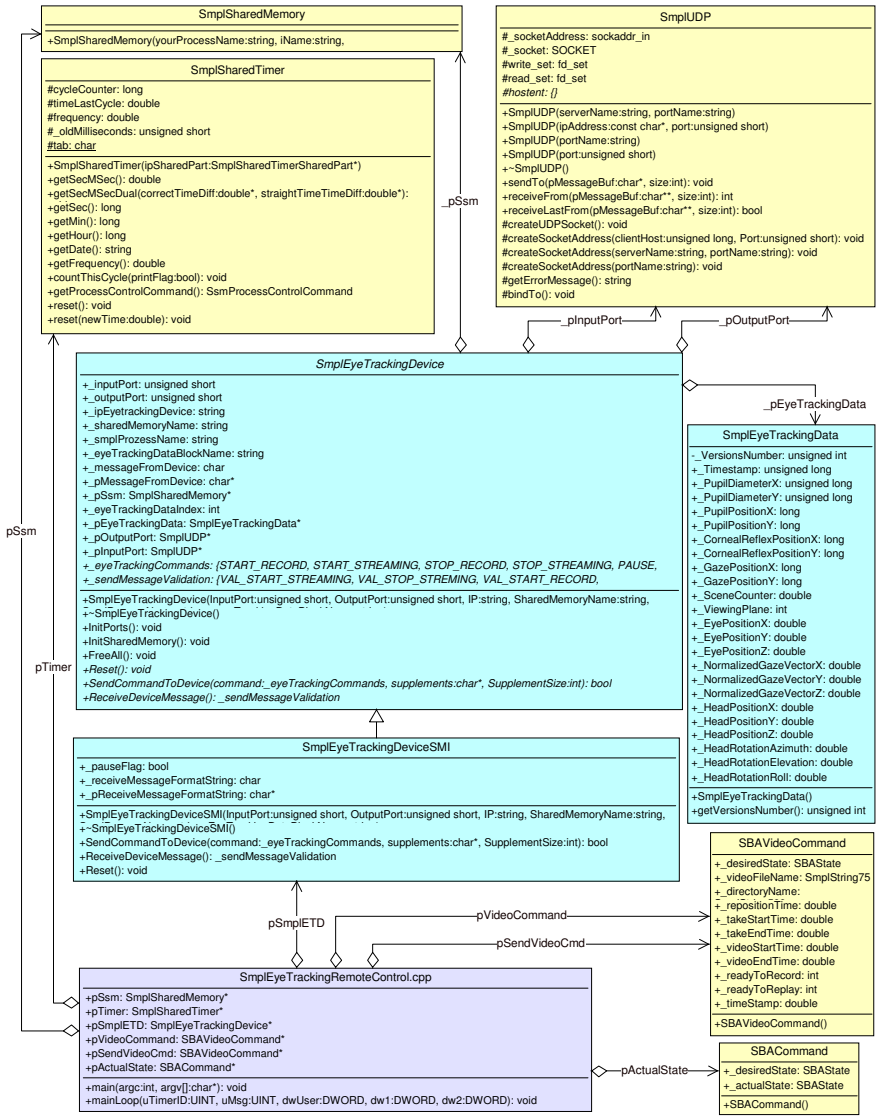


Abbildung 5.1: UML-Klassendiagramm des Datenerfassungsmodules „SmplEye-TrackingRemoteControl“, siehe auch im Anhang 8.1, Seite 100

Die spezielle Schnittstellenklasse „`SmplEyeTrackingDeviceSMI`“ erbt die Funktionalität der Klasse „`SmplEyeTrackingDevice`“ und stellt dem Datenerfassungsmodul speziell an das SMI Blickrichtungsmesssystem angepasste Funktionen zur Verfügung. Hierunter fallen das Senden von Kommandos an das SMI System und das Entgegennehmen von Daten über UDP-Verbindungen. Diese Daten werden direkt nach ihrer Ankunft durch die Funktion „`ReceiveDeviceMessage()`“ der Klasse „`SmplEyeTrackingDeviceSMI`“ in das generische Datenformat „`SmplEyetrackingData`“ umstrukturiert und danach über das SSM allen anderen SMPL++-Modulen zur Verfügung gestellt.

Mittels der bereits in der Smpl++-Umgebung vorhandenen Klassen „`SBAVideoCommand`“ und „`SBACommand`“ wird die Steuerung zum Aufzeichnen von Videos auf dem SMI Blickrichtunggsmesssystemrechner über das Modul „`SmplcaSBAro`“ ermöglicht.

5.1.2 Argumentenübergabe beim Start des Modules

Gestartet wird dieses Modul mittels der Main-Funktion, welche sich in der Datei „`SmplEyeTrackingRemoteControl.cpp`“ befindet. Die Funktionen und Variablen dieser Datei sind Global. Gestartet wird dieses Modul üblicherweise aus einer Stapelverarbeitungsdatei heraus, wobei zusätzliche Argumente wie das zu verwendende Blickrichtungsmesssystem mitübergeben werden können (siehe Listing: 5.1).

```
1:  start ./Programs/SmplEyeTrackingRemoteControl/
          SmplEyeTrackingRemoteControl.exe
2:          CycleMSec       10
3:          EyeTrackingDevice     SMI
4:          EyeTrackingDeviceIP     129.247.53.237
5:          InputPort       4444
6:          OutputPort      4444
7:          SsmName       ssm1
8:          EyeTrackingDataBlockName  ETData
9:          PATH        C:\\Temp\\
10:         SBAVideoCommand     ReceiveVideoCommand
11:                             SendVideoCommand
12:         VideoExtension      .mpeg
```

Listing 5.1: Starten des Datenverarbeitungsmodules mit Argumentenübergabe

Beim Start des Datenverarbeitungsmodules können folgende Argumente übergeben werden. Wird ein Argument nicht übergeben verwendet das Modul einen festeingestellten Standartwert (SW):

- CycleMSec:
  Gibt den Abstand zwischen den Ausführungen des Callback-Teils in Millisekunden an. SW: 7

- EyeTrackingDevice:
  Gibt an, welches Blickrichtungsmesssystem verwendet werden soll. Im Rahmen der Masterarbeit wurde bisher nur die spezielle Klasse für das SMI System entwickelt, sodass hier als Argument nur der Wert „SMI“ übergeben werden kann. SW: SMI

- EyeTrackingDeviceIP:
  Mit diesem Argument wird dem Modul die IP-Adresse des Blickrichtungsmesssystemrechners im Ethernet übergeben. SW: 192.168.2.42

- InputPort:
  Übergibt den zu verwendenden Port für das Empfangen von Datenpaketen vom Blickrichtungsmesssystemrechner über UDP an. SW: 4444

- OutputPort:
  Beschreibt den zu verwendenden Port beim Senden von „remote commands“ an den Blickrichtungsmesssystemrechner über UDP. SW: 4444

- SsmName:
  Stellt dem Modul den Namen des zu verwendenen „SSM“ zur Verfügung. SW: SmplEyeTrackingRemoteControl

- EyeTrackingDataBlockName:
  Gibt den Namen des Datenblocks an, unter welchem die vom Blickrichtungsmesssystemrechner erhaltenen Daten im „SSM“ abgelegt werden sollen. SW: SmplEyeTrackingDeviceData

- PATH:
  Enthält den lokalen Verzeichnisnamen zum Speichern der Videoaufzeichnung des Versuchs auf dem SMI Rechner. SW: C://temp//EyeTrackingVideo.mpeg

- SBAVideoCommand:
  Mit diesem Parameter werden die Datenblocknamen des „`ReceiveVideoCommand`“ und des „`SendVideoCommand`“ Datenblocks zur Videoaufzeichnungssteuerung über „`SmplcaSBAro`“ im „SSM“ bekanntgegeben. Standartwerte: ReceiveVideoCommand, SendVideoCommand

- VideoExtension:
  Gibt das zu verwendene Aufzeichnungsformat der Videos auf dem Blickrichtungsmesssystem an (z.B. „.mpeg“). SW: .mpg

5.1.3 Kommunikationsablauf des Modules

Die nachfolgende Grafik (siehe Abbildung: 5.2) stellt den implemetierten Kommunikationsablauf zwischen dem Smpl++- und dem SMI-Rechner dar. Damit diese Kommunikation funktionieren kann, muss zu Beginn die Blickrichtungsmesssoftware auf dem SMI-Rechner aktiviert und auf Remoteberieb gestellt werden. [SMI07].

Mit dem Starten des Datenerfassungsodules in Smpl++ wird als erstes das Kommando START_STREAMING ausgeführt. Dieses beinhaltet, dass ein Ausgabeformat gewählt und an das SMI-System übermittelt wird. Mit dem Setzen des Ausgabeformates wird zeitgleich auch das Datenstreaming des SMI Systems gestartet.

Ab diesem Zeitpunkt versendet das SMI-System mit 60Hz Datenpakete in der Form des festgelegten Ausgabeformates. Diese werden von dem Datenerfassungsmodul, welches mindestens mit der gleichen Frequenz betrieben werden muss um echtzeitfähig zu sein, entgegengenommen. Innerhalb der Masterarbeit wird das Datenerfassungsmodul mit 120Hz betrieben, was der doppelten Abtastfrequenz der UDP Verbindung entspricht. Abschließend werden die erhaltenen Daten an das „SSM“ weitergeleitet. Ab diesem Zeitpunkt ist eine Interpretation der Rohdaten möglich.

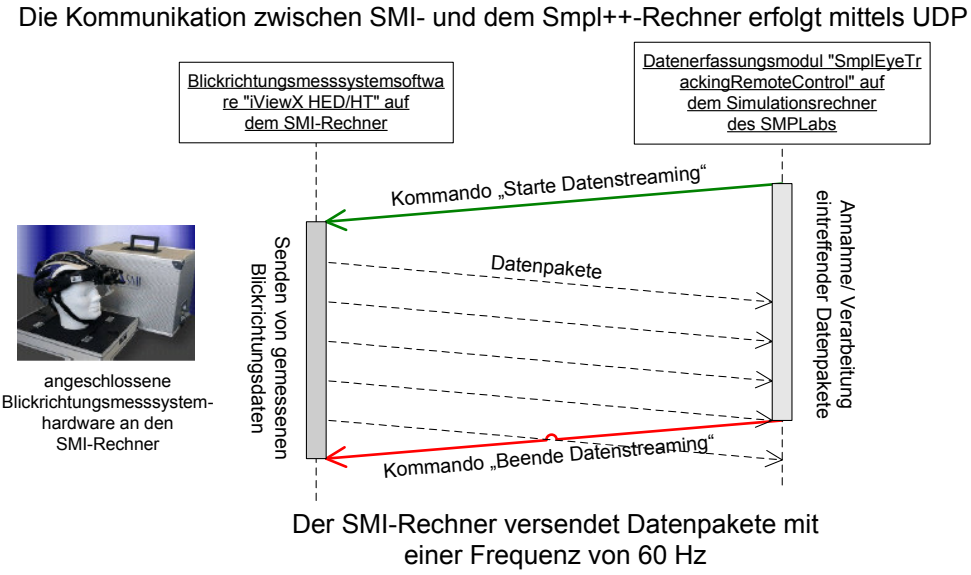


Abbildung 5.2: Prinzipieller Kommunikationsablaufes des Datenerfassungsmodules

Mittels des Online-Analyse-, Aufzeichnungs- und Replay-Tool „`SmplcaSBAro`“ können die in den „SSM“ geschriebenen Rohdaten für eine spätere Analyse oder zur Dokumentation gespeichert werden. Ein Aufzeichnen des Szenenvideos auf dem SMI-Rechner kann mittels „`SmplcaSBAro`“ zusätzlich zum Aufzeichnen der „SSM“ Daten gestartet werden. Das Starten der Videoaufzeichnung wird dem Datenerfassungsmodul mittels des „`ReceiveVideoCommand`“ Datenblocks bekanntgegeben. Das Modul sendet dann den Befehl START_RECORD an das SMI-System. Von dem SMI-System wird nach dem Starten der Videoaufzeichnung eine Bestätigung ET_REC an das Datenerfassungsmodul zurück gesendet. Für eine synchronisierte Wiedergabe der gespeicherten Rohdatensätze mit dem aufgezeichneten Video muss der Startzeitpunkt der Videoaufzeichnung möglichst genau bestimmt und mittels des Datenblocks „`SendVideoCommand`“ „`SmplcaSBAro`“ bekanntgegeben werden. Aufgrund der unbekannten Laufzeiten der Nachrichten zwischen dem SMI- und dem Smpl++-Rechner wird angenommen, dass der eigentliche Startzeitpunkt der Videoaufzeichnung ungefähr in der Mitte zwischen dem bekannten Zeitpunkt des Abschickens der Nachricht START_RECORD und dem Empfang der ET_REC Bestätigung liegt:

$$VideoStartTime = SendMessageTime + \left(\frac{SendMessageDeviceAnswerTime - SendMessageTime}{2.0}\right)$$

Um eine Videoaufzeichnung erfolgreich abzuschließen, ist vor der Beendigung des Datenerfassungmodules ein Stoppen der Aufzeichnung durch das Modul „`SmplcaSBAro`“ erforderlich. Zum Stoppen der Aufzeichnung sendet „`SmplcaSBAro`“ über den SSM „`SBAVideoCommand`“ das Kommando STOP_RECORD. Das selbe Kommando wird von dem Datenerfassungsmodul zum Beenden der Aufzeichnung an den SMI-Rechner gesendet. Wird das Datenerfassungsmodul ohne eine vorheriges Stoppen der Videoaufzeichnung beendet, kann die für „`SmplcaSBAro`“ notwendige Stoppzeit der Videoaufzeichnung nicht bestimmt werden. Diese Zeit wird wie die Startzeit interpoliert und ist ebenfalls nur der geschätzte Stoppzeitpunkt der Aufzeichnung auf dem SMI Rechner.

Bei Beendigung des Datenerfassungsmodules wird zuletzt eine STOP_STREAMING und STOP_RECORD Nachricht an das SMI System gesendet, dieses befindet sich ab diesem Zeitpunkt wieder in dem gleichen Zustand, wie vor dem Starten des Datenerfassungsmoduls.

5.2 Interpretationsmodul der Eyetracking-Rohdaten

Das Interpretationsmodul ermittelt in annähernd Echtzeit anhand zweier aufeinanderfolgender Blickrichtungsmesssystemrohdatensätze den aktuellen Typ (siehe Abschnitt 5.2.2) des Blickstrahls. Das Modul bestimmt nach dem Typ den Schnittpunkt des Blickstrahls mit definierten Planes und AOIs oder mit einer virtuellen Kugel, welche die Testperson umgibt.

Aus dem Typ und dem bestimmten Schnittpunkt des Blickstrahls lassen sich verschiedene Analyseparameter (siehe Abschnitt: 5.2.4) berechnen. Zur Visualisierung (siehe Abschnitt: 5.3) der interpretierten Daten werden diese über das SSM allen anderen Modulen der Smpl++-Entwicklungsumgebung zur Verfügung gestellt.

5.2.1 UML-Klassendarstellung des Interpetationsmodules

Das nachfolgende UML-Klassendiagramm (siehe Abbildung: 5.3 und vergrößert im Anhang 8.2, Seite 101) zeigt die Umsetzung des Interpretationsmodules „SmplEyeTrackingInterpretation“.

Die Interpretationsklasse „`SmplEyeTrackingDataInterpretation`“ beinhaltet alle zur Interpretation der Rohdaten benötigten Funktionalitäten. Zur Interpretation der Rohdaten muss durch die „Main“-Funktion nach der Instanzierung und Initialisierung der Interpretationsklasse nur noch die Funktion „`InterpretEyeTrackingData()`“ zyklisch aufgerufen werden. Die Ergebnisse der Interpretation werden von der Interpretationsklasse in der Datenklasse „`SmplEyeTrackingGlobals/SmplEyeTrackingInterpretedData`“ gespeichert, welche am Ende jedes Interpretationszyklusses in das SSM geschieben wird.

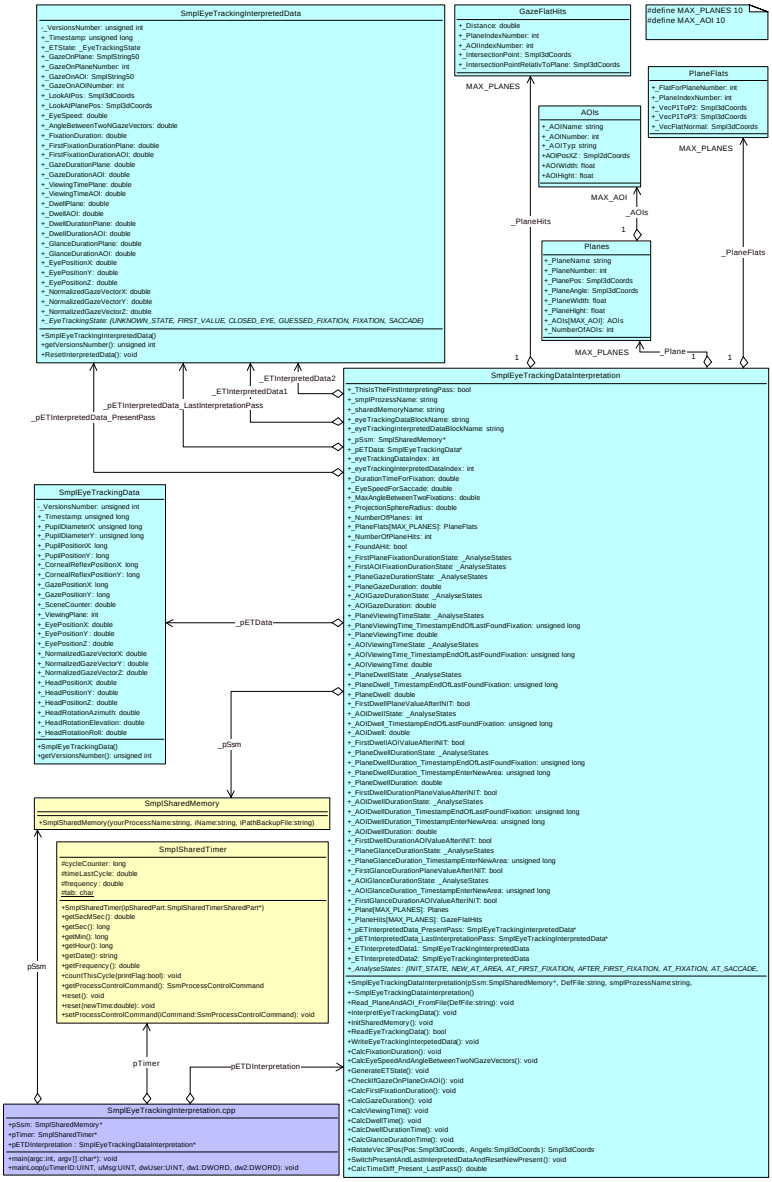


Abbildung 5.3: UML-Klassendiagramm des Interpretationsmodules „SmplEyeTrackingDataInterpretation“ (vergrößert im Anhang 8.2, Seite 101)

5.2.2 Blicktypbestimmung

Die Basis für fast alle Analyseparameter zur Interpretation der aufgezeichneten Blickrichtung ist eine Typunterscheidung des Blickstrahls, die beschreibt, ob eine Fixation vorliegt, eine Sakkade ausgeführt wird oder ob das Auge geschlossen ist.

Die Bestimmung, ob der Blickstrahl vom Typ Fixation oder Sakkade ist, erfolgt in der Regel über ein Schwellwertverfahren. In der Literatur werden je nach Autor einer bis drei der folgenden Schwellwertparameter benutzt:

- örtlich:
  dieser Schwellwert gibt an, um wieviel Grad der aktuelle Blickpunkt von der gefundenen Fixation abweichen darf, um noch als Fixation zu gelten

- zeitlich:
  dieser Schwellwert gibt die minimale zeitliche Dauer einer Fixation in Millisekunden an

- Geschwindigkeit:
  dieser Schwellwert beschreibt die minimale Geschwindigkeit einer Sakkade in Grad pro Sekunde zwischen zwei benachbarten Blickpunkten

- Beschleunigung:
  dieser Schwellwert definiert die minimale Beschleunigung einer Sakkade in $°/s^2$ zwischen zwei benachbarten Blickpunkten

In der Dissertation von Rötting M. [Röt01] befindet sich auf Seite 71 eine Übersicht über die verwendenden Schwellwertparameterkombinationen zur Bestimmung einer Fixation unterschiedlicher Autoren. So verwendeten zum Beispiel Crundall & Underwood (1998) [CU98] als Kriterium für eine Fixation eine Kombination aus den Schwellwerten örtlich ($< 2°$), zeitlich ($\geq$ 100ms) und Geschwindigkeit ($< 20°/s$).

Die Bestimmung des Blicktyps erfolgt in der Interpretationsklasse durch den Aufruf der Funktion „GenerateETState()“ mit der gleichen Schwellwertparameterkombination wie bei Crundall & Underwood. Gespeichert wird das Ergebins der Typbestimmung in der Variable „`_EyeTrackingState _ETState`“ der Datenklasse „`SmplEyeTrackingGlobals/SmplEyeTrackingInterpretedData`“. Die Variable „`_EyeTrackingState`“ ist ein „`enum`“, welches die folgenden Blicktypen vorsieht:

- UNKNOWN_STATE:
  nach der Initialisierung der Variable

- FIRST_VALUE:
  erst ab dem zweiten Datenpaket ist eine Typbestimmung möglich

- CLOSED_EYE:
  das Auge ist geschlossen; keine Blickrichtungsdaten verfügbar

- GUESSED_FIXATION:
  bei Unterschreitung des örtlichen Schwellwerts sowie des Schwellwerts für die Geschwindigkeit besteht eine vermutete Fixation

- FIXATION:
  wird zusätzlich der zeitliche Schwellwert überschritten, liegt eine Fixation vor

- SACCADE:
  bei Überschreitung des örtlichen oder des Geschwindigkeitsschwellwertes liegt eine Sakkade vor

Zur Berechnung der Blickgeschwindigkeit [°/s] sowie des Winkels [°] zwischen zwei benachbarten normalisierten Blickrichtungsvektoren werden die folgenden Formeln verwendet 5.1, 5.2, 5.3 aus [Bron00]:

$$\alpha[Rad] = \arccos \frac{A_x * B_x + A_y * B_y + A_z * B_z}{\sqrt{A_x^2 + A_y^2 + A_z^2} * \sqrt{B_x^2 + B_y^2 + B_z^2}} \tag{5.1}$$

$$Winkel[Grad] = \alpha[Grad] = \frac{\alpha[Rad]}{\pi * 180} \tag{5.2}$$

$$Blickgeschwindigkeit[Grad/s] = \frac{\alpha[Grad]}{dif[ms]/1000} \tag{5.3}$$

dif[ms] ist die Differenz zwischen den beiden Aufzeichnungszeitpunkten der Vektoren durch das Blickrichtungsmesssystem (die Zeitstempel des Systems sind in Millisekunden gegeben).

5.2.3 Trefferermittlung des Blickstrahls mit definierten Bereichen im dreidimensionalen Raum

Nach der Ermittlung des Typs des Blickstrahls wird für die Typen GUESSED_FIXATION, FIXATION und SACCADE eine Trefferermittlung des Blickstrahls mit zuvor definierten Bereichen durchgeführt. Das aus dieser Trefferermittlung resultierende Ergebnis stellt die zweite Berechnungsgrundlage für die meisten Analyseparameter aus der Literatur dar.

Die definierten Bereiche können frei gewählt werden, sind aber in der Regel durch den Versuchsaufbau vorgegeben. So ist z.B. in einem Fahrzeug einer dieser Bereiche eine Fläche (Plane) mit den Abmaßen der Frontscheibe. Auf diesen Planes können für eine genauere Untersuchung Bereiche von besonderem Interesse (AOIs) definiert werden. Ein Beispiel hierfür ist die Definition des Innenspiegels als AOI innerhalb der Plane der Frontscheibe.

Initialisierung von Planes und AOIs

Bevor das Interpretationsmodul die Treffer mit einer Plane oder AOI ermitteln kann, müssen diese Bereiche dem Interpretationsmodul übergeben werden. Dies geschieht mittels einer Definitionsdatei. Detaillierte Informationen zum Aufbau der Definitionsdatei befindet sich im Abschnitt 5.3.3. Durch die Funktion

„`Read_PlaneAndAOI_FromFile( string DefFile)`“ wird die beim Start des Interpretationsmodules angegebene Definitionsdatei ausgelesen und in dem „Array“ „`Planes _Plane[MAX_PLANES]`“ gespeichert `(MAX\_PLANES = 10)`. Die Struktur „Planes“ verfügt wiederum über ein „Array“ „`AOIs _AOIs[MAX_AOI]`“, in welchem die zur Plane gehörenden AOIs gespeichert werden `(MAX\_AOI = 10)`.

Vorbereitend für die spätere Trefferermittlung werden nach dem Einlesen der Definitionsdatei unbegrenzte Ebenen und deren Normale, welche relativ zum Koordinatenursprung des Blickrichtungsmesssystems mit der Ausrichtung der definierten Planes berechnet werden. Diese Ebenen und deren Normalen werden in einem „Array“ „`_PlaneFlats[MAX_PLANES]`“ des „structs“ „`PlaneFlats`“ gespeichert. Die Berechnung der Normalen der Ebenen erfolgt nach den Formeln für die Punkt-Richtungs-Form in der Parameterdarstellung [Pap01a] (siehe Abbildung 5.4 und Formel 5.4).

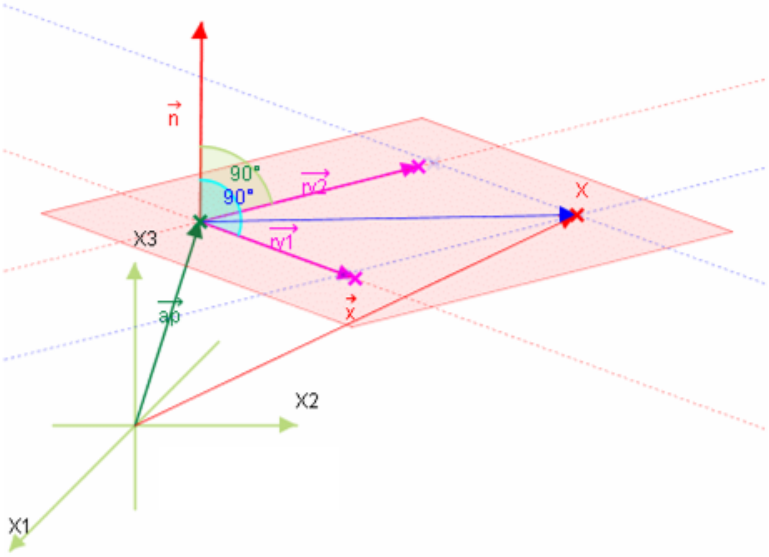


Abbildung 5.4: Darstellung einer Ebene und deren Normale $\vec{n}$
[Quelle: http://mathenexus.zum.de/pdf/geometrie/ebenen/]

$$\vec{n} = r\vec{y}1 \times r\vec{y}2 \tag{5.4}$$

Zur Ausrichtung der Ebene werden ihre Richtungsvektoren $r\vec{y}1$, $r\vec{y}2$ entsprechend der Lage der Planes in kartesischen Koordinaten rotiert. Dies erfolgt durch die Funktion „`RotateVec3Pos( Smpl3dCoords Pos, Smpl3dCoords Angels)`“. Die Rotation des Vektors wird innerhalb dieser Funktion mittels dreier Rotationsmatritzen realisiert (siehe Listing 5.2):

```
1:  double RotateMatrixX[3][3] = {
2:             { 1.0,  0.0,             0.0             },
3:             { 0.0,  cos(Angle._x),  -sin(Angle._x) },
4:             { 0.0,  sin(Angle._x),  cos(Angle._x)  }};
5:
6:  double RotateMatrixY[3][3] = {
7:             { cos(Angle._y),  0.0,  sin(Angle._y) },
8:             { 0.0,            1.0,  0.0           },
9:             { -sin(Angle._y), 0.0,  cos(Angle._y) }};
10:
11: double RotateMatrixZ[3][3] = {
12:            { cos(Angle._z),  -sin(Angle._z), 0.0 },
13:            { sin(Angle._z),  cos(Angle._z),  0.0 },
14:            { 0.0,            0.0,            1.0 }};
```

Listing 5.2: Rotationsmatritzen zur Rotation eines Punktes im kartesischen Koordinatensystem an der X,Y,Z Achse

Trefferbestimmung mit Planes und AOIs

Ob ein Treffer des Blickstrahls mit einer Plane und innerhalb dieser mit einer AOI vorliegt wird durch die Funktion „`CheckIfGazeOnPlaneOrAOI()`“ bestimmt. Eine detaillierte Beschreibung dieser Funktion befindet sich im Anhang (siehe: Standardflussdiagramm der Funktion „`CheckIfGazeOnPlaneOrAOI()`“ im Anhang: 8.3, Seite 102).

Ein Schnittpunkt des Blickstrahls mit der mathematischen Ebene einer Plane existriert, wenn der Normalenvektor der Ebene $\vec{n}$ nicht senkrecht auf dem normalisierten Blickrichtungsvektor $\vec{a}$ steht. Diese Bedingung wird mit der folgenden Formel 5.5 aus [Pap01b] überprüft:

$$\vec{n} \cdot \vec{a} \neq 0 \tag{5.5}$$

Der Schnittpunkt des Blickstrahls mit der mathematischen Ebene der Plane wird mittels der folgenden Formel 5.6 aus [Pap01b] berechnet (siehe Abbildung 5.5):

$$\vec{r_s} = \vec{r_1} + \left(\frac{\vec{n} \cdot (\vec{r_0} - \vec{r_1})}{\vec{n} \cdot \vec{a}} \right) \cdot \vec{a} \tag{5.6}$$

$\vec{r_s}$ Schnittpunktvektor, $\vec{r_0}$ untere linke Ecke der Plane, $\vec{r_1}$ Augenposition, $\vec{n}$ Normalenvektor der mathematischen Ebene, $\vec{a}$ Normalisierter Blickrichtungsvektor

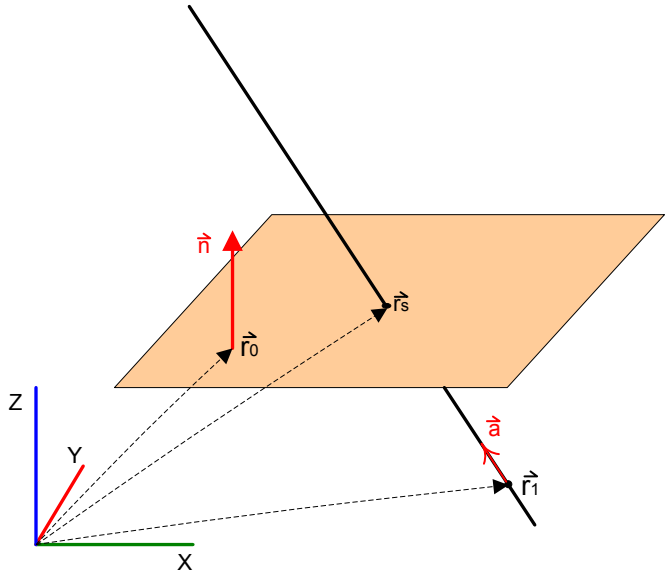


Abbildung 5.5: Darstellung zeigt den Schnittpunkt einer Geraden mit einer Ebene im dreidimensionalen Raum

Die Distanz zwischen dem Auge und dem Treffer auf der Ebene kann mit der folgenden Formel 5.7 bestimmt werden:

$$d = |\vec{r_s} - \vec{r_1}| \tag{5.7}$$

Trefferbestimmung mit umgebender Kugel

Für eine 3D-Visualisierung (siehe Kapitel 5.3) der Fixationen und Sakkaden werden die Schnittpunkte des Blickstrahls mit der Plane genutzt. Wenn kein Schnittpunkt zwischen Plane und Blickstrahl existiert, lassen sich die dortigen Fixationen und Sakkaden nicht im 3D-Modell darstellen. Um diese Punkte ebenfalls darstellen zu können, wird eine symmetrisch um den Koordinatenursprung des Blickrichtungsmessystems liegende Kugel verwendet. Die Schnittpunkte mit der Kugel werden unter Verwendung der nachfolgend beschriebenen Formeln berechnet [Bron00].

Zu einer Kugel im dreidimensionalen Raum gehört jeder Punkt, der den Abstand r vom Kugelursprung $\vec{k}$ hat, siehe Formel 5.8 und Abbildung 5.6.

$$\vec{p} = \vec{k} + \vec{r} \tag{5.8}$$

$$\text{für alle } |\vec{r}| = r$$

Daraus ergibt sich die folgende Gleichung 5.9:

$$r^2 = (p_x - k_x)^2 + (p_y - k_y)^2 + (p_z - k_z)^2 \tag{5.9}$$

Eine Gerade wird in der Punkt-Richtungs-Form in Parameterschreibweise durch die folgende Gleichung 5.10 beschrieben, wobei der Vektor $\vec{b}$ auf einen bekannten Punkt der Geraden zeigt, der Vektor $\vec{v}$ die Richtung der Geraden angibt und a einen Faktor zum Erreichen jedes beliebigen Punktes auf der Geraden darstellt:

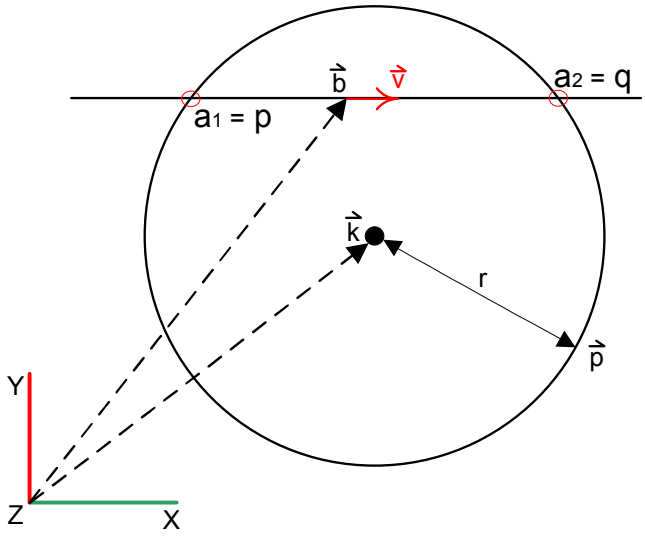


Abbildung 5.6: Darstellung zeigt die Schnittpunkte einer Geraden mit einer vereinfacht als Kreis dargestellten Kugel

$$\vec{p} = \vec{b} + \vec{v} \cdot a \tag{5.10}$$

Setzt man die Gleichung der Geraden in die Gleichung der Kugel ein und löst die resultierende Gleichung nach a auf, erhält man nach einem kleinen Umweg über die P/Q Formel die beiden Faktoren der Geradengleichung für die Schnittpunkte der Geraden mit der Kugeloberfläche. Die resultierenden Formeln für den Wert p 5.11 und q 5.12 der P/Q Formel lauten:

$$p = \frac{2\left(b_x - k_x\right)v_x + 2\left(b_y - k_y\right)v_y + 2\left(b_z - k_z\right)v_z}{v_x^2 + v_y^2 + v_z^2} \tag{5.11}$$

$$q = \frac{\left(b_x - k_x\right)^2 + \left(b_y - k_y\right)^2 + \left(b_z - k_z\right)^2}{v_x^2 + v_y^2 + v_z^2} \tag{5.12}$$

Solange sich das Auge der Testperson (Vektor $\vec{b}$ der Geradengleichung) innerhalb der Kugel befindet wird die P/Q Formel immer einen positiven und einen negativen Faktor zur Schnittpunktbestimmung mittels der Geradengleichung ergeben. Da uns für die Visualisierung der Daten des Blickrichtungsmesssystems nur der Schnittpunkt der Geraden mit der Kugeloberfläche in Richtung des normierten Blickvektors interessiert, benötigen wir nur den positiven Faktor zur Schnittpunktbestimmung.

5.2.4 Umgesetzte Analyseparameter aus der Literatur

Während der Entwicklung des Interpretationsmodules wurden ein Großteil der in dem DLR-internen Dokument [Let05] beschriebenen, für eine Analyse der Blickbewegungen benötigten Parameter integriert. Die folgende Liste zeigt die integrierten Parameter, gibt eine Kurzbeschreibung und zitiert Literaturstellen aus dem Dokument [Let05] zu den aufgelisteten Parametern. Die folgenden Parameter ab First Fixation Duration werden sowohl für Planes als auch für AOIs bestimmt:

- Plane und AOI:
  besonders definierte Bereiche innerhalb derer Fixationen gemessen werden. Die nachfolgenden Parameter werden sowohl für die angegebenen AOIs als auch für die Planes unabhängig voneinander berechnet
- Fixation duration (FD):
  Dauer einer Fixation, Berechnungsgrundlage für Variablen, die sich über Fixationen definieren
- First fixation duration (FFD):
  Dauer der ersten Fixation beim Eintritt in einen definierten Bereich (AOI); (siehe z.B. Henderson et al., 1989 [HF04]),
- Gaze duration (GD):
  die Gesamtdauer aller aufeinanderfolgenden Fixationen innerhalb einer AOI; (siehe z.B., Liversedge & Findlay, 2000 [LF00]; Starr & Rayner, 2001 [SR01])
- Viewing time (VT):
  die Gesamtdauer aller aufeinanderfolgenden Fixationen einschließlich der Dauer

der intra-regionalen Sakkaden[2]; (siehe z.B. Meyer & Lethaus, 2004 [ML04])

- Dwell (D):
  Beschreibt das Zeitintervall zwischen dem Beginn der ersten Fixation innerhalb einer AOI, über das Verlassen dieses Bereiches bis zum Beginn einer erneuten ersten Fixation nach einem Wiedereintritt in eine AOI; (siehe z.B. Karn, 2002 [Karn02])

- Dwell duration (DD):
  die Dwell duration unterscheidet sich von dem Parameter Dwell dadurch, dass das beschriebene Zeitintervall schon bei der ersten in der AOI eintretenden Sakkade beginnt und nicht erst bei Begin der ersten Fixation; (siehe z.B. Karn, 2002 [Karn02])

- Glance duration (GLD):
  beschreibt das Zeitintervall zwischen der ersten eine AOI betretenden Sakkade bis zur letzten die AOI verlassenden Sakkade; (siehe z.B. Victor et al., 2005 [VHE05])

Die Berechnung der Parameter (ab FFD) erfolgt zur Laufzeit des Interpretationsmodules in jedem Interpretationszyklus, wo der aktuelle Messwert vom Typ GUESSED_FIXATION, FIXATION oder SACCADE ist. Das Ergebnis dieser Berechnung ist zum Beispiel für den Parameter D ein Zeitintervall, welcher seinen maximalen Wert bei der ersten GUESSED_FIXATION nach dem Wiedereintritt in eine AOI hat (siehe Abbildung 5.10).

Innerhalb der Klasse „`SmplEyeTrackingDataInterpretation`“ ist jede dieser Berechnungen in einer eigenen Funktion z.B. „`CalcFixationDuration()`“ untergebracht. Diese Funktionen verwenden jeweils einen eigenen Zustandsautomaten[3] für das Berechnen der Parameter. Die Aufzählung „`enum _AnalyseStates`“ stellt die möglichen Zustände für den Automaten zur Verfügung:

[2] intra-regionale Sakkaden sind Sakkaden, welche Fixationen innerhalb einer AOI verbinden

[3] Ein endlicher Automat (EA, auch Zustandsmaschine, englisch finite state machine (FSM)) ist ein Modell des Verhaltens, bestehend aus Zuständen, Zustandsübergängen und Aktionen. [Wiki07a]

- INIT_STATE:
  Zustand bei Modulstart
- NEW_AT_AREA:
  im letzten Berechungszyklus wurde der Eintritt in eine AOI registriert
- AT_FIRST_FIXATION:
  im letzten Berechnungszyklus wurde der Beginn der ersten Fixation innerhalb einer AOI festgestellt
- AFTER_FIRST_FIXATION:
  die erste Fixation in einer AOI wurde beendet, die AOI aber noch nicht verlassen
- AT_FIXATION:
  im letzten Berechnungszyklus wurde der Beginn einer weiteren Fixation innerhalb der AOI festgestellt
- AT_SACCADE:
  Zustand zwischen Fixationen innerhalb einer AOI
- OUTSIDE_AREA:
  Im letzten Zyklus wurde die AOI verlassen

Nicht alle Parameter verwenden in ihren Berechnungsfunktionen jeden dieser Zustände. Zur Verdeutlichung der Umsetzung der Automaten zeigt das folgende UML-Zustandsdiagramm exemplarisch den Automaten der Funktion „`CalcDwellTime()`“ zum Berechnen des Analyseparameters D (siehe Abbildung 5.7).

Bei der Berechnung der Parameter gibt es in der Literatur eine Grauzone. Diese betrifft das Eintreten in eine AOI in Form einer über die AOI-Grenze wandernden GUESSED_FIXATION oder FIXATION. Es existiert keine einheitliche Definition in der Fachliteratur, aus der eindeutig hervorgeht, ob die erste Fixation, wenn sie in den Bereich der AOI hineinwandert, für die Berechnung der Parameter voll berücksichtigt werden muss. Die gleiche Grauzone gibt es für das Verlassen einer AOI. Es ist ebenfalls nicht definiert, ob eine innerhalb der AOI beginnende Fixation, wenn sie aus der AOI hinauswandert, vollständig in den zu berechnenden Parametern berücksichtigt werden

muss.

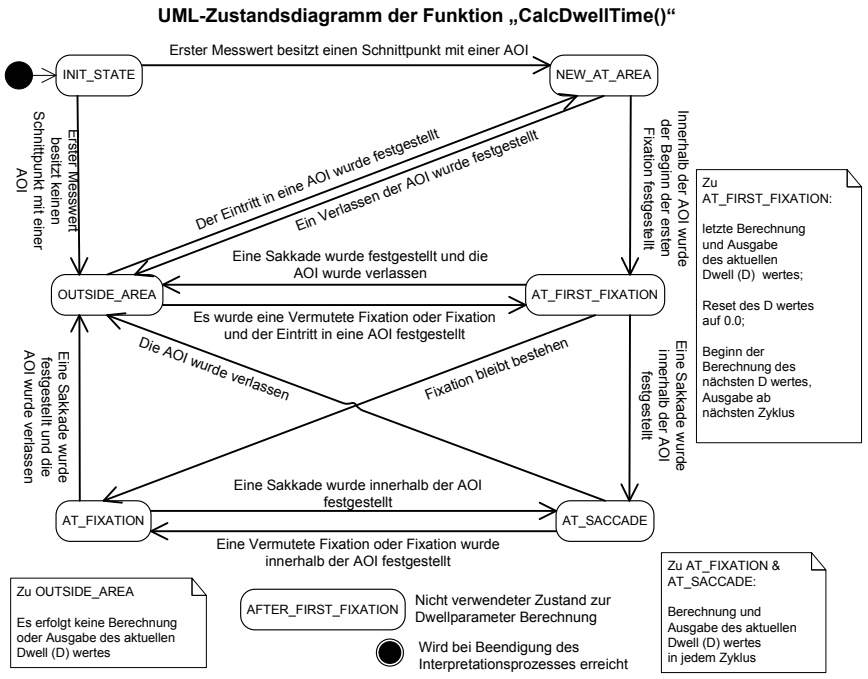


Abbildung 5.7: UML-Zustandsdiagramm der Funktion „CalcDwellTime()“

- In der aktuellen Umsetzung der Funktionen zur Berechnung der oben genannten Parameter werden sowohl Fixationen, die in eine AOI hineinwandern, als auch Fixationen die aus einer AOI hinauswandern, voll berücksichtigt.

- Aufgrund der zyklischen Berechnung und Ausgabe der Parameter zur Laufzeit des Modules werden GUESSED_FIXATIONs, die die Dauer einer vollwertigen FIXATION nicht erreichen, trotzdem bei der Berechnung der Parameter FD, FFD und GD berücksichtigt.

- Bei der Berechnung der Parameter D und DD wird die erste gefundene FIXATION oder GUESSED_FIXATION bei einem Neueintritt in eine AOI zur Berechnung des Endwertes des Zeitintervalls verwendet. Dieser Punkt wird auch dann verwendet, wenn die Fixation außerhalb der AOI beginnt und in diese hineinwandert.

Eine detaillierte Beschreibung der Randbedingungen bei der Berechnung der umgesetzten Analyseparameter befindet sich im Anhang 8.6, Seite 105 bis 8.12, Seite 111.

5.2.5 Argumentübergabe beim Start des Modules

Gestartet wird dieses Modul mittels der Main-Funktion, welche sich in der Datei „SmplEyeTrackingInterpretaion.cpp“ befindet. Üblicherweise erfolgt der Start aus einer Stapelverarbeitungsdatei heraus, wobei zusätzliche Argumente zum Initalisieren der Parameter des Modules übergeben werden können (siehe Listing: 5.3).

```
1:  start ./Programs/SmplEyeTrackingInterpretation/SmplEyeTrackingInterpretation.exe
2:      CycleMSec                           15
3:      SsmName                             ssm1
4:      EyeTrackingDataBlockName            SmplEyeTrackingData
5:      EyeTrackingInterpretedDataBlockName SmplEyeTrackingInterpretedData
6:      DefinitionFile                      ..\SMPL\EyeTracking-Def-Files\
7:                                                EyeTrackingDefinitonsSmplLab.def
8:      Thresholds                          20.0  2.0  100.0
9:      ProjectionSphereRadius              230.0
```

Listing 5.3: Starten des Interpretationsmodules mit Argumentenübergabe

Die folgenden Argumente können hierbei an das Modul übergeben werden:

- CycleMSec:
  Gibt den Abstand zwischen den Ausführungen des Callback-Teils in Millisekunden an. SW: 12
- SsmName:
  Stellt dem Modul den Namen des zu verwendenen „SSM“ zur Verfügung. SW: ssm1
- EyeTrackingDataBlockName:
  Gibt den Namen des Datenblocks an, unter welchem das Datenerfassungsmodul die empfangenen Blickdaten im „SSM“ abgelegt hat. SW: EyeTrackingData
- EyeTrackingInterpretedDataBlockName:
  Gibt den Namen des Datenblocks an, unter welchem die interpretierten Blickdaten

im „SSM“ abgelegt werden sollen. SW: EyeTrackingInterpretedData

- DefinitionFile:
  Dieses Argument übergibt den Pfad auf die Definitonsdatei, welche die Planes und AOIs beschreibt SW: EyeTracking-Def-Files//EyeTrackingDefinitonsSmpLab.def

- Thresholds:
  An dieser Stelle werden dem Modul die Schwellwerte zur Typbestimmung des Blickstrahls übergeben. Die Reihenfolge bei der Übergabe der benötigten drei Schwellwerte ist:

  1. Geschwindigkeitsschwellwert in $°/s$ SW: 20
  2. örtlicher Schwellwert in ° SW: 2
  3. zeitlicher Schwellwert in ms SW: 100

- ProjectionSphereRadius:
  Gibt den Radius der einhüllenden Kugel an, an welcher alle Datenpunkte außerhalb einer Plane visualisiert werden sollen. SW: 220

5.3 Eye Tracking Datenvisualisierung

Durch die Datenerfassung und die Interpretation der Rohdaten des angeschlossenen Blickrichtungsmesssystems erhalten wir einen kontinuierlichen Datenfluss von Informationen. Diese Informationen werden über das „SSM“ allen anderen Modulen der Smpl++ Applikations- und Entwicklungsplattform in angenäherter Echtzeit zugänglich gemacht und für ein späteres Replay gesichert. Da die Daten in ihrer Rohform zwar einen hohen Informationsgehalt besitzen, diesen aber durch Zahlenkolonnen nur schwer vermitteln, ist eine Visualisierung der vorliegenden Daten unumgänglich.

Die nachfolgenden zwei Unterkapitel beschreiben sowohl die Visualisierung der empfangenen Rohdaten, als auch die Visualisierung der interpretierten Daten in Diagrammform über das Smpl++-Modul „`SmplcaSBAro`“. Das dritte Unterkapitel beschreibt die Visualisierung der Blickrichtung in einem 3D-Modell des SMPLabs in Verbindung mit dem Smpl++-Modul „`SmplFSViewer`“.

5.3.1 Visualisierung der Rohdaten

Die erfassten Rohdaten „`SmplEyeTrackingData`“ (siehe UML-Diagramm 5.1) des Blickrichtungsmesssystems durch das iViewX HED/HT System von SMI (siehe Kapitel 2.2.6), können vollständig in Form von Diagrammen (siehe Abbildungen 5.9 und 5.8) mittels des Modules „`SmplcaSBAro`“ dargestellt werden.

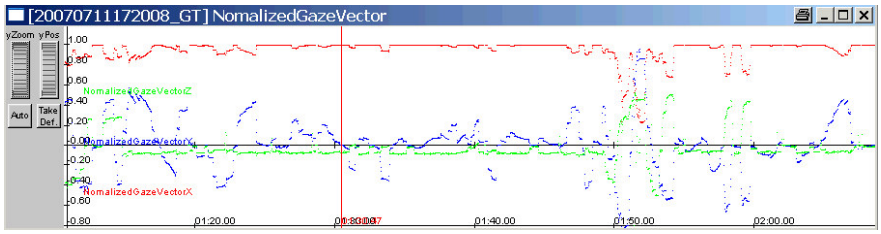


Abbildung 5.8: Darstellung des normierten Blickvektors in Diagrammform mittels „SmplcaSBAro“

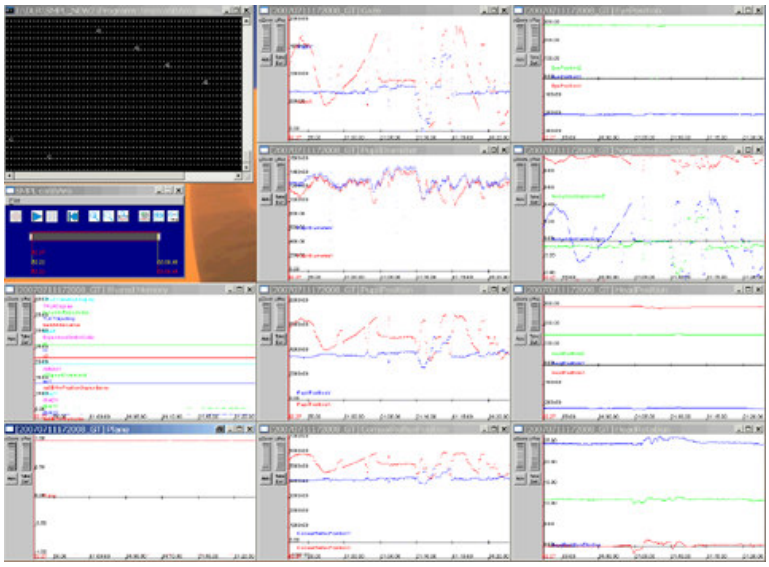


Abbildung 5.9: Darstellung aller Rohdaten des Blickrichtungsmesssystems in Diagrammform mittels „SmplcaSBAro“

Die Abbildung 5.8 zeigt einen Plot des normierten Blickvektors durch das Modul „`SmplcaSBAro`“. In dem Diagramm sind die Werte für die X-Komponente (rot), die Y-Komponente (blau) und der Z-Komponente (grün) des Vektors über der Zeit des SharedMemoryTimers (siehe Kapitel 2.1.7 unter SmplSharedMemory) in Sekunden aufgetragen. Auf den ersten Blick läßt sich anhand der relativ nahe bei eins liegenden Linie der X-Komponente erkennen, dass die Testperson überwiegend in Richtung der Leinwand des Simulators (mit Bezug auf das Koordinatensystem des Blickrichtungsmesssystems) geschaut hat.

5.3.2 Visualisierung der interpretierten Daten

Das Interpretationsmodul bestimmt anhand der erfassten Rohdaten des Blickrichtungsmesssystems, ob die Testperson eine der zuvor definierte Planes betrachtet. Zusätzlich ist es möglich, bestimmte AOI innerhalb dieser Ebene zu bestimmen, um die Blickrichtung der Testperson noch besser auswerten und verschiedene Analyseparameter aus der Literatur berechnen zu können. Die Ergebnisse dieser Berechnungen werden anschließend in Diagrammform über das Modul „`SmplcaSBAro`“dargestellt.

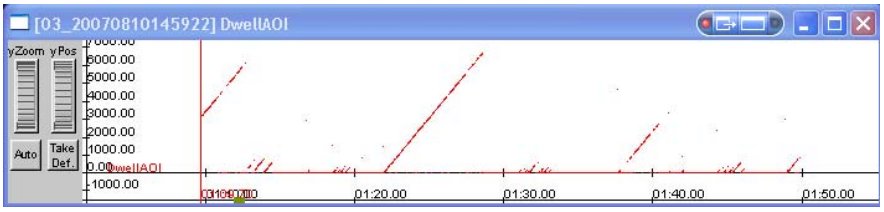


Abbildung 5.10: Darstellung des Analyseparameters Dwell (D) bzgl. AOIs in Diagrammform mittels „SmplcaSBAro“

Abbildung 5.11: Darstellung aller interpretierten Daten des Interpretationsmodules in Diagrammform mittels „SmplcaSBAro“

Die Abbildung 5.10 zeigt einen Plot des Analyseparameters Dwell (siehe Abschnitt 5.2.4) für die AOIs. In dem durch „`SmplcaSBAro`“ erzeugten Diagramm wird die Dwell in Millisekunden über der Zeit des „SharedMemory-Timers“ (siehe Kapitel 2.1.7 unter „`SmplSharedMemory`“) in Sekunden aufgetragen. Die vielen, dicht beieinander liegenden Punkte stellen die Dauer des Dwellparameters in einer AOI ab der ersten vermuteten Fixation dar. Der einzelne Punkt (z.B. über der eins von 01:40:00) beschreibt den Maximalwert eines Dwellwertes, welcher bei der ersten vermuteten Fixation nach Verlassen und erneutem Betreten einer AOI berechnet wird.

5.3.3 Visualisierung der Blickrichtung im 3D-Modell des SMPLabs

Die Darstellung der Daten in Diagrammform bietet zusammen mit dem Livebild des Blickrichtungsmesssystems, welches auch für ein Replay aufgezeichnet werden kann, eine gute Möglichkeit, das Verhalten der Testperson während eines Versuches zu analysieren. Auch wenn die Diagrammform den Informationsgehalt der Daten besser wiedergibt als es Zahlenkolonnen könnten, ist es oft doch schwierig, anhand der Diagramme die genaue Blickrichtung der Testperson innerhalb des Versuchsaufbaus nachzuvollziehen.

Eine mögliche Abhilfe könnte eine zweidimensionale (2D) Darstellung des Sichtbereiches sein (siehe Abbildung 5.13). Diese Variante der Darstellung eignet sich erfahrungsgemäß [Fle00] gut für die Dokumentation der analysierten Ergebnisse einer Versuchsreihe in schriftlicher Form.

Die 2D-Darstellung der interpretierten Augenbewegungsdaten hat den Nachteil, dass sie eine Analyse und Visualisierung der Daten nur aus zuvor implementierten Perspektiven erlaubt. Für größre Flexibilität in der Auswahl der zu visualisierenden Perspektive der interpretierten Daten kann ein 3D-Modell verwendet werden. Ein solches Modell würde durch eine frei positionierbare Kamera eine freie Perspektivenwahl für die Visualisierung des Versuchsaufbaus (z.B. das SMPLab) ermöglichen. Somit ist ein solches 3D-Modell in der Lage, fast alle denkbaren Perspektiven als 2D-Darstellungen ohne zusätzlichen Implementierungsaufwand auf dem Monitor wiederzugeben. Nachteilig an der Umsetzung der Visualisierung durch ein 3D-Modell ist der erhöhte Arbeitsaufwand für die Implementierung des Modells gegenüber einer 2D-Perspektive. Hierbei beansprucht eine notwendige Einarbeitung in eine 3D-Bibliothek die meiste zusätzliche Zeit.

Zur Umsetzung der 3D-Darstellung wird die freie 3D-Bibiliothek „Open Scene Graph“ (OSG) verwendet. Diese findet innerhalb der Smpl++-Umgebung schon Verwendung bei der Generierung und Darstellung der zu befahrenden Versuchsstrecken durch das Modul „`SmplFSViewer`“.

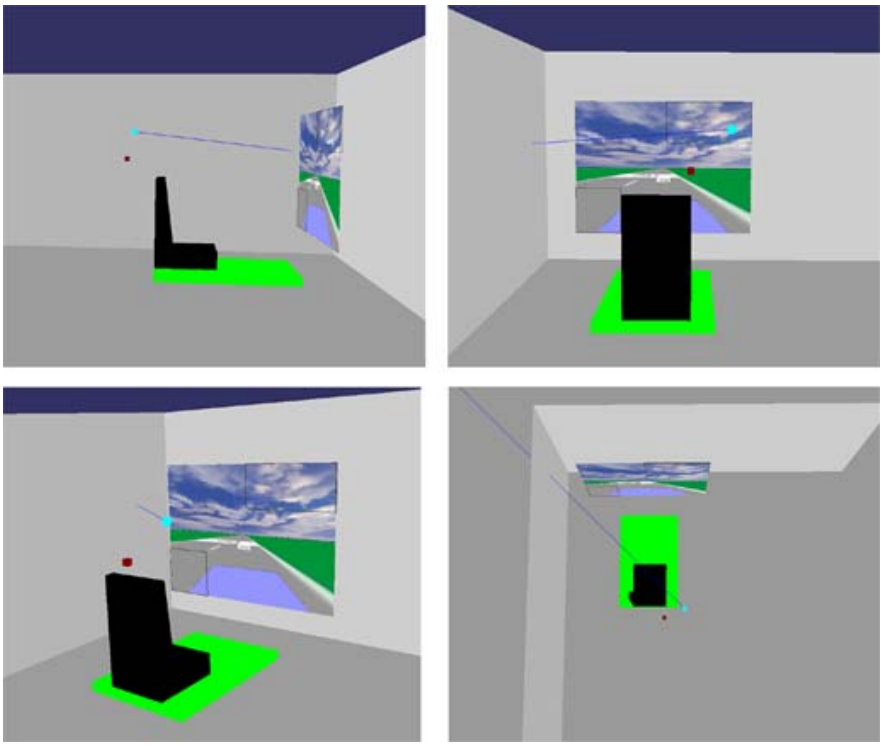


Abbildung 5.12: 3D-Modell des SMPLabs

Die Abbildung 5.12 zeigt ein 3D-Modell des SMPLabs. In dem Modell werden folgende Objekte aus dem SMPLab dargestellt:

- Raum:
  Der Raum besteht in der Abbildung 5.12 nur aus einer sehr großen Ebene mit definiertem Abstand zum iViewHT Cube. Zusätzlich ist ein Zuschalten der linken und vorderen, hinter dem SimulationScreen stehenden, Wand möglich.

- Plattform des Simulators:
  Die Basisplattform, an welcher alle benötigten Steuer- und Sitzkomponenten des Simulators befestigt sind.

- Fahrersitz:
  Modell des Fahrersitzes des Simulators.

- Koordinatenursprung:

  Der Koordinatenursprung des Blickrichtungsmesssystems (bei SMI der magnetfelderzeugende Würfel des iViewX HT Systems) wird innerhalb des 3D-Modells als Würfel visualisiert. Das Koordinatensystem des 3D-Modells ist identisch mit dem Koordinatensystem des Blickrichtungsmesssystems, so dass der dargestellte Würfel auch den Nullpunkt des 3D-Modells beschreibt.

- Achsen des Koordinatensystems

  Zur Veranschaulichung des Koordinatensystems können die Achsen eingeblendet werden. In Abbildung 5.12 wird die X- Achse des Koordinatensystems rot, die Y-Achse grün und die Z- Achse blau dargestellt.

- Auge:

  Die Kugel symbolisiert die durch SMI analysierte Position des Auges während des Versuches. Die Linie, welche von dem Auge ausgeht, veranschaulicht die Blickrichtung der Testperson. Anhand der Schnittpunkte dieser Linie mit anderen Objekten innerhalb des 3D-Modells läßt sich leicht erkennen, wohin die Testperson gerade schaut.

- Definierte Ebenen (Planes):

  Die vor dem Versuch definierten Ebenen (Planes) werden innerhalb des 3D-Modells als weiße Rechtecke dargestellt. Eine Ausnahme bildet die Ebene mit dem Namen „SimulationScreen“. Sie überträgt das Bild der Fahrerkamera, sodass die Fahrt der Versuchsperson im Simulator dort wie auf einer Leinwand dargestellt wird. Sowohl das 3D-Labor als auch die 3D-Teststrecke werden von dem gleichen osg.Viewer erzeugt, sind aber bis auf die Übertragung der Fahrt auf die „SimulationScreen“ Plane als Leinwand vollständig seperate 3D-Modelluniversen.

- Definierte AOIs:

  Die besonderen AOIs befinden sich innerhalb der dargestellten schwarzen Rahmen auf den Planes.

Visualisierung von Fixationen und Sakkaden im 3D-Modell

Neben einer Darstellung der Blickrichtung in Form eines Blickstrahls werden für eine vereinfachte Analyse der aufgenommenen Blickrichtungsmessdaten die interpretierten Fixationen und Sakkaden ebenfalls in der Visualisierung dargestellt.

Die Abbildung 5.13 zeigt eine der 2D-Varianten zur Visualisierung von Fixationen und Sakkaden aus der Dissertation [Fle00]. Diese 2D-Abbildung ist nicht verzerrt, da die dargestellten Bedienelemente eines Flugzeugcockpits annährend im Halbkreis um den Piloten angeordnet waren. In der Abbildung stellen rote Punkte gemessene Blickpunkte, rote Kreuze interpretierte Fixationen und die schwarzen Linien Direktverbindungen der Fixationen untereinander dar.

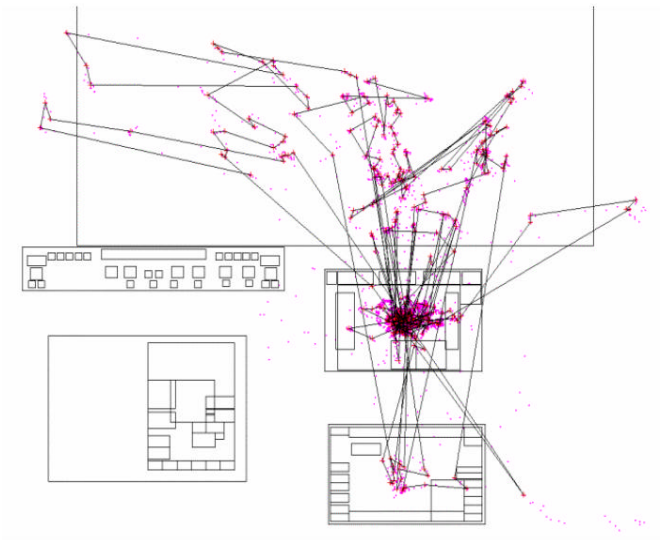


Abbildung 5.13: Fixationen und Sakkaden in einer 2D-Darstellung [Fle00]

Für die Visualisierung von Fixationen und Sakkaden innerhalb des 3D-Modells des SMPLabs stehen durch diese Arbeit verschiedene darstellende Ansichten zur Verfügung, die mittels der Taste „F12“ bei aktivem „`SmplFSViewer`“ in der angegebenen Reihenfolge durchgewechselt werden können („`enum EyeTrackingViewModes`“ in `SmplFSViewer.cpp`):

1. SIMULATE_THE_WORLD:

   Bildschirmfüllende Darstellung der zu befahrenden Teststrecke, es werden keine Blickrichtungsmessdaten dargestellt.

2. SMPLAB:

   Darstellung des SMPLabs, Visualisierung des Blickstrahls (siehe Abbildung: 5.12).

3. BASICDATA:

   Reduzierte Darstellung des SMPLabs auf Planes und AOIs. Einblenden der Interpretierten Blickpunkte durch Kugeln (siehe Abbildung: 5.14 (1)).

4. BASICS_WITH_FIXCONNECTIONLINES:

   Gleiche Konfiguration wie bei BASICDATA, zusätzlich werden die bei der Interpretation ermittelten Blickpunkte vom Typ FIXATION und GUESSED_FIXATION als 3D-Kreuz in das 3D-Modell eingeblendet. Die Blickpunkte vom Typ FIXATION und GUESSED_FIXATION werden mittels einer Linie direkt miteinander verbunden dargestellt (siehe Abbildung: 5.14 (2)).

5. BASICCONNECTIONLINES_WITH_FIXATIONS:

   Gleiche Konfiguration des SMPLabs und der Blickpunkte wie bei BASICS_WITH_FIXCONNECTIONLINES. In dieser Ansicht werden allerdings die Verbindungslinien zwischen den Blickpunkten vom Typ FIXATION und GUESSED_FIXATION über die verbleibenden Blickpunkte dargestellt (siehe Abbildung: 5.14 (3)).

6. FIXATIONCONNECTIONLINES:

   Reduziertes SMPLab, 3D-Kreuz zur Darstellung der Blicktypen FIXATION und GUESSED_FIXATION sowie Direktverbindungslinien zwischen Blickpunkten von diesem Typ. Keine Darstellung der Basispunkte (siehe Abbildung: 5.14 (4)).

7. FIXATIONCONNECTION_OVER_BASICS:

   Reduziertes SMPLab, 3D-Kreuz zur Darstellung der Blicktypen FIXATION und GUESSED_FIXATION sowie Verbindungslinien zwischen den Blickpunkten dieses Typs über die nicht dargestellten Blickpunkte (siehe Abbildung: 5.14 (3 ohne die Basispunkte)).

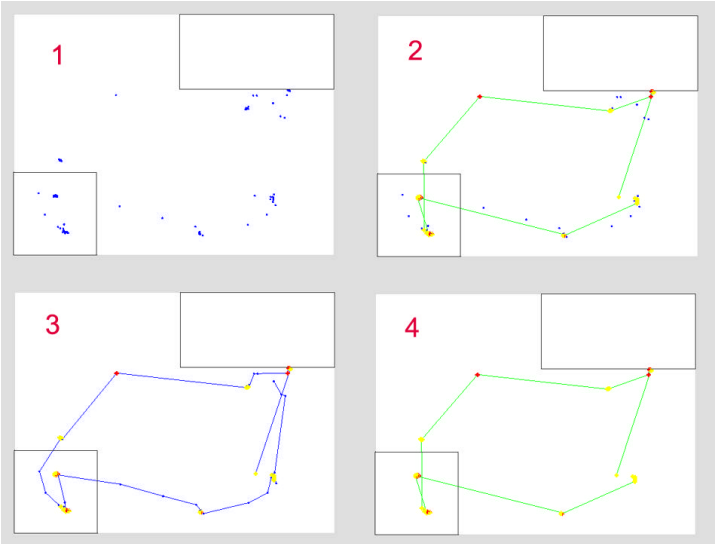


Abbildung 5.14: verschiedene Darstellungen von Fixationen und Sakkaden im 3D-Modell

Da die Darstellung von Fixationen und Sakkaden mit zunehmender Anzahl von dargestellten Messpunkten unübersichtlich wird und da bei zunehmender Anzahl von darzustellenden 3D-Objekten die Framerate des „`SmplFSViewers`“ sinkt, ist es möglich, mittels der Taste „F11“ alle eingeblendeten 3D-Objekte der Blickpunkte, Fixationen, vermutete Fixationen und der Verbindungslinien aus dem 3D-Modell zu entfernen.

Aktuelles Darstellungsschema in der Visualisierung

Zur besseren Unterscheidung von Fixationen, vermuteten Fixationen, sowie Sakkaden und ermittelten Blickpunkten werden diese in unterschiedlichen Farben und Formen im 3D-Modell des SMPLabs dargestellt. Als Vorlage für die gewählte Farbverteilung der verschiedenen Blicktypen dient das Modell einer Ampel:

- ROT: Fixation, dargestellt als 3D-Kreuz
- GELB: Vermutete Fixation, eine Fixation könnte stattfinden, ebenfalls als 3D-Kreuz dargestellt
- GRÜN: Sakkade, sichtbar bei der Darstellung des Blickstahles (siehe 5.12)

Die Blickpunkte sind als blicktypneutral in der Farbe BLAU dargestellt und werden durch kleine Kugeln visualisiert.

Die Linien der Direktverbindungen zwischen den Blicktypen FIXATION und GUESSED_FIXATION werden GRÜN, bzw. die Verbindungslinien der genannten Blicktypen über die Blickpunkte werden BLAU dargestellt.

Eine Änderung des Farbschemas läßt sich jederzeit durch geringe Anpassungen der RGB-Werte in den Funktionen der Klasse „`SmplEyeTracking3DVisualization`“ erreichen, welche die 3D-Objekte erstellen Zum Beispiel können in der Funktion „`BuildFixationPoints()`“ die Farben für eine Fixation oder Vermutete Fixation der 3D-Kreuze verändert werden.

UML-Darstellung der 3D-Visualisierungsklasse

Das folgende UML-Diagramm (siehe Abbildung 5.15 und vergrößert im Anhang 8.4, Seite 103) zeigt die Klasse „`SmplEyeTracking3DVisualization`“ integriert in das „SmplFSViewer“-Projekt. Die Klasse „`SmplEyeTracking3DVisualization`“ erzeugt nach dem Aufruf des Konstruktors „`SmplEyeTracking3DVisualization( string DefFile, ...)`“ anhand der in der Definitionsdatei angegebenen Elemente (siehe weiter unten 5.3.3) die 3D-Objekte.

Durch die Funktion „`AddET3DVisualisationToGroup(osg::Group*Root, float XPos,float YPos,float ZPos)`“ wird das generierte 3D-Modell mit der bestehenden 3D-Welt des „`SmplFSViewers`“ verbunden und dargestellt.

Zur Aktualisierung der VirtualEyePosition und der VirtualGazePosition muss das „`SmplFSViewer`“ -Modul regelmäßig die Funktion „`ReadDataFromSSM()`“ ausführen, die die aktuelle Position des Auges und den normierten Blickvektor aus den im „SSM“ hinterlegten interpretierten Blickrichtungsdaten definiert in „`SmplEyeTrackingInterpretedData.h`“ kopiert und die entsprechenden 3D-Modelle neu positioniert.

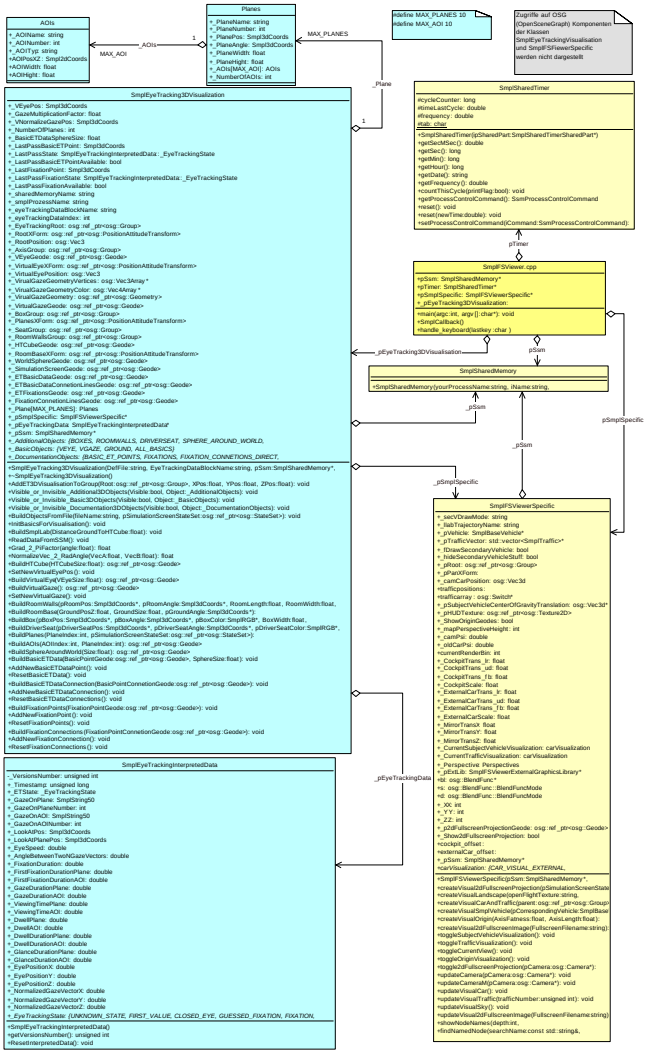


Abbildung 5.15: UML-Darstellung der in den „SmplFSViewer“ integrierten „SmplEye-Tracking3DVisualization“ Klasse, siehe auch im Anhang 8.4 Seite 103

Zusätzlich verfügt die Klasse „`SmplEyeTracking3DVisualization`“ über die Möglichkeit alle für die Darstellung der Blickrichtung nicht unbedingt notwendigen 3D-Objekte, ein/auszublenden. Dies wird durch die Funktion „`Visible_or_Invisible_Additional3DObjects(bool Visible, _AdditionalObjects Object)`“ ermöglicht. `_AdditionalObjects` sind alle Boxen *BOXES*, sowie Fahrzeugsitze *SEAT*, die Achsen des Koordinatensystems *AXIS* und die Wände des Raumobjekts *ROOMWALLS*. Zur Vereinfachung gibt es zusätzlich den AdditionalObject-Typ *ALL_ADDITIONALS*, mit welchem alle zusätzlichen 3D-Objekte auf einmal ein- bzw. ausgeblendet werden können.

Die Funktion „`Visible_or_Invisible_Basic3DObjects( bool Visible, _BasicObjects Object)`“ ermöglicht das Ein- Ausschalten von Basisobjekten für die Visualisierung des SMPLabs wie das virtuelle Auge *VEYE*, Blickstrahl *VGAZE* oder Boden des Raums *GROUND*. Außerdem steht der Typ *ALL_BASICS* zum Ein- bzw. Ausschalten aller Basis-3D-Objekte zur verfügung.

Die dritte Funktion dieser Art „`Visible_or_Invisible_Documentation3DObjects ( bool Visible, _DocumentationObjects Object )`“ ermöglicht das Ein- und Ausschalten der 3D-Objekte zur Darstellung von Fixationen und Sakkaden in dem 3D-Modell. Es stehen die Parameter *BASIC_ET_POINTS* für die Blickpunkte, *FIXATIONS* für die 3D-Kreuze, *FIXATION_CONNECTIONS_DIRECT* für die Direktverbindungslinien zwischen Fixationen und vermuteten Fixationen, sowie *FIXATION_CONNECTIONS_OVER_BASICS* für die Verbindung über die Blickpunkte zur Verfügung. Mit *ALL_DOCUMENTATIONS* lässt sich wie zuvor auch alle 3D-Objekte gleichzeitig ein- bzw. ausschalten.

3D-Modell Initialisierung

Beim Erzeugen der für die Visualisierung des 3D-Modells des SMPLabs zuständigen Klasse „`SmplEyeTracking3DVisualization`“ wird dieser eine auf ASCII-Zeichen basierende Datei bekannt gegeben. Diese Datei ermöglicht ein Parametrisieren der Objekte innerhalb des 3D-Modells. Der Aufbau der Datei wurde den bereits in anderen Modulen der Smpl++-Plattform Verwendung findenen Dateien nachempfunden. Der Aufbau dieser Dateien weist Parallelen zu dem gängigen XML-Schema auf. Es besitzt auch Elemente, welche aus einem passenden Paar *<Start Tag>* und *</End Tag>* bestehen, gefolgt von einer Reihe elementabhängiger Attribute.

Das erste Element ist *<3dLab_INIT>* . Wird diesem Element als „InitTyp“ das Attribut „SmpLab“ übergeben, muss ebenfalls das zweite Attribut „DistanceHTCube_toFloor“ in [cm] mit angegeben werden (siehe Listing 5.4). Bei der

Auswahl dieses Initialisierungstyps greift die Klasse „`SmplEyeTracking3DVisualization`“ auf die Funktion „`SmplLabParameterInit()`“ zurück, welcher alle noch benötigten Parameter zur Erzeugung des 3D-Modells vorliegen.

```
1:  <3dLab_INIT>
2:    InitType SmplLab DistanceHTCube_toFloor -126.0
3:  </3dLab_INIT>
```

Listing 5.4: SmplLab Model Initialisierung

Zum individuellen Erstellen eines 3D-Modells steht der „InitTyp“ „Generic“ zur Verfügung. Mit dem Element *<3DObject>* können in dem „Generic“ Modus verschiedene 3D-Objekte erzeugt werden. Das erste Attribut des Elements *<3DObject>* ist der Typ, der festlegt, was für ein 3D-Objekt erzeugt wird. Je nach Typwahl unterscheiden sich die folgenden Attribute. Es stehen folgende Typen zur Auswahl:

- BOX:
  Dieser Typ dient zum Erzeugen einer beliebigen Kiste an übergebener Position mit übergebenen Winkel, Farbe und Abmaßen. Dieser Typ kann beliebig oft zum Erzeugen weiterer Kisten verwendet werden. Die erzeugten Kisten zählen zu den „AdditionalObjects“ und können alle gemeinsam ein- bzw. ausgeblendet werden.
- SEAT:
  Mit dem SEAT Typ wird ein Fahrersitz mit den übergebenen Abmaßen, der Position, Winkel und der übergebenen Farbe erzeugt. Mehrfachanwendung dieses Typs zum Erzeugen weiterer Sitze ist möglich. Die erzeugten Fahrersitze gehören ebenfalls zu den „AdditionalObjects“ und können alle zusammen sichtbar oder unsichtbar geschaltet werden.
- ROOM:
  Bei Angabe der Abmaße sowie der linken unteren Ecke des Raumes werden als „AdditionalObjects“ die linke und die hintere Wand des Raumes erzeugt. Mehrfachnutzung dieses Types ist möglich.

- AXIS:
  Erzeugt als ein- bzw. ausblendbares Objekt die Achsen des Koordinatensystems mit der eingestellten Länge und Dicke.

- GROUND:
  Dient zum Erstellen der Bodenebene mit übergebener Größe, Winkel und Abstand in Z-Richtung zum HTCube. Der Boden gehört nicht zu den „AdditionalObjects“, muss aber dennoch nicht zwingend erzeugt werden.

- HTCUBE:
  Visualisiert den Nullpunkt des Koordinatensystems in Form eines Würfels mit der eingestellten Größe. Gehört ebenfalls nicht zu den „AdditionalObjects“ muss trotzdem nicht unbedingt ausgeführt werden.

- VEYE:
  Für die Darstellung der Blickrichtung muss mit diesem Typ der Durchmesser des Auges eingestellt werden.

- VGAZE:
  Die Länge der Linie, welche die Blickrichtung darstellt, wird über den hier eingestellten Faktor bestimmt. Daher muss dieser Typ ebenfalls einmal bei der „Generic“ -Initalisierung angegeben werden.

Das folgende Listing 5.5 zeigt die Erzeugung des SmplLabs über die „Generic“ initalisierung. Die Maßeinheit für Positionsangaben oder Abmaße der Objekte ist [cm], für Winkel [Grad], Farben müssen im Bereich [0..1] angegeben werden und Faktoren sind [einheitenlos].

```
<3dLab_INIT> InitType Generic

  // Position means the left down edge of a 3DObject
  <3DObject>
     Type BOX
     BoxColor 1.0 0.6 0.5
     PositionXYZ_toHTCube -61.7 50.0 -126.0
     AngleXYZ_toHTCube 0 0 -46
     Size_WidthLengthHight 100 160 10
  </3DObject>

  <3DObject>
     Type SEAT
     SeatColor 0.5 0.5 1.0
     PositionXYZ_toHTCube -61.7 50.0 -115.0
     AngleXYZ_ToHTCube 0 0 -46
     DSWidth_DSLength_DSSeatLength_DSBacklength_DSHight_DSSeatHight
             55.0 70.0 50.0 10.0 100.0 20.0
  </3DObject>

  <3DObject>
     Type ROOM
     PositionXYZ_toHTCube -411.7 -173.53 -126.0
     AngleXYZ_toHTCube 0 0 -46
     Size_RWidth_RLength_RHight 482.0 661.0 250.0 1500.0
  </3DObject>

  <3DObject>
    Type GROUND
    PosZ -126.0
    AngleXYZ_toHTCube 0 0 -46
    Size 1500.0
  </3DObject>

  <3DObject>
           Type HTCUBE
     Size 5.0
  </3DObject>

  <3DObject>
     Type VEYE
     SizeDuration 5.0
  </3DObject>

  <3DObject>
     Type VGAZE
     GazeFactor 1000.0
  </3DObject>

  <3DObject>
     Type AXIS
     AxisFatness 0.8
     AxisLength 1000.0
  </3DObject>
</3dLab_INIT>
```

Listing 5.5: Generic Model Initialisierung

Initialisierung von Planes und AOIs

In der Datei zum Initialisieren des 3D-Modells werden außerdem die Planes und AOIs definiert. Die Klasse „`SmplEyeTracking3DVisualization`“ bietet im Moment die Möglichkeit, zehn Planes mit jeweils zehn AOIs zu verwalten (siehe Definitionen von `MAX_PLANES`, `MAX_AOI` in `SmplEyeTracking3DVisualization.h`). Eingeleitet wird der Definitionsbereich der Planes und AOIs durch das Element <*PlaneSection*> (siehe Listing 5.6). Eine Plane beginnt nach dem Element <*Plane*>. Folgende Argumente sind dabei für die Planes zu definieren:

- Number:
  Die Nummer der zu erstellenden Plane.

- Name:
  Ein beliebig gewählter String als Name für die Plane. **Ausnahme:** Bei dem String „SimulationScreen“ erhält die Plane die Kamera des fahrenden Testfahrzeuges als Textur zugewiesen, sodass die Fahrt der Testperson direkt auf der Plane dargestellt wird.

- PlanePos:
  Dreidimensionaler Vektor auf die untere, linke Ecke der darzustellenden Plane. Die Beschreibung dieses Punktes erfolgt im Koordinatensystem von „EyeTreckingRoot“ siehe Abbildung 5.17. Maßeinheit: [cm]

- PlaneAngleXYZ:
  Angabe des Winkels zur Rotation der Plane um die X-,Y-,Z-Achse des „EyeTrackingRoot“ Koordinatensystems. Maßeinheit: [Grad]

- PlaneWidth:
  Breite der Plane in X-Richtung, im „PlaneXForm“ Koordinatensystem der jeweiligen Plane. Maßeinheit: [cm]

- PlaneHight:
  Höhe der Plane in Z-Richtung, im „PlaneXForm“ Koordinatensystem der jeweiligen Plane. Maßeinheit: [cm]

Beendet wird eine Plane durch *</Plane>*. Innerhalb eines Plane-Elementes kann mittels das Elementes *<AOI>* jeweils eine Area of Intrest definiert werden. Die folgenden Argumente sind dabei zu definieren:

- Number:
  Die Nummer der zu erstelenden AOI. Diese Nummer sollte für eine problemlose Darstellung der AOIs in einem Diagramm planeübergreifend einmalig sein.

- Name:

  Bezeichnung der zu definierenden AOI in Form eines Strings.

- Typ:

  Form der AOI. Bisher ist nur der Typ RECT (rectangle) implementiert.

- AOIPos:

  X-Z-Vektor auf die Position der unteren linken Ecke (bei Typ RECT) der zu erstellenden AOI in dem Koordinatensystem der jeweiligen Plane „PlaneXPos“.

- AOIWidth:

  Breite der AOI in X-Richtung der Plane im „PlaneXPos“ Koordinatensystem. **Achtung:** Keine Maximalbegrenzung der AOI-Breite durch Breite der Plane.

- AOIHight:

  Höhe der AOI in Z-Richtung der Plane im „PlaneXPos“ Koordinatensystem. **Achtung:** Keine Maximalbegrenzung der AOI-Höhe durch Höhe der Plane.

Durch das Element *</AOI>* ist die Definition einer AOI beendet.

```
 1:  // How to build Planes and AOIs (AOIs: Now only typ
 2:  // rectangle "RECT" implemented)
 3:  //
 4:  //  +-----------------------------------+
 5:  //  |                                   |
 6:  //  |Z/\    +AOIWidth----+          PlaneHight
 7:  //  |  |    |        AIOHight           |
 8:  //  |  |    AOIPosXZ-----+ -->X         |
 9:  //  |                                   |
10:  //  PlanePosXYZ----------PlaneWidth-----+
11:  //
12:  // Plane with name "SimulationScreen" will show the
13:  // drivercamera as texture
14:
15:  <PlaneSection>
16:    <Plane>
17:      Number 0 Name SimulationScreen PlanePos 216.0 -128.0 89.0 PlaneAngleXYZ
             180.0 0.0 90.0  PlaneWidth 193.0 PlaneHight 145.0
18:      <AOI> Number 1 Name AOI1 Typ RECT AOIPos 0.0 0.0 AOI_Width 50.0 AOI_Hight
             50.0 </AOI>
19:      <AOI> Number 2 Name AOI2 Typ RECT AOIPos 100.0 100.0 AOI_Width 93.0
             AOI_Hight 45.0 </AOI>
20:    </Plane>
21:
22:    <Plane>
23:      Number 1 Name Plane2 PlanePos -42.64 98.523 -96.0 PlaneAngleXYZ -45.0 0.0
             40.0  PlaneWidth 60.0 PlaneHight 50.0
24:      <AOI> Number 3 Name AOI1 Typ RECT AOIPos 10 10.0 AOI_Width 40.0 AOI_Hight
             30.0 </AOI>
25:    </Plane>
26:  </PlaneSection>
```

Listing 5.6: Definieren von Planes und AOIs

Organisation der 3D-Objekte

Zur besseren Übersicht über die Verknüpfung der einzelnen 3D-Objekte der Klasse „`SmplEyeTracking3DVisualization`“ durch Strukturen der 3D-Bibliothek OSG, stellt die Abbildung 5.16 bzw. im Anhang 8.5, Seite 104 vergrößert den Aufbau des 3D-SMPLabs in einen Baumdiagramm dar. Die gelb markierten Felder sind OSG-Objekte vom Typ „`osg::PositionAttitudeTransform`“ und beschreiben jeweils Koordinatentransformation für die an diesem Zweig des Baumdiagrammes angehängten 3D-Objekte.

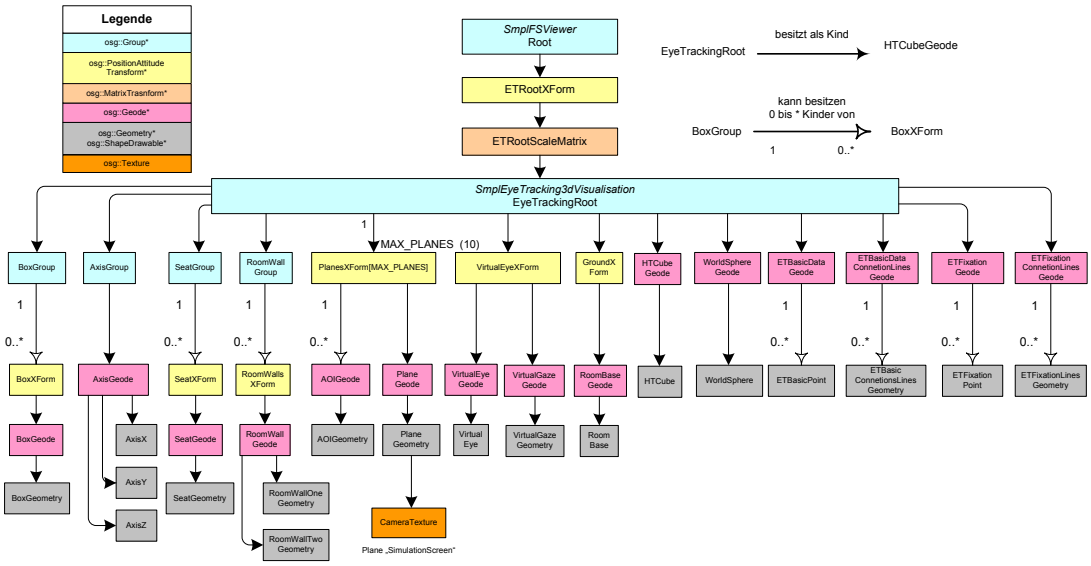


Abbildung 5.16: Organisation der 3D-Objekte als Baumdiagramm

Die Abbildung 5.17 zeigt die verschiedenen Koordinatensysteme innerhalb der 3D-Visualisierung, die durch die Verwendung des OSG-Objekts „`osg::PositionAttitudeTransform`“ entstehen.

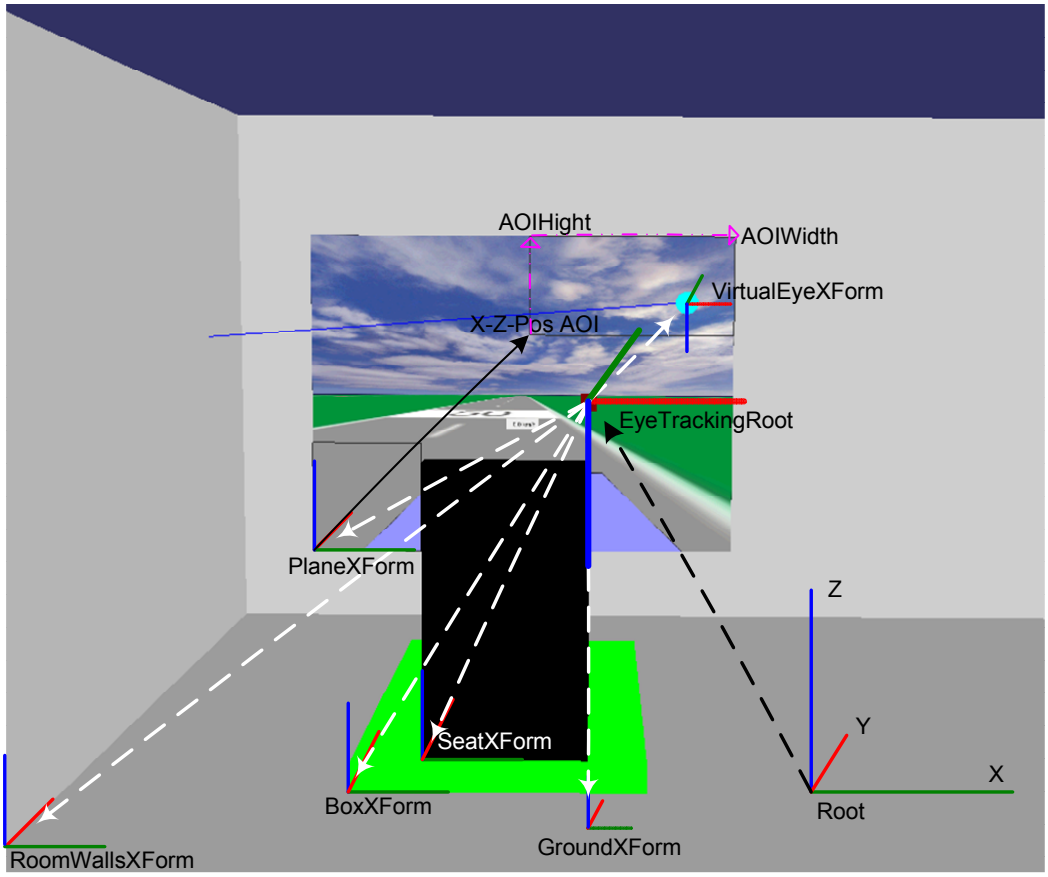


Abbildung 5.17: zeigt die verschiedenen Koordinatensysteme der 3D-Visualisierung

Kapitel 6

Softwareentwicklung - Verifikation

Das letzte Kapitel dokumentiert die Implementierungsphase dieser Masterarbeit. Es dokumentiert das entwickelte Datenerfassungsmodul „`SmplEyeTrackingRemoteControl`“ zur Kommunikation mit den Blickrichtungsmessgeräten, es beschreibt sowohl die verwendete Mathematik als auch das Modul „`SmplEyeTrackinInterpretation`“ zur Interpretation der durch das Datenerfassungsmodul erhaltenen Augenbewegungsdaten und es stellt die Ergänzungen des Moduls „`SmplFSViewer`“ vor, die eine Visualisierung der interpretierten Augenbewegungsmessdaten in einem 3D-Modell ermöglichen.

Dieses Kapitel beschreibt die verwendeten Tests zur Verifikation der erzeugten Daten der entwickleten Module und Modulergänzungen.

6.1 Verifikation der Daten des Datenerfassungsmodules

Zur Verifikation der empfangenen Daten des Blickrichtungsmesssystems von SMI durch das Modul „`SmplEyeTrackingRemoteControl`“ wird ein Soll ⇔ Ist Vergleich verwendet. Die Sollwerte werden durch lokal auf dem SMI-Rechner gespeicherte Messwerte des Blickrichtungsmessgerätes vorgegeben, die Istwerte durch die über das Datenerfassungsmodul erhaltenen Daten, welche mittels des Modules „`SmplcaSBAro`“ dem SharedMemory entnommen, gespeichert und in Form von Tabellen und Diagrammen visualisiert werden.

Der Vergleich von aufgezeichneten Solldaten mit Istdaten ergab, dass diese identisch sind.

6.2 Verifikation der Daten des Interpretationsmodules

Zur Verifikation des Interpretationsmoduls wird ein Testmodul verwendet, welches das Datenempfangsmodul ersetzt und eine Serie von zuvor festgelegten „`SmplEyeTrackingData`“ Datensätzen dem Interpretationsmodul über den Shared-Memory zur Verfügung stellt. Anhand der vom Interpretationsmodul im Shared-Memoy abgelegten Datensäte „`SmplEyeTrackinIterprededData`“ können diese interpretierten Daten mit den erwarteten Ergebnissen für die Interpretation der Eingangsdaten vom Testmodul verglichen werden.

Ein Verifikation der erwarteten Ergebnisse mit den zur Laufzeit des Moduls erzeugten Daten ergab bislang einige kleinere Bugs, die sofort behoben werden konnten. Die Verifikation des Moduls ist noch nicht abgeschlossen und wird fortgesetzt.

6.3 Verifikation der visualiserten Daten

Die Verifiezierung der Klasse „`SmplEyeTracking3DVisualisation`“, welche das Modul „`SmplFSViewer`“ um die Darstellung der interpretierten Daten in einem 3D-Modell er-

weitert erfolgt visuell durch eine kritische Prüfung der dargestellten 3D-Objekte in dem 3D-Modell. Hierbei wird überprüft ob die 3D-Objekte bei bekannten Eingangsparametern an den erwarteten Positionen in der erwarteten Form im 3D-Modell visualisiert werden.

Eine grobe Überprüfung von Form und Positionierung der 3D-Objekte im Modell wird z.B. im modellierten SMPLab durch ein vergleichen der Größen- und Abstandsverhältnisse des realen SMPLabs mit dem virtuellen ermöglicht.

Für eine Überprüfung der Positionierung der visualiserten Blickpunkte und Fixationen eignen sich darzustellende Messpunkte, die sich direkt in einer Ecke der Planes oder AOIs befinden. Dort lassen sich Abweichungen von der erwarteten Position der Visualiserung dieses Messwertes am einfachsten erkennen.

Die visuelle Überprüfung des 3D-Modells ergab keine unerwarteten Formen oder Positionierungen der 3D-Objekte in der Visualiserung des SMPLabs und der interpretierten Daten.

6.4 Fehlerquellen

Bei einer Betrachtung der vollständigen „eye tracking„ Verarbeitungskette, von der Aufnahme des Auges durch ein Blickrichtungsmesssystem bis hin zur Visualisierung der Daten, ergeben sich einige mögliche Fehlerquellen. Diese Fehlerquellen können örtlich zu einer Abweichungen eines visualiserten Blickpunktes von dem tatsächlich betrachteten Punkt der Testperson führen. Hinzu kommen mögliche zeitliche Fehlerquellen, die zu einer Verzögerung der Visualiserung führen können. Die nachfolgenden Auflistungen beschreiben einige mögliche örtliche und zeitliche Fehlerquellen. Eine Untersuchung der Fehlerfortpflanzung über die „eye tracking“ Verarbeitungskette würde eine Ausage über die Auswirkung der verschiedenen Fehlerquellen auf die visualisierten Daten ermöglichen. Eine solche Untersuchung konnte innerhalb des zeitlichen Rahmens dieser Masterarbeit nicht durchgeführt werden.

6.4.1 Örtliche Fehlerquellen

Mögliche Fehlerquellen bei Verwendung des Blickrichtungsmesssystems iViewX HED/HT von SMI

1. Eine ungenaue Kalibrierung der Szenenkamera, der Augenkamera
2. Abweichungen durch Fehler beim Einmessen von SMI internen Planes
3. Störungen im Magnetfeld des HT Systems (z.B. Kopfstütze des Fahrersitzes)
4. Ein Verrücken oder Wackeln an der magnetfelderzeugenden Komponete des HT Systems zur Laufzeit eines Versuches
5. Toleranzen der Analysesoftware bei der Bestimmung der Augenbewegungen anhand des Kamerabildes und der HT-Daten
6. Fehlerhaft eingestellte Thresholds für die Bestimmung der Pupille und des corneal reflex im Bild der Augenkamera
7. Verlust von Datenpaketen bei dem Streamen der Messdaten über UDP

Mögliche Fehlerquellen bei der Verarbeitung der Messdaten durch die Module der Smpl++-Plattform

1. Fehlerhafter Empfang von Daten über UDP
2. Rundungsfehler beim Generalisieren des Datenformats im Datenerfassungsmodul
3. Rundungsfehler bei der Berechnung der interpretierten Daten durch das Interpretationsmodul
4. Ungenau eingemessene Planes und AOIs führen zu einer fehlerhaften Berechnung und Visualisierung der Augenbewegungsmessdaten
5. Verlust von Nachkommastellen beim Umkonvertieren der in double berechneten interpretierten Daten nach float für die Erzeugung der 3D-Objekte mit OSG
6. Ungenauigkeiten durch Fehler beim Rendern der 3D-Objekte durch OSG

6.4.2 Zeitliche Fehlerquellen

Fehlerquellen die zu einer zeitlichen Verzögerung der Visualisierung führen können

1. SMI interne Verzögerungen von der Aufnehme eines Bildes der Augenkamera über die Analyse dieses Bildes bis zum Streamen der berechneten Parameter
2. Dauer der Übertragung der Daten zwischen dem Blickrichtungsmesssystem und der Smpl++-Plattform mittels UDP
3. Verzögerungen durch die Laufzeiten des Datenverarbeitungs- und Interpretationsmoduls
4. Von jedem Schreiben der Daten in den Shared-Memory bis zur Entnahme dieser Daten durch ein Zielmodul versteicht eine unbekannte Zeitspanne
5. Das Rendern der 3D-Modelle durch OSG benötigt eine gewisse Zeit

Kapitel 7

Zusammenfassung

Das voran gegangene Kapitel beschreibt die Tests zur Verifikation der Daten die durch Module, entstanden während dieser Arbeit, erzeugt wurden.

Dieses Kapitel beschreibt die erreichten Ziele durch diese Masterarbeit, gefolgt von einem kurzer Ausblick auf mögliche Aufgaben zur Weiterentwicklung der bestehenden „SmplEyeTracking“ Module.

7.1 Erreichte Ziele

Mit Bezug auf die Aufgabenstellung / Zielsetzung (siehe Kapitel: 1) wurden die folgenden Ziele erreicht:

1. Für die Integration von Blickrichtungsmesssystemen in die Applikations- und Entwicklungsplattform Smpl++ wurde das Modul „`SmplEyeTrackingRemoteCommand`“ entwickelt (siehe Kapitel 5.1). Dieses Modul verfügt über eine für die meisten Blickrichtungsmesssysteme gültige Klasse „`SmplEyeTrackingDevice`“[1], die Grundfunktionen wie das Erstellen einer UDP-Verbindung zwischen dem Blickrichtungsmesssystem und dem Modul zur Verfügung stellt. Zusätzlich verfügt das Modul über eine spezifische Klasse für das Versenden und Empfangen von Nachrichten des Blickrichtungsmesssystems IViewX HED/HT von SMI. Die empfangenen Daten vom SMI-System werden durch die Datenklasse „`SmplEyeTrackingGlobals/SmplEyeTrackingData`“ über das „SSM“ allen aktiven Smpl++-Modulen einer Automation zur Verfügung gestellt.

2. Durch das Smpl++-Modul „`SmplEyeTrackingDataInterpretation`“ (siehe Kapitel: 5.2) wird eine Interpretation der empfangenen Messdaten ermöglicht. Anhand der durch das Messsystem ermittelten Augenposition und dem normierten Blickvektor kann dieses Modul zur Laufzeit des Datenempfangs aus zwei aufeinanderfolgenden Messdatensätzen bestimmen, ob eine vermutete Fixation, Fixation oder Sakkade vorliegt. Für eine genaue Analyse der Blickrichtung ist es möglich, besondere Bereiche (Planes und AOIs) über eine ASCII-Zeichen basierte Datei zu definieren. Das Interpretationsmodul ermittelt in jedem Zyklus neben Typ des Blicks, ob der Blickstrahl die Planes oder AOIs trifft. Aus dem Zusammenspiel von bestimmten Fixationen und den ermittelten Treffern in bestimmten Bereichen berechnet dieses Modul diverse Analyseparameter aus der angegebenen Literatur. Die Ergebnisse dieses Interpretationsmoduls werden über die Datenklasse „`SmplEyeTrackingGlobals/SmplEyeTrackingInterpretedData`“ und dem „SSM“ allen gleichzeitig laufenden Smpl++-Modulen einer Automation zur

[1] Die Klasse dieses Moduls ist für alle Blickrichtungsmesssysteme gültig, die über eine UDP Schnittstelle zum Streamen der gemessene Augenposition und des normierten Blickvektor verfügen

Verfügung gestellt

3. Für die Visualisierung der interpretierten Blickrichtungsmessdaten wurde das Modul „`SmplFSViewer`“ um die Klasse „`SmplEyeTracking3DVisualization`“ (siehe Kapitel: 5.3.3) erweitert. Diese Klasse ermöglicht durch die freie 3D-Bibliothek „Open Scene Graph“ die Darstellung eines 3D-Modells des SMPLabs, der definierten Planes und AOIs sowie der Blickrichtung und der vermuteten Fixationen, Fixationen und Sakkaden. Über das in Smpl++ bereits vorhandene Modul „`SmplcaSBAro`“ ist ein Aufzeichnen der Daten aus dem „SSM“ für spätere Replays möglich. Ausserdem ermöglicht „`SmplcaSBAro`“ die Visualisierung dieser Daten in Form von Diagrammen.

4. Die optionale Onlineanwendung wurde bisher noch nicht umgesetzt.

5. Dokumentiert wurden die entwickelten Smpl++-Module sowohl durch diesen schriftlichen Teil der Masterarbeit, als auch durch das freie Tool Doxygen.

Mit Abschluss dieser Arbeit verfügt die Applikations- Entwicklungsplattform Smpl++ über Klassenbibliotheken, welche eine Integration von Blickrichtungsmesssystemen ermöglichen. Diese Klassenbibliotheken erlauben eine Online-Erfassung, Online-Analyse und Online-Visualisierung der Augenbewegungsmessdaten.

7.2 Soll ⇔ Ist Vergleich der Zeitplanung

In den folgenden Abbildungen wird der in der Planungsphase entwickelte Soll-Zeitplan 7.1 dem kurz vor der Abgabe der Masterarbeit aktualisierten Ist-Zeitplan 7.2 gegenüber gestellt.

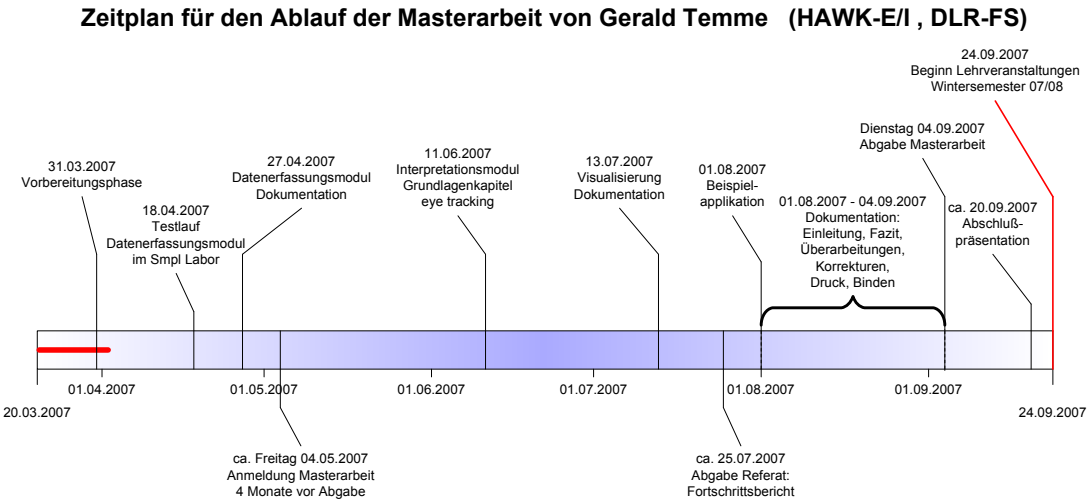


Abbildung 7.1: Soll-Zeitplan erstellt während der Planungsphase

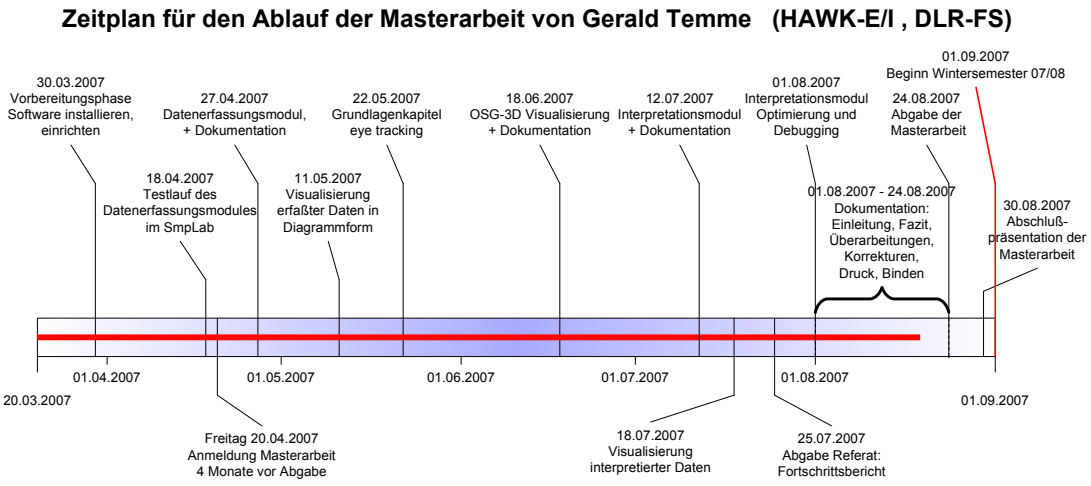


Abbildung 7.2: Ist-Zeitplan kurz vor Abgabe der Masterarbeit

Die beiden Abbildungen im Vergleich zeigen, dass es zu einigen Änderungen im zeitlichen Ablauf dieser Arbeit kam.

1. Der Zeitrahmen zur Durchführung dieser Arbeit wurde um 24 Tage verkürzt, da die Arbeit zum Beginn des Wintersemesters 07/08 abgegeben werden muss und nicht wie in der Planungsphase angenommen zu Beginn der Lehrveranstaltungen des Wintersemesters 07/08, um ein Bezahlen von Studien- und Semestergebühren (zusammen ca. 700 €) zu entgehen. Diese Verkürzung des gesamt Zeitrahmens findet sich in dem um 11 Tage reduzierten Bereich für die Dokumentation (geschweifte Klammer) sowie in der Vorbereitungzeit des Kolloquiums wieder.

2. Nach der planungsgemäß abgeschlossenen Entwicklung des Datenerfassungsmodules (27.04) kam es zu einer Umstellung in dem geplannten Ablauf zur Realisierung der Meilensteine. Im Zuge dieser Umstellung wurde die Entwicklung der Visualisierung vor der Umsetzung des Interpretationsmodules durchgeführt. Für diese Umstellung der Meilensteine gab es verschiedene Gründe:

   - Die vorgezogene Umsetzung der Visualisierung der gemessenen Augenbewegungsdaten als Diagramme ermöglichten eine vereinfachte Verifizierung der erhaltenen Augenbewegungsmessdaten (11.05).
   - Es stellte sich heraus, dass für eine Analyse der Blickrichtung die durch das Blickrichtungsmesssystem zur Verfügung gestellte Funktionalität zur Definition von Planes und AOIs nicht ausreichen, da das Blickrichtungsmesssystem für Blickpunkte ausserhalb aller definierten Planes keine brauchbaren Parameter zur Verfügung stellt. Aus diesem Grund verwendet das Interpretationsmodul nur die beiden Grundparameter der Blickrichtungsmesssysteme (Augenposition und normierter Blickvektor), welche eine von der Analysesoftware des Messsystems unabhängige Bestimmung der Schnittpunkte mit zuvor definierten Planes und AOIs (bzw. mit einer Kugel wenn keine Schnittpunkt mit einer Plane vorhanden ist) ermöglicht. Durch einen Fehler in der Analysesoftware von SMI wurde der normierte Blickvektor nicht über UDP an die Smpl++-Plattform mitübertragen. Da dieser Wert für eine Interpre-

tation der Augenbewegungen unverzichtbar ist, konnte die Entwicklung des Interpretationsmodul erst nach einem Update der Software von SMI effektiv durchgeführt werden (ab 12.07).

- Die Visualisierung der Blickrichtung, von Fixationen und Sakkaden wurde in einem während der Planungphase noch nicht vorgesehenen 3D-Modell mit OSG realisiert. Dadurch entstand ein zeitlicher Mehraufwand zur Einarbeitung in die freie 3D-Bibliothek OSG von ca. acht Tagen.

3. Durch die gestigene Komplexität des Interpretationsmodules wurde an diesem, auch nach der eigentlichen Fertigstellung des Modules am 12.07, noch bis zum 01.08 viel Zeit für eine Optimierung und ein Debugging des Quellcodes verwendet. Daher wurde die geplante Beispielapplikation bis heute (19.08) noch nicht realisiert.

4. Und wie fast immer bei Master- und Diplomarbeiten, wurde leider auch hier der zeitliche Aufwand für das Schreiben und Korrigieren der Dokumentation in der Planungsphase unterschätzt.

7.3 Ausblicke

Der nachfolgende Abschnitt beschreibt kurz mögliche Aufgaben zur Weiterentwicklung der entwickelten Smpl++ Module:

1. Das Modul „`SmplEyeTrackingRemoteCommand`“ sollte um eine spezielle Klasse für das zweite am DLR-Standort Braunschweig vertretene Blickrichtungsmessgerät von faceLAB^{tm} erweitert werden.

2. Dem Modul „`SmplEyeTrackingDataInterpretation`“ könnten weitere Berechnungen von Analyseparametern (z.B. Sakkadendauer) hinzugefügt werden.

3. Auch könnte dem Modul „`SmplEyeTrackingDataInterpretation`“ die Trefferbestimmung mit zusätzlichen AOI-Formen (z.B. Kreise oder Dreiecke) ermöglicht werden.

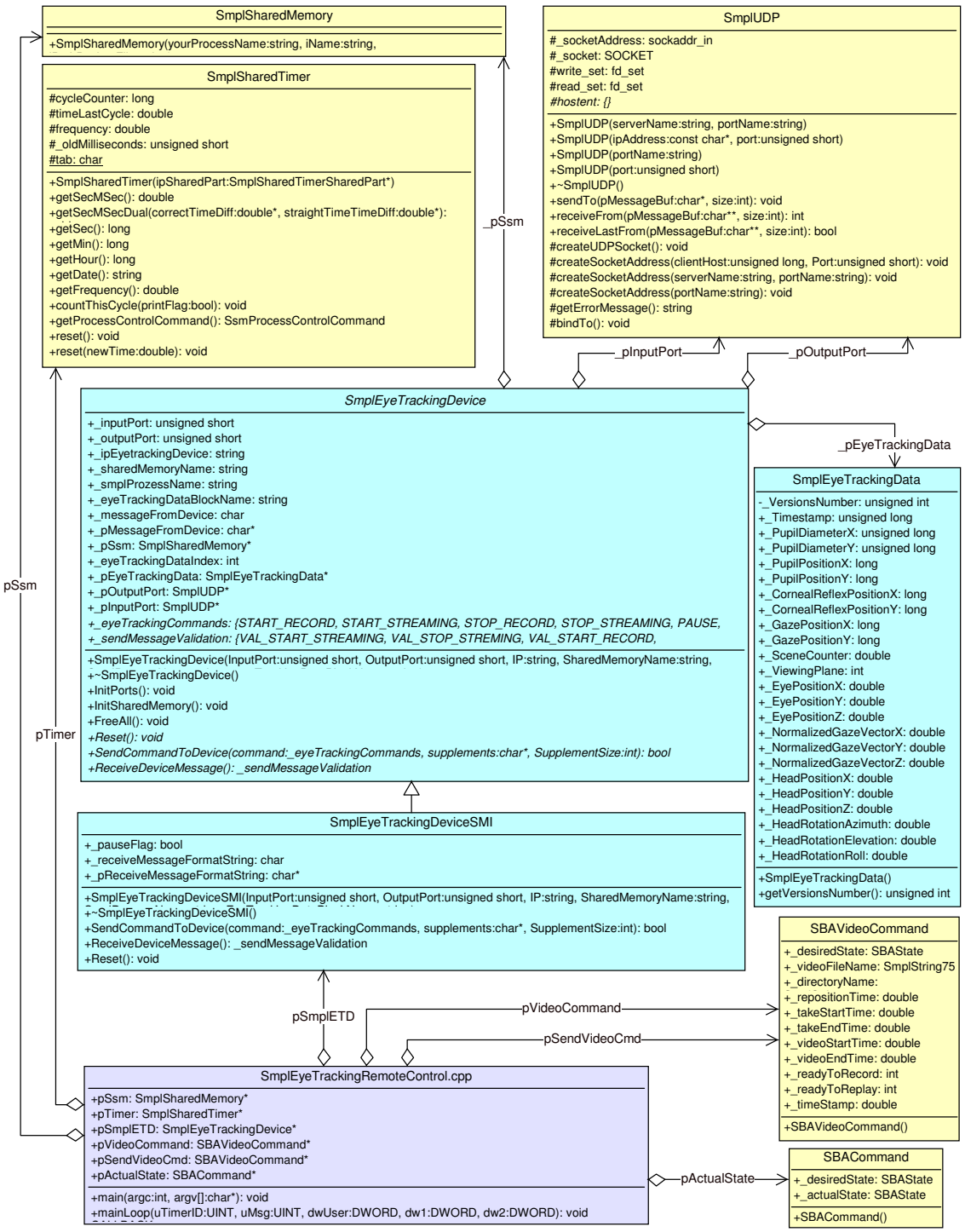
4. Für ein Visualisieren der zusätzlichen AOI-Formen müsste auch die Visualisierungsklasse „`SmplEyeTracking3DVisualization`“ entsprechend erweitert werden.

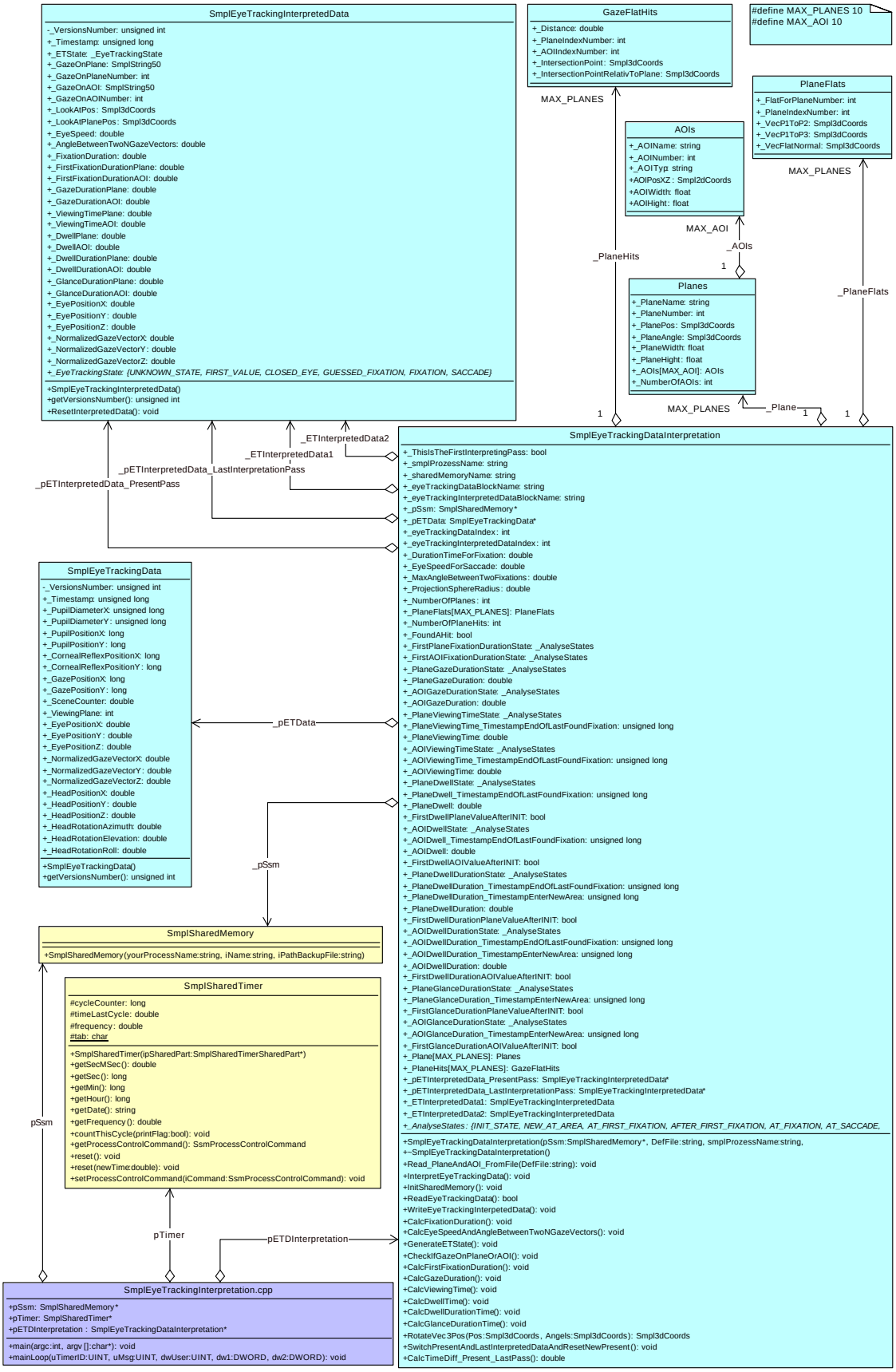
5. Für eine genaue Verifizierung der interpretieren und visualisierten Blickrichtungsmessdaten sollte die Fehlerfortpflanzung dieser Daten vom Blickrichtungsmesssystem über das Datenerfassungsmodul und dem Interpretationmodul bis zur Visualisierung in dem 3D-Modell mit OSG untersucht werden.

6. Der erste Einsatz der Module zur Analyse eines Versuchsaufbaus als wichtigen Praxistest, steht ebenfalls noch aus.

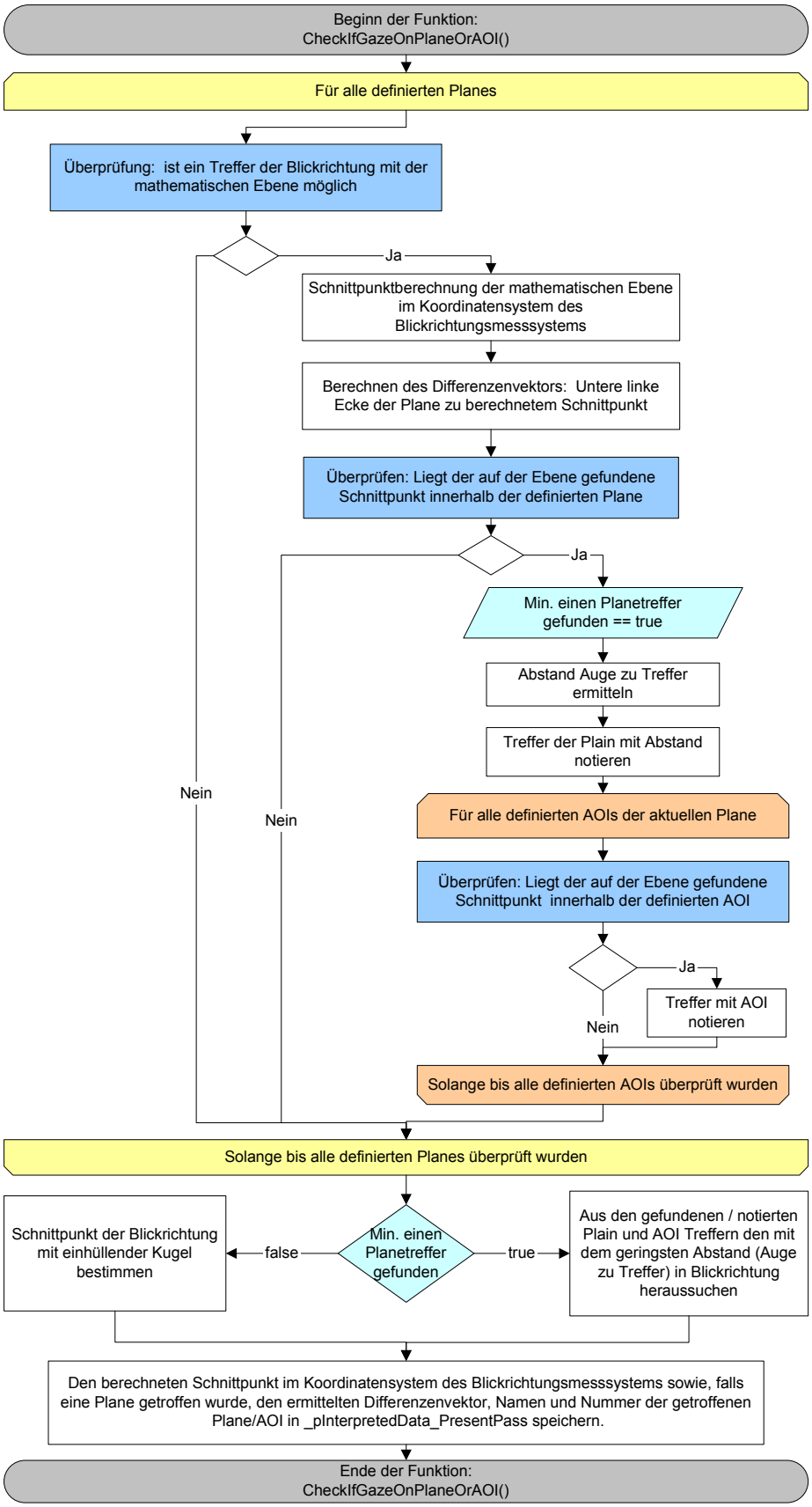
Kapitel 8

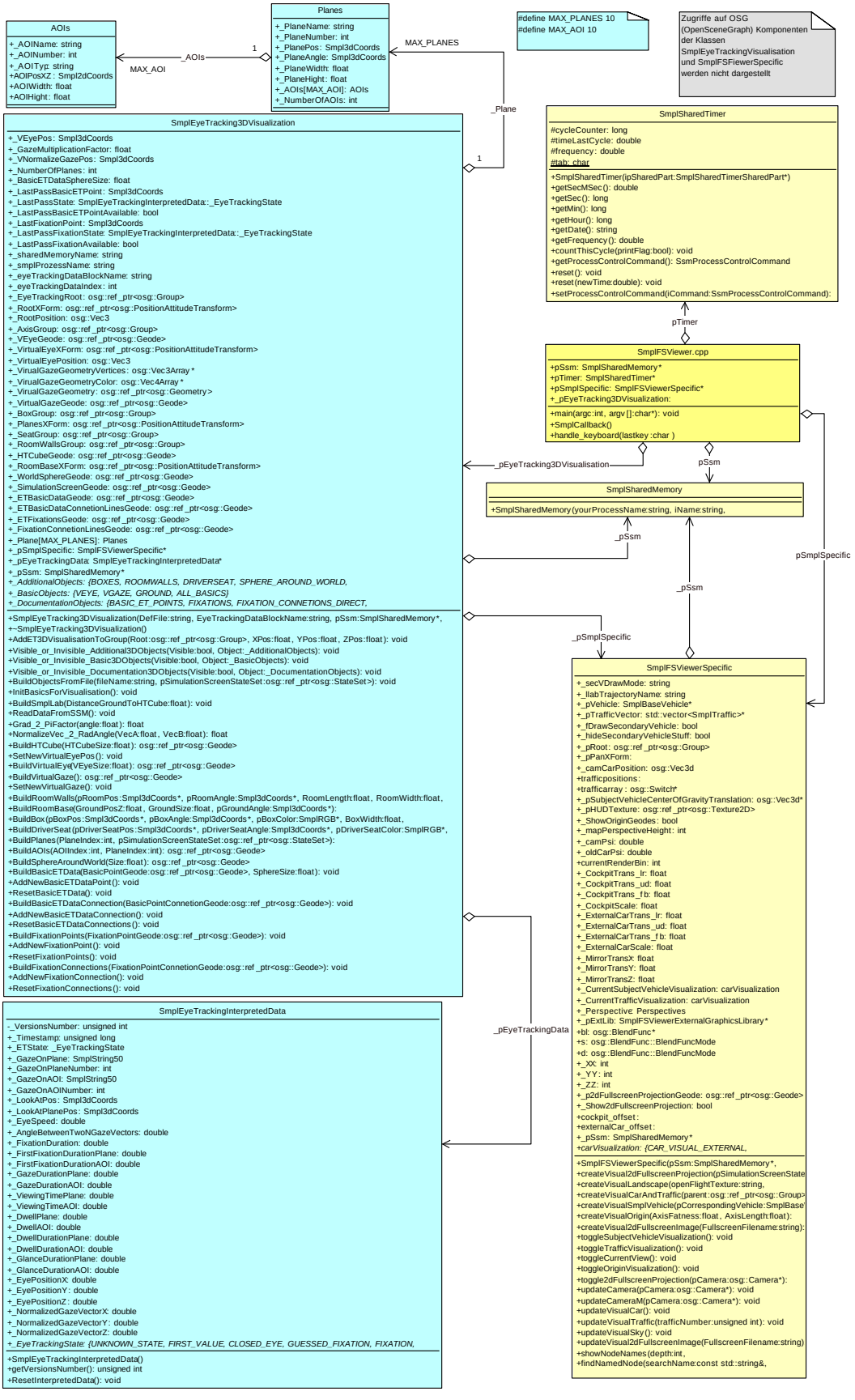
Anhang

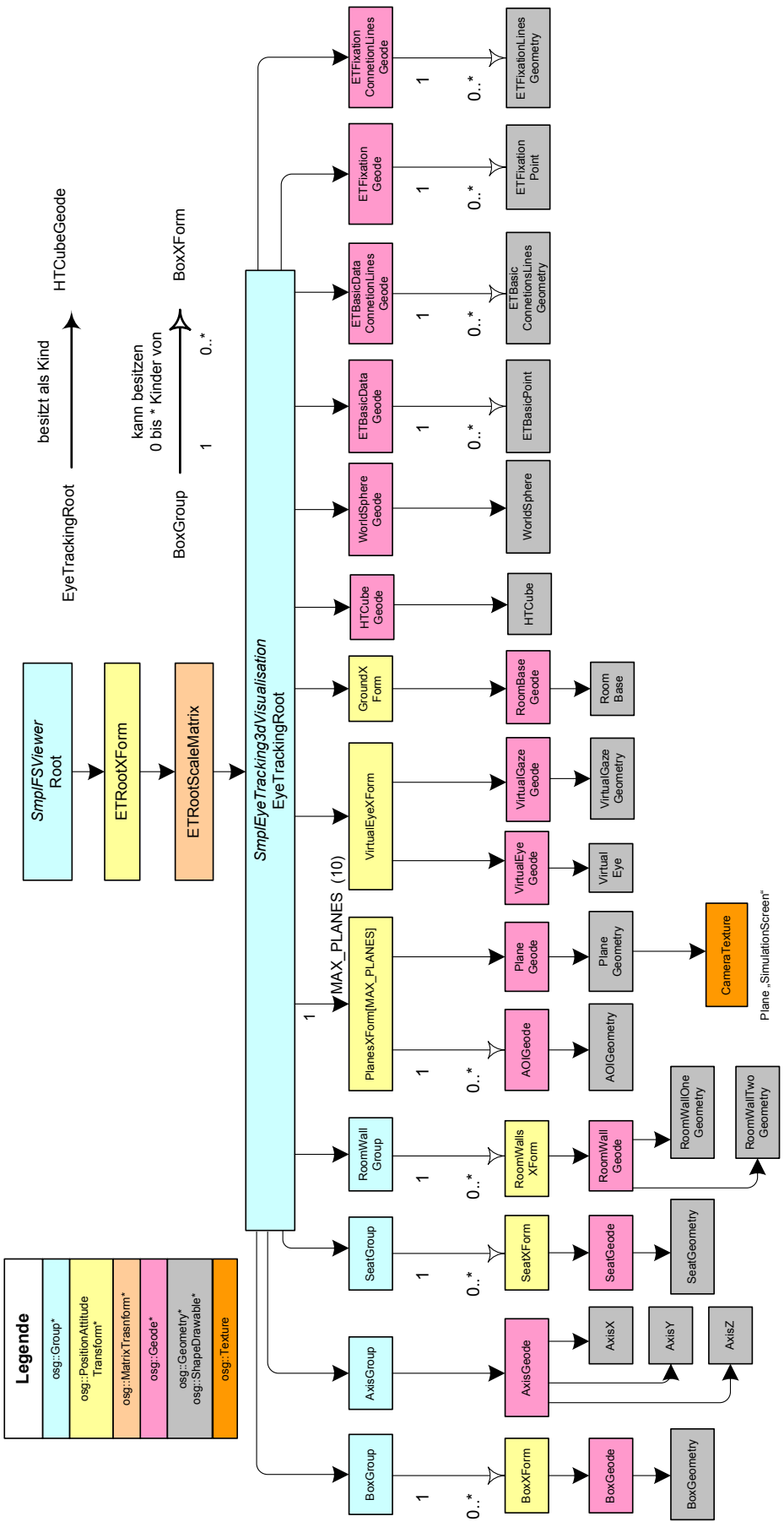

SmplSharedMemory
+SmplSharedMemory(yourProcessName:string, iName:string,
SmplSharedTimer
#cycleCounter: long
#timeLastCycle: double
#frequency: double
#_oldMilliseconds: unsigned short
#tab: char
+SmplSharedTimer(ipSharedPart:SmplSharedTimerSharedPart*)
+getSecMSec(): double
+getSecMSecDual(correctTimeDiff:double*, straightTimeTimeDiff:double*):
+getSec(): long
+getMin(): long
+getHour(): long
+getDate(): string
+getFrequency(): double
+countThisCycle(printFlag:bool): void
+getProcessControlCommand(): SsmProcessControlCommand
+reset(): void
+reset(newTime:double): void
_pSsm
SmplUDP
#_socketAddress: sockaddr_in
#_socket: SOCKET
#write_set: fd_set
#read_set: fd_set
#hostent: {}
+SmplUDP(serverName:string, portName:string)
+SmplUDP(ipAddress:const char*, port:unsigned short)
+SmplUDP(portName:string)
+SmplUDP(port:unsigned short)
+~SmplUDP()
+sendTo(pMessageBuf:char*, size:int): void
+receiveFrom(pMessageBuf:char**, size:int): int
+receiveLastFrom(pMessageBuf:char**, size:int): bool
#createUDPSocket(): void
#createSocketAddress(clientHost:unsigned long, Port:unsigned short): void
#createSocketAddress(serverName:string, portName:string): void
#createSocketAddress(portName:string): void
#getErrorMessage(): string
#bindTo(): void
_pInputPort
_pOutputPort
SmplEyeTrackingDevice
+_inputPort: unsigned short
+_outputPort: unsigned short
+_ipEyetrackingDevice: string
+_sharedMemoryName: string
+_smplProzessName: string
+_eyeTrackingDataBlockName: string
+_messageFromDevice: char
+_pMessageFromDevice: char*
+_pSsm: SmplSharedMemory*
+_eyeTrackingDataIndex: int
+_pEyeTrackingData: SmplEyeTrackingData*
+_pOutputPort: SmplUDP*
+_pInputPort: SmplUDP*
+_eyeTrackingCommands: {START_RECORD, START_STREAMING, STOP_RECORD, STOP_STREAMING, PAUSE,
+_sendMessageValidation: {VAL_START_STREAMING, VAL_STOP_STREMING, VAL_START_RECORD,
+SmplEyeTrackingDevice(InputPort:unsigned short, OutputPort:unsigned short, IP:string, SharedMemoryName:string,
+~SmplEyeTrackingDevice()
+InitPorts(): void
+InitSharedMemory(): void
+FreeAll(): void
+Reset(): void
+SendCommandToDevice(command:_eyeTrackingCommands, supplements:char*, SupplementSize:int): bool
+ReceiveDeviceMessage(): _sendMessageValidation
_pEyeTrackingData
SmplEyeTrackingData
-_VersionsNumber: unsigned int
+_Timestamp: unsigned long
+_PupilDiameterX: unsigned long
+_PupilDiameterY: unsigned long
+_PupilPositionX: long
+_PupilPositionY: long
+_CornealReflexPositionX: long
+_CornealReflexPositionY: long
+_GazePositionX: long
+_GazePositionY: long
+_SceneCounter: double
+_ViewingPlane: int
+_EyePositionX: double
+_EyePositionY: double
+_EyePositionZ: double
+_NormalizedGazeVectorX: double
+_NormalizedGazeVectorY: double
+_NormalizedGazeVectorZ: double
+_HeadPositionX: double
+_HeadPositionY: double
+_HeadPositionZ: double
+_HeadRotationAzimuth: double
+_HeadRotationElevation: double
+_HeadRotationRoll: double
+SmplEyeTrackingData()
+getVersionsNumber(): unsigned int
pSsm
pTimer
SmplEyeTrackingDeviceSMI
+_pauseFlag: bool
+_receiveMessageFormatString: char
+_pReceiveMessageFormatString: char*
+SmplEyeTrackingDeviceSMI(InputPort:unsigned short, OutputPort:unsigned short, IP:string, SharedMemoryName:string,
+~SmplEyeTrackingDeviceSMI()
+SendCommandToDevice(command:_eyeTrackingCommands, supplements:char*, SupplementSize:int): bool
+ReceiveDeviceMessage(): _sendMessageValidation
+Reset(): void
SBAVideoCommand
+_desiredState: SBAState
+_videoFileName: SmplString75
+_directoryName:
+_repositionTime: double
+_takeStartTime: double
+_takeEndTime: double
+_videoStartTime: double
+_videoEndTime: double
+_readyToRecord: int
+_readyToReplay: int
+_timeStamp: double
+SBAVideoCommand()
pSmplETD
pVideoCommand
pSendVideoCmd
SmplEyeTrackingRemoteControl.cpp
+pSsm: SmplSharedMemory*
+pTimer: SmplSharedTimer*
+pSmplETD: SmplEyeTrackingDevice*
+pVideoCommand: SBAVideoCommand*
+pSendVideoCmd: SBAVideoCommand*
+pActualState: SBACommand*
+main(argc:int, argv[]:char*): void
+mainLoop(uTimerID:UINT, uMsg:UINT, dwUser:DWORD, dw1:DWORD, dw2:DWORD): void
pActualState
SBACommand
+_desiredState: SBAState
+_actualState: SBAState
+SBACommand()


SmplEyeTrackingInterpretedData
-_VersionsNumber: unsigned int
+_Timestamp: unsigned long
+_ETState: _EyeTrackingState
+_GazeOnPlane: SmplString50
+_GazeOnPlaneNumber: int
+_GazeOnAOI: SmplString50
+_GazeOnAOINumber: int
+_LookAtPos: Smpl3dCoords
+_LookAtPlanePos: Smpl3dCoords
+_EyeSpeed: double
+_AngleBetweenTwoNGazeVectors: double
+_FixationDuration: double
+_FirstFixationDurationPlane: double
+_FirstFixationDurationAOI: double
+_GazeDurationPlane: double
+_GazeDurationAOI: double
+_ViewingTimePlane: double
+_ViewingTimeAOI: double
+_DwellPlane: double
+_DwellAOI: double
+_DwellDurationPlane: double
+_DwellDurationAOI: double
+_GlanceDurationPlane: double
+_GlanceDurationAOI: double
+_EyePositionX: double
+_EyePositionY: double
+_EyePositionZ: double
+_NormalizedGazeVectorX: double
+_NormalizedGazeVectorY: double
+_NormalizedGazeVectorZ: double
+_EyeTrackingState: {UNKNOWN_STATE, FIRST_VALUE, CLOSED_EYE, GUESSED_FIXATION, FIXATION, SACCADE}
+SmplEyeTrackingInterpretedData()
+getVersionsNumber(): unsigned int
+ResetInterpretedData(): void
GazeFlatHits
+_Distance: double
+_PlaneIndexNumber: int
+_AOIIndexNumber: int
+_IntersectionPoint: Smpl3dCoords
+_IntersectionPointRelativToPlane: Smpl3dCoords
#define MAX_PLANES 10
#define MAX_AOI 10
PlaneFlats
+_FlatForPlaneNumber: int
+_PlaneIndexNumber: int
+_VecP1ToP2: Smpl3dCoords
+_VecP1ToP3: Smpl3dCoords
+_VecFlatNormal: Smpl3dCoords
AOIs
+_AOIName: string
+_AOINumber: int
+_AOIType: string
+AOIPosXZ: Smpl2dCoords
+AOIWidth: float
+AOIHight: float
Planes
+_PlaneName: string
+_PlaneNumber: int
+_PlanePos: Smpl3dCoords
+_PlaneAngle: Smpl3dCoords
+_PlaneWidth: float
+_PlaneHight: float
+_AOIs[MAX_AOI]: AOIs
+_NumberOfAOIs: int
MAX_PLANES
MAX_AOI
_PlaneHits
_AOIs
_PlaneFlats
_Plane
1
SmplEyeTrackingDataInterpretation
+_ThisIsTheFirstInterpretingPass: bool
+_smplProzessName: string
+_sharedMemoryName: string
+_eyeTrackingDataBlockName: string
+_eyeTrackingInterpretedDataBlockName: string
+_pSsm: SmplSharedMemory*
+_pETData: SmplEyeTrackingData*
+_eyeTrackingDataIndex: int
+_eyeTrackingInterpretedDataIndex: int
+_DurationTimeForFixation: double
+_EyeSpeedForSaccade: double
+_MaxAngleBetweenTwoFixations: double
+_ProjectionSphereRadius: double
+_NumberOfPlanes: int
+_PlaneFlats[MAX_PLANES]: PlaneFlats
+_NumberOfPlaneHits: int
+_FoundAHit: bool
+_FirstPlaneFixationDurationState: _AnalyseStates
+_FirstAOIFixationDurationState: _AnalyseStates
+_PlaneGazeDurationState: _AnalyseStates
+_PlaneGazeDuration: double
+_AOIGazeDurationState: _AnalyseStates
+_AOIGazeDuration: double
+_PlaneViewingTimeState: _AnalyseStates
+_PlaneViewingTime_TimestampEndOfLastFoundFixation: unsigned long
+_PlaneViewingTime: double
+_AOIViewingTimeState: _AnalyseStates
+_AOIViewingTime_TimestampEndOfLastFoundFixation: unsigned long
+_AOIViewingTime: double
+_PlaneDwellState: _AnalyseStates
+_PlaneDwell_TimestampEndOfLastFoundFixation: unsigned long
+_PlaneDwell: double
+_FirstDwellPlaneValueAfterINIT: bool
+_AOIDwellState: _AnalyseStates
+_AOIDwell_TimestampEndOfLastFoundFixation: unsigned long
+_AOIDwell: double
+_FirstDwellAOIValueAfterINIT: bool
+_PlaneDwellDurationState: _AnalyseStates
+_PlaneDwellDuration_TimestampEndOfLastFoundFixation: unsigned long
+_PlaneDwellDuration_TimestampEnterNewArea: unsigned long
+_PlaneDwellDuration: double
+_FirstDwellDurationPlaneValueAfterINIT: bool
+_AOIDwellDurationState: _AnalyseStates
+_AOIDwellDuration_TimestampEndOfLastFoundFixation: unsigned long
+_AOIDwellDuration_TimestampEnterNewArea: unsigned long
+_AOIDwellDuration: double
+_FirstDwellDurationAOIValueAfterINIT: bool
+_PlaneGlanceDurationState: _AnalyseStates
+_PlaneGlanceDuration_TimestampEnterNewArea: unsigned long
+_FirstGlanceDurationPlaneValueAfterINIT: bool
+_AOIGlanceDurationState: _AnalyseStates
+_AOIGlanceDuration_TimestampEnterNewArea: unsigned long
+_FirstGlanceDurationAOIValueAfterINIT: bool
+_Plane[MAX_PLANES]: Planes
+_PlaneHits[MAX_PLANES]: GazeFlatHits
+_pETInterpretedData_PresentPass: SmplEyeTrackingInterpretedData*
+_pETInterpretedData_LastInterpretationPass: SmplEyeTrackingInterpretedData*
+_ETInterpretedData1: SmplEyeTrackingInterpretedData
+_ETInterpretedData2: SmplEyeTrackingInterpretedData
+_AnalyseStates: {INIT_STATE, NEW_AT_AREA, AT_FIRST_FIXATION, AFTER_FIRST_FIXATION, AT_FIXATION, AT_SACCADE,
+SmplEyeTrackingDataInterpretation(pSsm:SmplSharedMemory*, DefFile:string, smplProzessName:string,
+~SmplEyeTrackingDataInterpretation()
+Read_PlaneAndAOI_FromFile(DefFile:string): void
+InterpretEyeTrackingData(): void
+InitSharedMemory(): void
+ReadEyeTrackingData(): bool
+WriteEyeTrackingInterpetedData(): void
+CalcFixationDuration(): void
+CalcEyeSpeedAndAngleBetweenTwoNGazeVectors(): void
+GenerateETState(): void
+CheckIfGazeOnPlaneOrAOI(): void
+CalcFirstFixationDuration(): void
+CalcGazeDuration(): void
+CalcViewingTime(): void
+CalcDwellTime(): void
+CalcDwellDurationTime(): void
+CalcGlanceDurationTime(): void
+RotateVec3Pos(Pos:Smpl3dCoords, Angels:Smpl3dCoords): Smpl3dCoords
+SwitchPresentAndLastInterpretedDataAndResetNewPresent(): void
+CalcTimeDiff_Present_LastPass(): double
_ETInterpretedData2
_ETInterpretedData1
_pETInterpretedData_LastInterpretationPass
_pETInterpretedData_PresentPass
SmplEyeTrackingData
-_VersionsNumber: unsigned int
+_Timestamp: unsigned long
+_PupilDiameterX: unsigned long
+_PupilDiameterY: unsigned long
+_PupilPositionX: long
+_PupilPositionY: long
+_CornealReflexPositionX: long
+_CornealReflexPositionY: long
+_GazePositionX: long
+_GazePositionY: long
+_SceneCounter: double
+_ViewingPlane: int
+_EyePositionX: double
+_EyePositionY: double
+_EyePositionZ: double
+_NormalizedGazeVectorX: double
+_NormalizedGazeVectorY: double
+_NormalizedGazeVectorZ: double
+_HeadPositionX: double
+_HeadPositionY: double
+_HeadPositionZ: double
+_HeadRotationAzimuth: double
+_HeadRotationElevation: double
+_HeadRotationRoll: double
+SmplEyeTrackingData()
+getVersionsNumber(): unsigned int
_pETData
_pSsm
SmplSharedMemory
+SmplSharedMemory(yourProcessName:string, iName:string, iPathBackupFile:string)
SmplSharedTimer
#cycleCounter: long
#timeLastCycle: double
#frequency: double
#tab: char
+SmplSharedTimer(ipSharedPart:SmplSharedTimerSharedPart*)
+getSecMSec(): double
+getSec(): long
+getMin(): long
+getHour(): long
+getDate(): string
+getFrequency(): double
+countThisCycle(printFlag:bool): void
+getProcessControlCommand(): SsmProcessControlCommand
+reset(): void
+reset(newTime:double): void
+setProcessControlCommand(iCommand:SsmProcessControlCommand): void
pSsm
pTimer
pETDInterpretation
SmplEyeTrackingInterpretation.cpp
+pSsm: SmplSharedMemory*
+pTimer: SmplSharedTimer*
+pETDInterpretation : SmplEyeTrackingDataInterpretation*
+main(argc:int, argv []:char*): void
+mainLoop(uTimerID:UINT, uMsg:UINT, dwUser:UINT, dw1:DWORD, dw2:DWORD): void

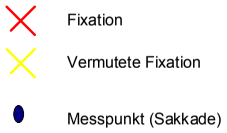
Standardflussdiagramm der Funktion „CheckIfGazeOnPlaneOrAOI()“ aus der Klasse „SmplEyeTrackingDataInterpretation“

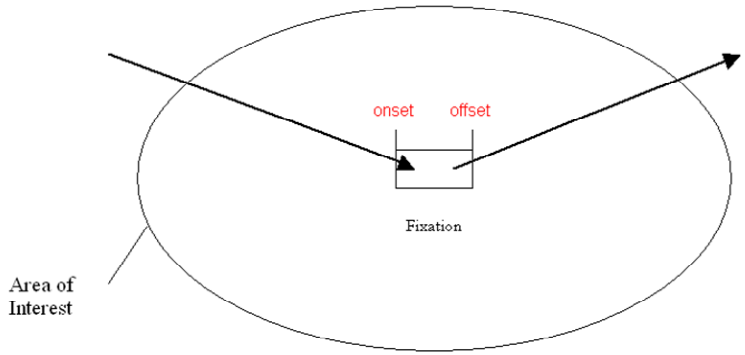



AOIs
+_AOIName: string
+_AOINumber: int
+_AOIType: string
+AOIPosXZ : Smpl2dCoords
+AOIWidth: float
+AOIHight: float
_AOIs
MAX_AOI
1
Planes
+_PlaneName: string
+_PlaneNumber: int
+_PlanePos: Smpl3dCoords
+_PlaneAngle: Smpl3dCoords
+_PlaneWidth: float
+_PlaneHight: float
+_AOIs[MAX_AOI]: AOIs
+_NumberOfAOIs: int
MAX_PLANES
#define MAX_PLANES 10
#define MAX_AOI 10
Zugriffe auf OSG (OpenSceneGraph) Komponenten der Klassen SmplEyeTrackingVisualisation und SmplFSFiewerSpecific werden nicht dargestellt
_Plane
1
SmplEyeTracking3DVisualization
+_VEyePos: Smpl3dCoords
+_GazeMultiplicationFactor: float
+_VNormalizeGazePos: Smpl3dCoords
+_NumberOfPlanes: int
+_BasicETDataSphereSize: float
+_LastPassBasicETPoint: Smpl3dCoords
+_LastPassState: SmplEyeTrackingInterpretedData::_EyeTrackingState
+_LastPassBasicETPointAvailable: bool
+_LastFixationPoint: Smpl3dCoords
+_LastPassFixationState: SmplEyeTrackingInterpretedData::_EyeTrackingState
+_LastPassFixationAvailable: bool
+_sharedMemoryName: string
+_smplProzessName: string
+_eyeTrackingDataBlockName: string
+_eyeTrackingDataIndex: int
+_EyeTrackingRoot: osg::ref_ptr<osg::Group>
+_RootXForm: osg::ref_ptr<osg::PositionAttitudeTransform>
+_RootPosition: osg::Vec3
+_AxisGroup: osg::ref_ptr<osg::Group>
+_VEyeGeode: osg::ref_ptr<osg::Geode>
+_VirtualEyeXForm: osg::ref_ptr<osg::PositionAttitudeTransform>
+_VirtualEyePosition: osg::Vec3
+_VirualGazeGeometryVertices: osg::Vec3Array *
+_VirualGazeGeometryColor: osg::Vec4Array *
+_VirualGazeGeometry: osg::ref_ptr<osg::Geometry>
+_VirtualGazeGeode: osg::ref_ptr<osg::Geode>
+_BoxGroup: osg::ref_ptr<osg::Group>
+_PlanesXForm: osg::ref_ptr<osg::PositionAttitudeTransform>
+_SeatGroup: osg::ref_ptr<osg::Group>
+_RoomWallsGroup: osg::ref_ptr<osg::Group>
+_HTCubeGeode: osg::ref_ptr<osg::Geode>
+_RoomBaseXForm: osg::ref_ptr<osg::PositionAttitudeTransform>
+_WorldSphereGeode: osg::ref_ptr<osg::Geode>
+_SimulationScreenGeode: osg::ref_ptr<osg::Geode>
+_ETBasicDataGeode: osg::ref_ptr<osg::Geode>
+_ETBasicDataConnetionLinesGeode: osg::ref_ptr<osg::Geode>
+_ETFixationsGeode: osg::ref_ptr<osg::Geode>
+_FixationConnetionLinesGeode: osg::ref_ptr<osg::Geode>
+_Plane[MAX_PLANES]: Planes
+_pSmplSpecific: SmplFSViewerSpecific*
+_pEyeTrackingData: SmplEyeTrackingInterpretedData*
+_pSsm: SmplSharedMemory*
+_AdditionalObjects: {BOXES, ROOMWALLS, DRIVERSEAT, SPHERE_AROUND_WORLD,
+_BasicObjects: {VEYE, VGAZE, GROUND, ALL_BASICS}
+_DocumentationObjects: {BASIC_ET_POINTS, FIXATIONS, FIXATION_CONNETIONS_DIRECT,
+SmplEyeTracking3DVisualization(DefFile:string, EyeTrackingDataBlockName:string, pSsm:SmplSharedMemory*,
+~SmplEyeTracking3DVisualization()
+AddET3DVisualisationToGroup(Root:osg::ref_ptr<osg::Group>, XPos:float, YPos:float, ZPos:float): void
+Visible_or_Invisible_Additional3DObjects(Visible:bool, Object:_AdditionalObjects): void
+Visible_or_Invisible_Basic3DObjects(Visible:bool, Object:_BasicObjects): void
+Visible_or_Invisible_Documentation3DObjects(Visible:bool, Object:_DocumentationObjects): void
+BuildObjectsFromFile(fileName:string, pSimulationScreenStateSet:osg::ref_ptr<osg::StateSet>): void
+InitBasicsForVisualisation(): void
+BuildSmplLab(DistanceGroundToHTCube:float): void
+ReadDataFromSSM(): void
+Grad_2_PiFactor(angle:float): float
+NormalizeVec_2_RadAngle(VecA:float, VecB:float): float
+BuildHTCube(HTCubeSize:float): osg::ref_ptr<osg::Geode>
+SetNewVirtualEyePos(): void
+BuildVirtualEye(VEyeSize:float): osg::ref_ptr<osg::Geode>
+BuildVirtualGaze(): osg::ref_ptr<osg::Geode>
+SetNewVirtualGaze(): void
+BuildRoomWalls(pRoomPos:Smpl3dCoords*, pRoomAngle:Smpl3dCoords*, RoomLength:float, RoomWidth:float,
+BuildRoomBase(GroundPosZ:float, GroundSize:float, pGroundAngle:Smpl3dCoords*):
+BuildBox(pBoxPos:Smpl3dCoords*, pBoxAngle:Smpl3dCoords*, pBoxColor:SmplRGB*, BoxWidth:float,
+BuildDriverSeat(pDriverSeatPos:Smpl3dCoords*, pDriverSeatAngle:Smpl3dCoords*, pDriverSeatColor:SmplRGB*,
+BuildPlanes(PlaneIndex:int, pSimulationScreenStateSet:osg::ref_ptr<osg::StateSet>):
+BuildAOIs(AOIIndex:int, PlaneIndex:int): osg::ref_ptr<osg::Geode>
+BuildSphereAroundWorld(Size:float): osg::ref_ptr<osg::Geode>
+BuildBasicETData(BasicPointGeode:osg::ref_ptr<osg::Geode>, SphereSize:float): void
+AddNewBasicETDataPoint(): void
+ResetBasicETData(): void
+BuildBasicETDataConnection(BasicPointConnetionGeode:osg::ref_ptr<osg::Geode>): void
+AddNewBasicETDataConnection(): void
+ResetBasicETDataConnections(): void
+BuildFixationPoints(FixationPointGeode:osg::ref_ptr<osg::Geode>): void
+AddNewFixationPoint(): void
+ResetFixationPoints(): void
+BuildFixationConnections(FixationPointConnetionGeode:osg::ref_ptr<osg::Geode>): void
+AddNewFixationConnection(): void
+ResetFixationConnections(): void
SmplSharedTimer
#cycleCounter: long
#timeLastCycle: double
#frequency: double
#tab: char
+SmplSharedTimer(ipSharedPart:SmplSharedTimerSharedPart*)
+getSecMSec(): double
+getSec(): long
+getMin(): long
+getHour(): long
+getDate(): string
+getFrequency(): double
+countThisCycle(printFlag:bool): void
+getProcessControlCommand(): SsmProcessControlCommand
+reset(): void
+reset(newTime:double): void
+setProcessControlCommand(iCommand:SsmProcessControlCommand):
pTimer
SmplFSViewer.cpp
+pSsm: SmplSharedMemory*
+pTimer: SmplSharedTimer*
+pSmplSpecific: SmplFSViewerSpecific*
+_pEyeTracking3DVisualization:
+main(argc:int, argv []:char*): void
+SmplCallback()
+handle_keyboard(lastkey :char)
_pEyeTracking3DVisualisation
pSsm
SmplSharedMemory
+SmplSharedMemory(yourProcessName:string, iName:string,
_pSsm
pSmplSpecific
_pSsm
_pSmplSpecific
SmplFSViewerSpecific
+_secVDrawMode: string
+_llabTrajectoryName: string
+_pVehicle: SmplBaseVehicle*
+_pTrafficVector: std::vector<SmplTraffic>*
+_fDrawSecondaryVehicle: bool
+_hideSecondaryVehicleStuff: bool
+_pRoot: osg::ref_ptr<osg::Group>
+_pPanXForm:
+_camCarPosition: osg::Vec3d
+trafficpositions:
+trafficarray : osg::Switch*
+_pSubjectVehicleCenterOfGravityTranslation: osg::Vec3d*
+_pHUDTexture: osg::ref_ptr<osg::Texture2D>
+_ShowOriginGeodes: bool
+_mapPerspectiveHeight: int
+_camPsi: double
+_oldCarPsi: double
+currentRenderBin: int
+_CockpitTrans_lr: float
+_CockpitTrans_ud: float
+_CockpitTrans_fb: float
+_CockpitScale: float
+_ExternalCarTrans_lr: float
+_ExternalCarTrans_ud: float
+_ExternalCarTrans_fb: float
+_ExternalCarScale: float
+_MirrorTransX: float
+_MirrorTransY: float
+_MirrorTransZ: float
+_CurrentSubjectVehicleVisualization: carVisualization
+_CurrentTrafficVisualization: carVisualization
+_Perspective: Perspectives
+_pExtLib: SmplFSViewerExternalGraphicsLibrary*
+bl: osg::BlendFunc*
+s: osg::BlendFunc::BlendFuncMode
+d: osg::BlendFunc::BlendFuncMode
+_XX: int
+_YY: int
+_ZZ: int
+_p2dFullscreenProjectionGeode: osg::ref_ptr<osg::Geode>
+_Show2dFullscreenProjection: bool
+cockpit_offset:
+externalCar_offset:
+_pSsm: SmplSharedMemory*
+carVisualization: {CAR_VISUAL_EXTERNAL,
+SmplFSViewerSpecific(pSsm:SmplSharedMemory*,
+createVisual2dFullscreenProjection(pSimulationScreenState
+createVisualLandscape(openFlightTexture:string,
+createVisualCarAndTraffic(parent:osg::ref_ptr<osg::Group>
+createVisualSmplVehicle(pCorrespondingVehicle:SmplBase
+createVisualOrigin(AxisFatness:float, AxisLength:float):
+createVisual2dFullscreenImage(FullscreenFilename:string):
+toggleSubjectVehicleVisualization(): void
+toggleTrafficVisualization(): void
+toggleCurrentView(): void
+toggleOriginVisualization(): void
+toggle2dFullscreenProjection(pCamera:osg::Camera*):
+updateCamera(pCamera:osg::Camera*): void
+updateCameraM(pCamera:osg::Camera*): void
+updateVisualCar(): void
+updateVisualTraffic(trafficNumber:unsigned int): void
+updateVisualSky(): void
+updateVisual2dFullscreenImage(FullscreenFilename:string)
+showNodeNames(depth:int,
+findNamedNode(searchName:const std::string&,
_pEyeTrackingData
SmplEyeTrackingInterpretedData
-_VersionsNumber: unsigned int
+_Timestamp: unsigned long
+_ETState: _EyeTrackingState
+_GazeOnPlane: SmplString50
+_GazeOnPlaneNumber: int
+_GazeOnAOI: SmplString50
+_GazeOnAOINumber: int
+_LookAtPos: Smpl3dCoords
+_LookAtPlanePos: Smpl3dCoords
+_EyeSpeed: double
+_AngleBetweenTwoNGazeVectors: double
+_FixationDuration: double
+_FirstFixationDurationPlane: double
+_FirstFixationDurationAOI: double
+_GazeDurationPlane: double
+_GazeDurationAOI: double
+_ViewingTimePlane: double
+_ViewingTimeAOI: double
+_DwellPlane: double
+_DwellAOI: double
+_DwellDurationPlane: double
+_DwellDurationAOI: double
+_GlanceDurationPlane: double
+_GlanceDurationAOI: double
+_EyePositionX: double
+_EyePositionY: double
+_EyePositionZ: double
+_NormalizedGazeVectorX: double
+_NormalizedGazeVectorY: double
+_NormalizedGazeVectorZ: double
+_EyeTrackingState: {UNKNOWN_STATE, FIRST_VALUE, CLOSED_EYE, GUESSED_FIXATION, FIXATION,
+SmplEyeTrackingInterpretedData()
+getVersionsNumber(): unsigned int
+ResetInterpretedData(): void

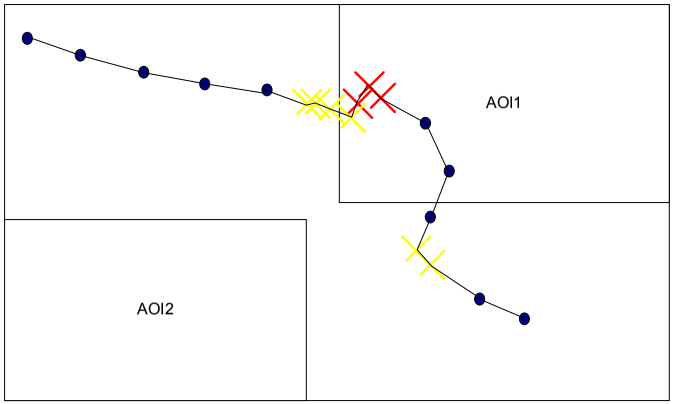

Legende
osg::Group*
osg::PositionAttitude Transform*
osg::MatrixTrasnform*
osg::Geode*
osg::Geometry* osg::ShapeDrawable*
osg::Texture
SmplFSViewer Root
ETRootXForm
ETRootScaleMatrix
SmplEyeTracking3dVisualisation EyeTrackingRoot
EyeTrackingRoot
besitzt als Kind
HTCubeGeode
BoxGroup
kann besitzen 0 bis * Kinder von
BoxXForm
1
0..*
MAX_PLANES (10)
BoxGroup
BoxXForm
BoxGeode
BoxGeometry
AxisGroup
AxisGeode
AxisX
AxisY
AxisZ
SeatGroup
SeatXForm
SeatGeode
SeatGeometry
RoomWall Group
RoomWalls XForm
RoomWall Geode
RoomWallOne Geometry
RoomWallTwo Geometry
PlanesXForm[MAX_PLANES]
AOIGeode
AOIGeometry
Plane Geode
Plane Geometry
CameraTexture
Plane „SimulationScreen"
VirtualEyeXForm
VirtualEye Geode
Virtual Eye
VirtualGaze Geode
VirtualGaze Geometry
GroundX Form
RoomBase Geode
Room Base
HTCube Geode
HTCube
WorldSphere Geode
WorldSphere
ETBasicData Geode
ETBasicPoint
ETBasicData ConnetionLines Geode
ETBasic ConnetionsLines Geometry
ETFixation Geode
ETFixation Point
ETFixation ConnetionLines Geode
ETFixationLines Geometry

Analyseparameter: FixationDuration [FD]





Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]

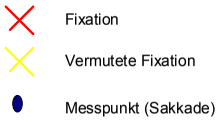


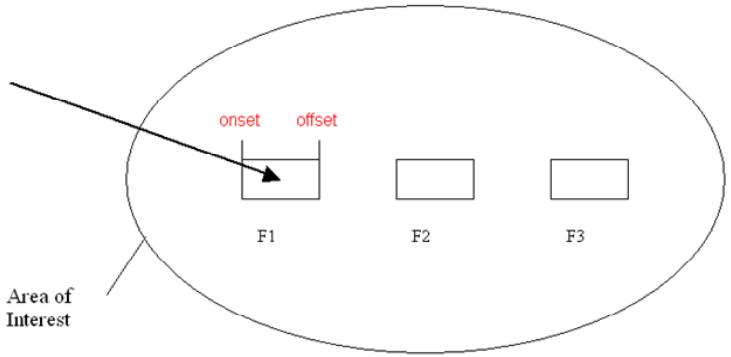
Plane

Bei der Berechnung der FixationDuration wird die Zeitdifferenz zwischen Vermuteten Fixationen und Fixationen aufaddiert und ausgegeben.

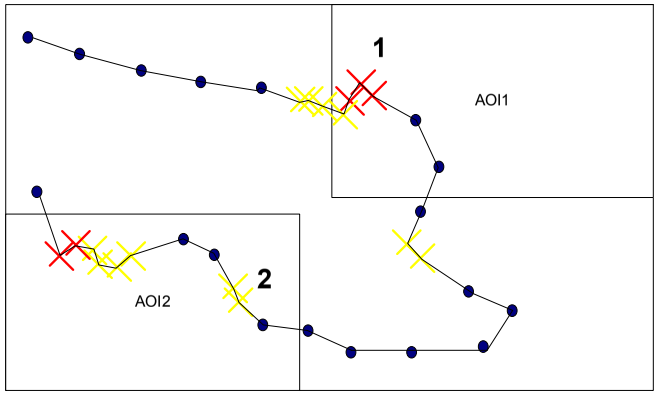
Demzufolge werden auch Vermutete Fixationen, die später nicht zu einer vollwertigen Fixation werden, ausgegeben!

Analyseparameter: FirstFixationDuration [FFD]





Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]



Plane

Darstellung zeigt die Onlineberechnung des Analyseparameters für AOIs, entspricht der Berechnung dieses Parameters für Planes

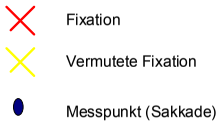
Bei der Berechnung der FirstFixationDuration wird die Zeitdifferenz zwischen der ersten gefundenen Vermuteten Fixation und Fixationen in einer AOI aufaddiert und ausgegeben.

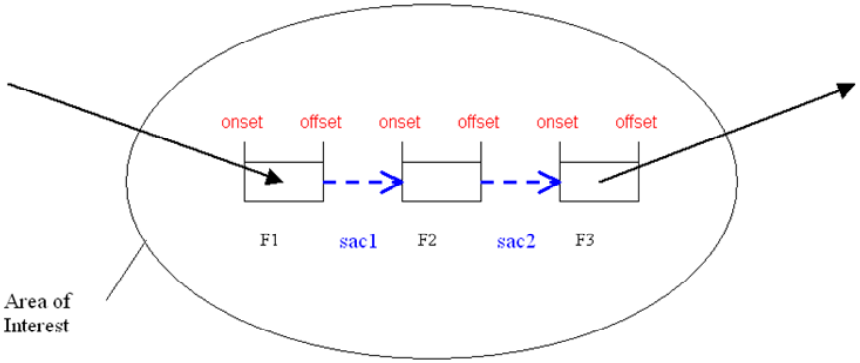
Wandert eine Fixation in die AOI (siehe 1), wird die Gesamtdauer dieser Fixation ausgegeben (inklusive der Dauer der Fixation vor dem Betreten der AOI).

In AOI2 würde als FFD die Dauer der als erstes gefundenen Fixation (2) ausgegeben werden, auch wenn diese im Nachhinein nicht zu einer vollwertigen Fixation wird.

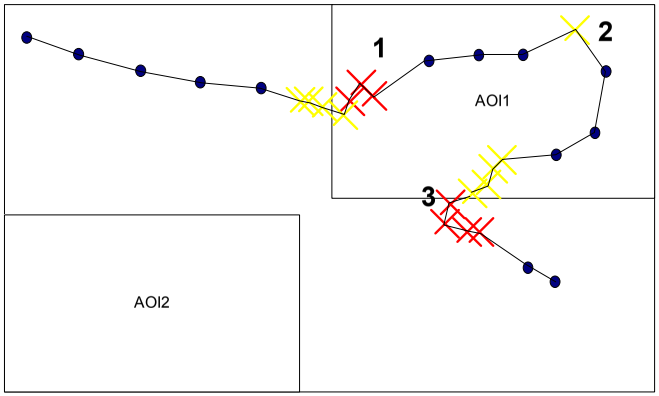
FFD, die in einer AOI beginnen, werden aufaddiert und ausgegeben, nur solange die Punkte innerhalb der AOI liegen. Bei einer aus der AOI hinauswandernden Fixation würde der Wert beim Verlassen einer AOI sein Maximum erlangen.

Analyseparameter: GazeDuration [GD]





Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]



Plane

Darstellung zeigt die Onlineberechnung des Analyseparameters für AOIs, entspricht der Berechnung dieses Parameters für Planes

Bei der Berechnung der GazeDuration werden die Fixationszeiten innerhalb einer AOI zusammengerechnet und ausgegeben.

Wandert eine Fixation (siehe 1) in die AOI, wird die Gesamtdauer dieser Fixation berücksichtigt.

Die Dauer von Vermuteten Fixationen, die zu keiner vollwertigen Fixation werden (2), wird ebenfalls hinzuaddiert.

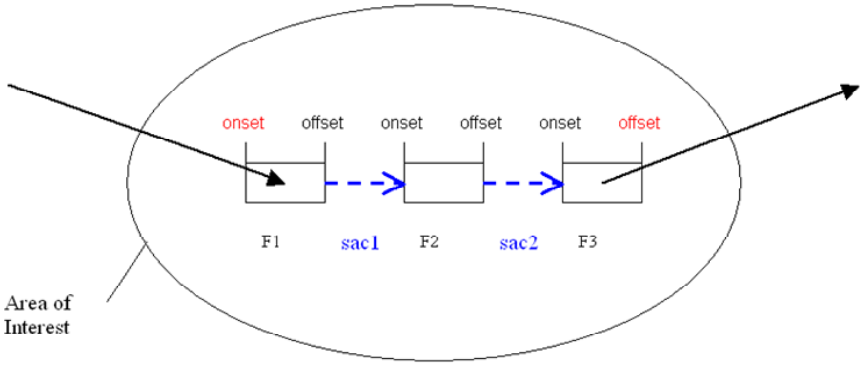
Fixationen, die in einer AOI beginnen, werden hinzuaddiert (3) und ausgegeben nur solange die Punkte innerhalb der AOI liegen.

Analyseparameter: ViewingTime [VD]

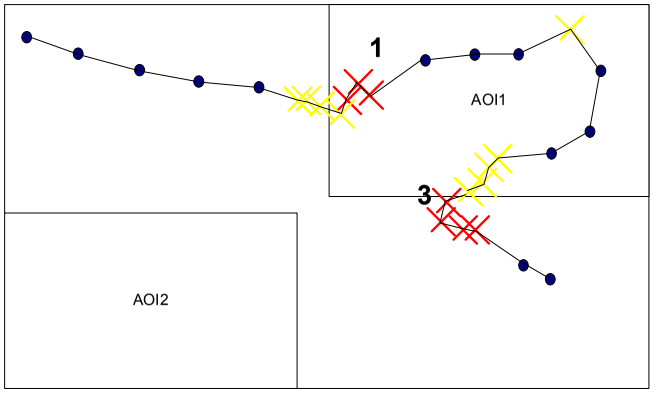
Fixation

Vermutete Fixation

Messpunkt (Sakkade)



Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]



Plane

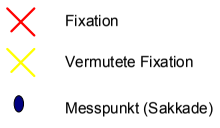
Darstellung zeigt die Onlineberechnung des Analyseparameters für AOIs, entspricht der Berechnung dieses Parameters für Planes

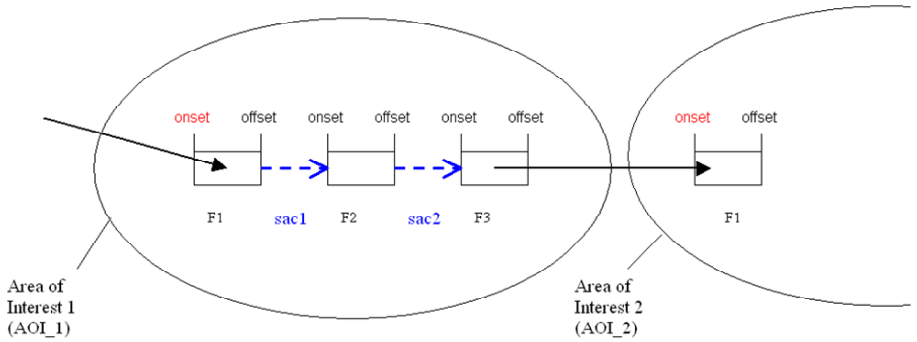
Bei der Berechnung der ViewingTime werden ab der ersten gefundenen Vermuteten Fixation die Zeitdifferenzen zwischen zwei Messwerten aufaddiert (auch bei Sakkaden). Ausgegeben wird der berechnete Wert immer dann, wenn eine Vermutete Fixation oder Fixation vorliegt!

Wandert eine Fixation (siehe 1) in die AOI, wird die Gesamtdauer dieser Fixation berücksichtigt.

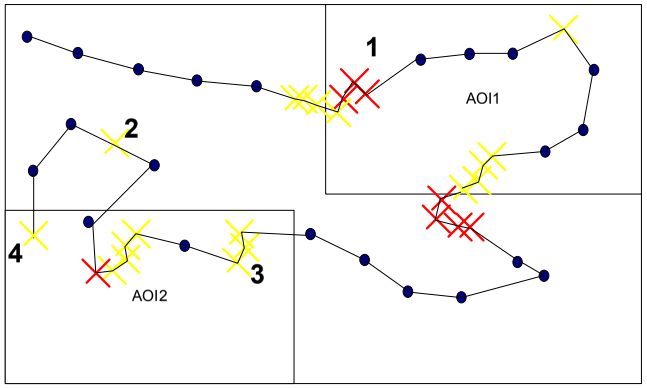
Fixationen, die in einer AOI beginnen, werden hinzuaddiert (3) und ausgegeben nur solange die Punkte innerhalb der AOI liegen.

Analyseparameter: Dwell [D]





Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]



Plane

Darstellung zeigt die Onlineberechnung des Analyseparameters für AOIs, entspricht der Berechnung dieses Parameters für Planes

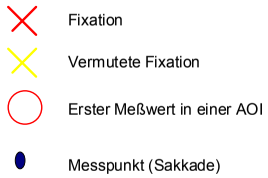
Bei der Berechnung der Dwell werden ab der ersten Vermuteten Fixation alle vorkommenden Zeitdifferenzen zwischen zwei Messwerten aufaddiert (auch bei Sakkaden). Ausgegeben wird der berechnete Wert immer dann, wenn eine Vermutete Fixation oder Fixation innerhalb einer AOI vorliegt!

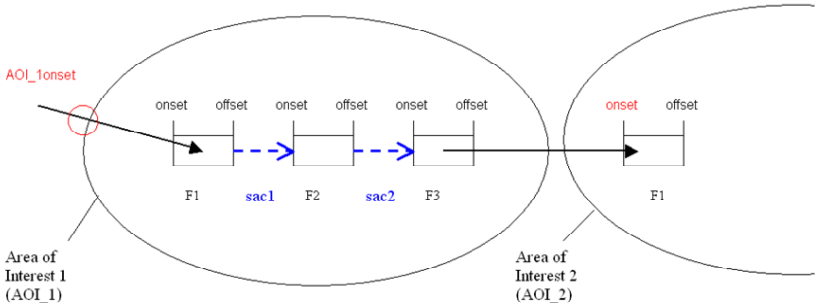
Wandert eine Fixation (siehe 1) in die AOI, wird die Gesamtdauer dieser Fixation berücksichtigt.

Die letzte Ausgabe des Dwellwertes und der Beginn eines neuen Dwellwertes ist die erste Vermutete Fixation in einer AOI (3 oder 4). Dort wird zuletzt die Zeitdifferenz vom Verlassen einer AOI bis zum Wiedereintritt in eine AOI hinzuaddiert.

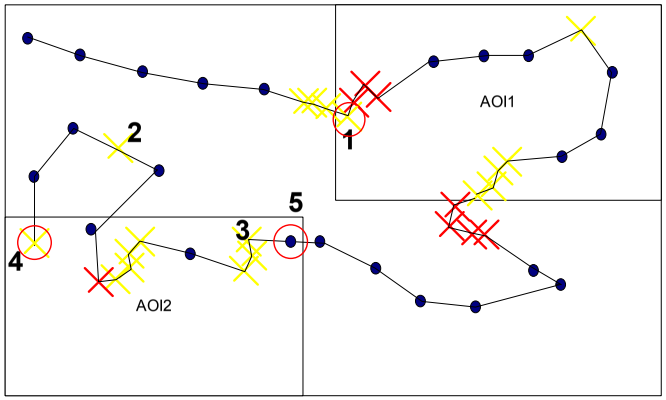
Außerhalb der AOI wird kein Wert ausgegeben (siehe z.B. 2).

Analyseparameter: DwellDuration [DD]





Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]



Plane

Darstellung zeigt die Onlineberechnung des Analyseparameters für AOIs, entspricht der Berechnung dieses Parameters für Planes

Bei der Berechnung der DwellDuration werden alle Zeitdifferenzen ab dem Eintritt in eine AOI zwischen zwei Messwerten aufaddiert (auch bei Sakkaden). Ausgegeben wird der berechnete Wert immer dann, wenn eine Vermutete Fixation, Fixation oder Sakkade innerhalb einer AOI vorliegt!

Wandert eine Fixation (siehe 1) in die AOI, wird diese Fixation ab dem Eintrittsmeßwert berücksichtigt.

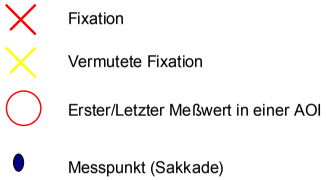
Die DwellDuration beginnt mit dem ersten Wert in einer AOI (z.B. 5).

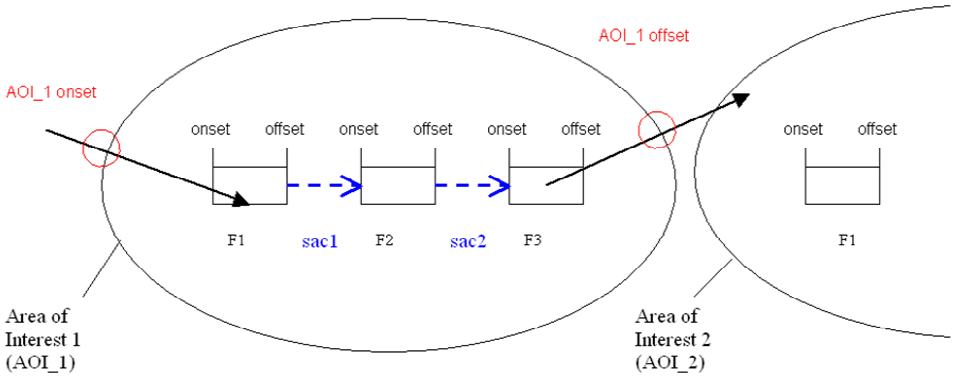
Die letzte Ausgabe des DwellDurationwertes ist die erste gefundene Vermutete Fixation in einer AOI (3 oder 4). Dort wird zuletzt die Zeitdifferenz vom Verlassen einer AOI bis zum Wiedereintritt in eine AOI hinzuaddiert.

Die erste Ausgabe eines neuen DwellDurationwertes erfolgt nach der ersten gefundenen Vermuteten Fixation in einer AOI (3 oder 4) unter Berücksichtigung der verstrichenen Zeit seit dem Eintritt in die AOI (seit z.B. 5).

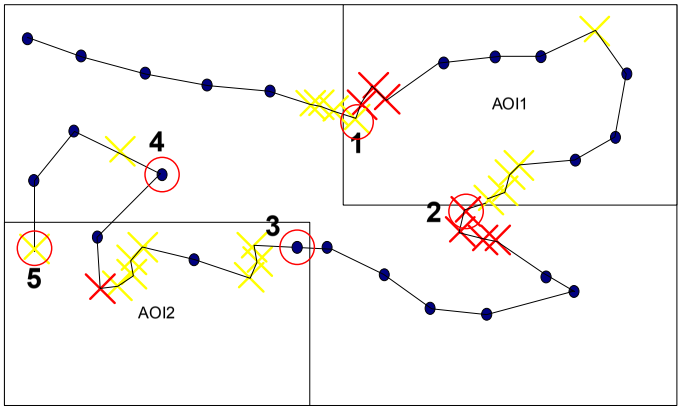
Außerhalb der AOI wird kein Wert ausgegeben (siehe z.B. 2).

Analyseparameter: GlanceDuration [GLD]





Beschreibendes Schaubild aus dem Dokument :
„PJ_FS_Blickbewegungen_050616.doc“ von Lethaus F. [DLR]



Plane

Darstellung zeigt die Onlineberechnung des Analyseparameters für AOIs, entspricht der Berechnung dieses Parameters für Planes

Die GlanceDuration gibt den Zeitraum vom Betreten einer AOI (1, 3 oder 5) bis zum Verlassen einer AOI (2 oder 4) an.

Der Wert wird vom ersten (1) bis zum letzten Meßwert in der AOI (vor 2) berechnet und ausgegeben.

Abbildungsverzeichnis

Listingverzeichnis

Literaturverzeichnis

[**Buch-Literatur:**]

[Bron00] **Bronstein: Taschenbuch der Mathematik** 5. Auflage; 2000; ISBN 3-8171-2005-2

[Fi99] **Fischer B.: Blickpunkte** 1999; ISBN 3-456-83147-1

[Fle00] **Flemisch, F.: Pointillistische Analyse der visuellen und nichtvisuellen Interaktionsressourcen am Beispiel Pilot - Assistenzsystem** Dissertation; 2000

[Ju06] **Jungeblut, J.: Entwicklung einer Klassenbibliothek in C++ zur Integration von CAN-Peripherie in eine Applikationsund Entwicklungsplattform im Automotivebereich** Diplomarbeit; 2006

[Let05] **Lethaus F.: Das Messen von Blickbewegungen -Kurzüberblick-** DLR-internes Dokument; 2005; PJ_FS_Blickbewegungen_050616.doc

[Oes05] **Oestereich, B.: Die UML 2.0 Kurzreferenz für die Praxis** 4. Auflage; 2005; ISBN 3-486-577883

[Pap01a] **Papula L.: Mathematische Formelsammlung** 7. Auflage; 2001; ISBN 3-528-64442-7

[Pap01b] **Papula L.: Mathematik für Ingenieure und Naturwissenschaftler Band 1** 10. Auflage; 2001; ISBN 3-528-94236-3

[PK05] **Prinz P., Kirch-Prinz U.: C++ Lernen und professionell anwenden** 3. Auflage; 2005; ISBN 3-8266-1534-4

[Röt01] **Rötting M.: Parametersystematik der Augen- Blickbewegung für arbeitswissenschaftliche Untersuchungen** Dissertation; 2001

[Schl07] **Schlemmer, J.: Entwicklung eines Programms zur Weiterverarbeitung von eyetracking-Daten und zweidimensionale Darstellung von dreidimensionalen Blickpunkten in einem Video** Diplomarbeit; 2007

[STR00] **Stroustrup, Dr. B.: Die C++ Programmiersprache, Deutsche Übersetzung der Special Edition (Professionelle Programmierung)** 4. Auflage; 2000; ISBN 3-8273-1660-X

[**Weblinks:** Stand 17.07.07]

[Agil07] **Manifesto for Agile Software Development**
Weblink: http://www.agilemanifesto.org

[BM07] **Behme H. & Mintert S.: XML in der Praxis; Extensible Markup Language für Profis**
Online-Buch Weblink: http://www.linkwerk.com/pub/xmlidp/2000/

[Cpp07] **C++ Community** Weblink: http://www.c-plusplus.de/cms/

[DLR07a] **Deutsches Zentrum für Luft- und Raumfahrt**
Weblink: http://www.dlr.de

[DLR07b] **Institut für Verkehrsführung und Fahrzeugsteuerung (FS) des DLR**
http://www.dlr.de/fs/desktopdefault.aspx/tabid-1213/

[Dox07] **Documentation system doxygen**
Weblink: http://www.stack.nl/∼ dimitri/doxygen

[FL07] **Seeing machines** Weblink: http://www.facelab.com

[JRV07] **Joos M., Rötting M. & Velichkovsky B.M.: Bewegungen des menschlichen Auges: Fakten, Methoden und innovative Anwendungen**
Weblink: http://rcswww.urz.tu-dresden.de/∼ cogsci/pdf/joos02.pdf

[Lex07] **Automobilhersteller Lexus**
Weblink: http://www.lexus.de/pursuit_perfection/index.asp

[OMG07] **Object Management Groups (OMG's) UML**
Weblink: http://www.omg.org/gettingstarted/what_is_uml.htm

[OSG07] **OpenSceneGraph** Weblink: http://www.openscenegraph.org

[SMI07] **SensoMotoric-Instrumens** Weblink: http://www.smi.de

[Wiki07a] **Wikipedia über endliche Automaten**
http://de.wikipedia.org/wiki/Endlicher_Automat

[Wiki07b] **Wikipedia über Agile_Softwareentwicklung**
http://de.wikipedia.org/wiki/Agile_Softwareentwicklung

[Wiki07c] **Wikipedia über UML**
Weblink: http://de.wikipedia.org/wiki/Unified_Modeling_Language

[Wiki07d] **Wikipedia über UDP**
Weblink: http://de.wikipedia.org/wiki/User_Datagram_Protocol

[Wiki07e] **Wikipedia über TCP/IP**
Weblink: http://de.wikipedia.org/wiki/TCP/IP

[Wiki07f] **Wikipedia über XML**
Weblink: http://de.wikipedia.org/wiki/Extensible_Markup_Language

[Wiki07g] **Wikipedia über C++**
Weblink: http://www.c-plusplus.de/cms/

[**Zietierte Literaturstelle** aus [Röt01]]

[CU98] **Crundall, D. & Underwood, G.: Effects of experience and processing demands on visual Information acquistion in drivers.** *Ergonomics*, 41(4), 448-458.

[**Zietierte Literaturstellen** aus [Let05]]

[HF04] **Henderson, J.M., & Ferreira, F.: Scene perception for psycholinguists. In J.M. Henderson & F. Ferreira (Eds.), The interface of language, vision, and action: Eye movements and the visual world (pp. 1-58)** 2004; New York: Psychology Press

[Karn02] **Karn, K.S.: Definitions: "dwell"vs. "gaze"** 2002

[LF00] **Liversedge, S.P., & Findlay, J.M.: Saccadic eye movements and cognition. TRENDS in Cognitive Science, 4, 8-14** 2000

[ML04] **Meyer, A.S., & Lethaus, F.: The use of eye tracking in studies of sentence generation. In J.M. Henderson & F. Ferreira (Eds.), The interface of language, vision, and action: Eye movements and the visual world (pp. 191 211)** 2004; New York: Psychology Press

[SR01] **Starr, M.S., & Rayner, K.: Eye movements during reading: some current controversies. TRENDS in Cognitive Science, 5, 158-163** 2001

[VHE05] **Victor, T.W., Harbluk, J.L., & Engström, J.A.: Sensitivity of eye-movement measures to in-vehicle task difficulty. Transportation Research Part F, 8(2), 167-190** 2005