Universität Koblenz-Landau

Fraunhofer IAIS

Diplomarbeit

Visualisierung und Interaktion von Trajektorien und assoziierter multidimensionaler Daten mittels Multitouch-Tablets

von

Marc Rödder

Aufgabenstellung und Betreuung:

Prof. Dr. Stefan Müller
M.Sc. David d'Angelo
Koblenz, den 21. Februar 2012

Erklärung

Ich versichere, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

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

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

Koblenz, den 21. Februar 2012

Institut für Computervisualistik
AG Computergraphik
Prof. Dr. Stefan Müller
Postfach 20 16 02
56 016 Koblenz
Tel.: 0261-287-2727
Fax: 0261-287-2735
E-Mail: stefanm@uni-koblenz.de

UNIVERSITÄT KOBLENZ · LANDAU

Fachbereich 4: Informatik

Aufgabenstellung für die Diplomarbeit
Marc Rödder
(Matrikel-Nr. 204 210 163)

Thema: Visualisierung und Interaktion von Trajektorien und assoziierter multidimensionaler Daten mittels Multitouch-Tablets

CamInSens[1] ist ein vom BMBF geförderter Forschungsverbund mit dem Ziel, ein praxis- und rechtskonformes sowie intelligentes Videosystem aufzubauen.

In derzeitigen Überwachungssystemen werden Daten von Videokamerainfrastrukturen dem Sicherheitspersonal üblicherweise in einer zentralen Stelle als Projektionsfläche angezeigt. Diese Systeme leisten jedoch nicht, Bedrohungssituationen schon im Moment ihrer Entstehung zu erkennen, da die Videodaten meist erst nach Stattfinden eines Vorfalls zur Aufklärung oder Verfolgung genutzt werden.

Im Gegensatz dazu soll CamInSens den Benutzer nun automatisch und unmittelbar auf potentielle Gefährdungssituationen aufmerksam machen. Die Auswertung der Bildfolgen soll dabei nicht nur zur Erkennung von auffälligen Verhaltensmustern, sondern auch zur Steuerung der beweglichen Kameras dienen.

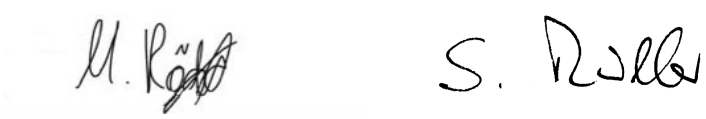
Diese Diplomarbeit soll nun erreichen, auf Basis von Daten aus dem CamInSens-Verbund eine effektive Interaktion und Visualisierung auf mobilen Geräten zu ermöglichen. Die daraus gewonnenen Informationen werden anschliessend von Sicherheitspersonal genutzt, um korrekte Entscheidungen hinsichtlich verschiedener Bedrohungssituationen treffen zu können. Die Eingabedaten bestehen hierbei aus automatisch bewerteten Trajektorien sowie assoziierten Zusatzinformationen, die bereits durch CamInSens bereitgestellt werden. Dabei handelt es sich zum Beispiel um die Bildinformationen aller an einer Trajektorie beteiligten Kameras oder weiterer Datenquellen (Glasbruch-, Schloss-, Lichtsensoren etc).

Die Herausforderung dieser Aufgabenstellung ist, die Überflutung von Informationen, insbesondere von Trajektorien, in eine schnell erfassbare Form zu bringen und darzustellen. Es soll dem Sicherheitspersonal dadurch möglich werden, korrekte Entscheidungen zu treffen, um Gefahrensituationen zu entschärfen oder zu verhindern.

Weitere Beachtung gilt dem Formfaktor und der Interaktionsmöglichkeiten gängiger Multitouch-Tablets. Sowohl müssen Visualisierungstechniken abgeändert oder neu entwickelt werden, als auch entsprechende Multitouch-Interaktionslogiken gefunden werden.

Die Diplomarbeit wird am Fraunhofer-Institut für Intelligente Analyse- und Informationssysteme (IAIS) in Sankt Augustin durchgeführt.

Betreuer: M.Sc. in Computer Science David d'Angelo (Fraunhofer IAIS, St. Augustin)



Koblenz, den 19. August 2011 - Marc Rödder - - Prof. Dr. Stefan Müller-

1 http://caminsens.org

Danksagung

Ich möchte zuallererst meinen Eltern danken, die mich jederzeit finanziell als auch moralisch auf dem Weg zu dieser Abschlussarbeit unterstützt haben. Vielen Dank dafür!

Natürlich danke ich auch den Herren Prof. Stefan Müller und David d'Angelo für die sehr gute Betreuung während der Durchführung und Verschriftlichung dieser Diplomarbeit. Ebenso danke ich den Mitarbeitern bei Fraunhofer IAIS, namentlich Dr. Manfred Bogen, Stefan Rilling, Delger Lhamsuren, Waldemar Schwan und Martin Panknin. Es hat immer Spass mit euch gemacht, und ich habe aus meinem knappen Jahr bei Fraunhofer sehr viel mitgenommen!

Ich danke allen Mitstreitern in der Bibliothek während der letzten knapp 2 Jahre. Ohne euch wäre mir das Lernen und Arbeiten während der Diplomprüfungen und der Diplomarbeit wohl nicht so einfach gefallen! Danke auch an meine Mitbewohner und an mein nahes Umfeld, die es akzeptiert haben, dass ich die letzten Monate nahezu unter Ausschluss jeglichen Soziallebens verbracht habe.

Danke außerdem an die Probanden meines Benutzertests und natürlich an Herrn Jelinski (IVE mbH) und die Firma Serco, die uns einen Besuch in der 3S-Zentrale im Hauptbahnhof Köln sowie in der Sicherheitszentrale der JVA Hünfeld ermöglichten. Es war in jeder Hinsicht eine interessante Erfahrung!

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Tabellenverzeichnis

1. Einleitung

1.1. Motivation

Bereits seit den 1950er Jahren wird Videoüberwachung in Deutschland eingesetzt, um bedarfsgerechte Entscheidungen auch ohne persönliche Anwesenheit an einem Ort oder mit besserer Übersicht treffen zu können. So wurde aufgrund des gestiegenen Automobil-Verkehrsaufkommens 1956 in Hamburg ein Verkehrsleitsystem vorgestellt, welches mit Hilfe von Videoüberwachung, fernsteuerbaren Ampeln und Bedienpersonal den innerstädtischen Verkehr regeln konnte. 1958 zog München nach und erweiterte bis 1965 das System auf neunzehn Kameras, die zusätzlich geschwenkt und geneigt werden konnten. [KAM]

In den nächsten Jahren folgten immer mehr Städte diesem Beispiel und mit dem technischen Fortschritt und der daraus resultierenden Vergünstigung der Geräte hielt die Überwachungstechnik bis Ende der 1970er Jahre auch Einzug in viele andere Bereiche wie Banken oder Grenzübergänge.

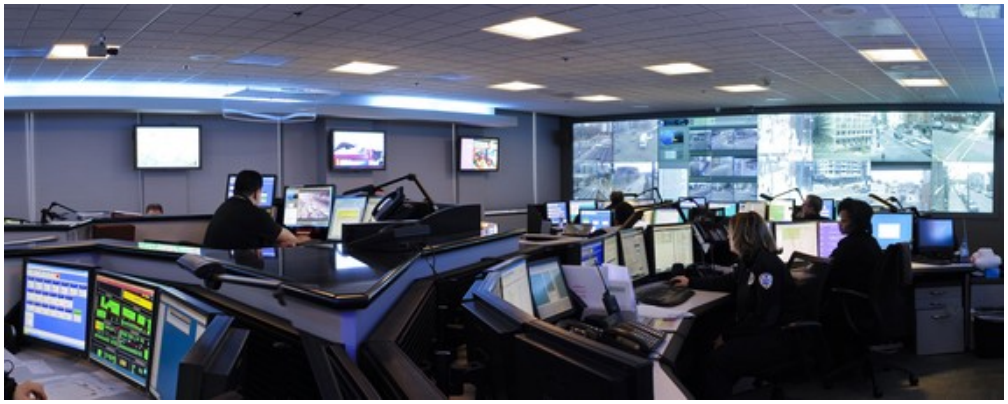


Abbildung 1.1.: Kontrollraum in einer Chicagoer Sicherheitszentrale, Quelle: [KON]

Heute werden Kameras an vielen Orten benutzt - Bahnhöfe, Stadien, Tankstellen, Kaufhäuser, sogar ganze Innenstädte werden flächendeckend überwacht. Als Beispiel einer in großem Stil videoüberwachten Stadt kann Chicago dienen, wo neben 1.500 offiziell bekannten Kameras der Polizei ebenfalls zahlreiche Kameras anderer staatlicher oder privater Einrichtungen (Bus, Bahn, Schulen etc.) ihren Beitrag liefern. Das von International Business Machines Corp. [IBM] entwickelte Kameranetz erlaubt dabei sogar die Bereitstellung von Videokameras in Privathaushalten, so dass die Gesamtzahl der verfügbaren Kamerainstanzen auf 15.000 geschätzt wird. [CCT]

Videoüberwachungssysteme werden derzeit in diesen Bereichen überwiegend genutzt, um Sicherheit gewährleisten zu können. Neben der Abschreckung von Krimi-

nellen durch alleinige Anwesenheit der Kameras [UED] besteht die Systematik der Nutzung von Videodaten in diesem konventionellen Szenario allerdings meist nur darin, die bewegten Bilder auf einer Projektionsfläche in einer Zentrale für die Sicherheitskräfte sichtbar zu machen (siehe Abb. 1.1). Mit dieser Technik gelingt es jedoch nicht, Bedrohungen bereits präventiv zu erkennen und im Augenblick der Entstehung einzudämmen, vielmehr kann aufgrund der grenzwertig hohen kognitiven Auslastung der Mitarbeiter nur reagiert statt agiert werden. Die aufgezeichneten Bilder werden in solchen Fällen daher überwiegend erst im Nachhinein zur Rekonstruktion und Aufklärung des Sachverhalts herangezogen.

Es gilt nun, den zwischenzeitlichen technischen und intellektuellen Fortschritt zu nutzen, um diese ungesehenen Daten effektiver zu verarbeiten und dadurch effizientere Arbeitsbedingungen für das Sicherheitspersonal zu schaffen.

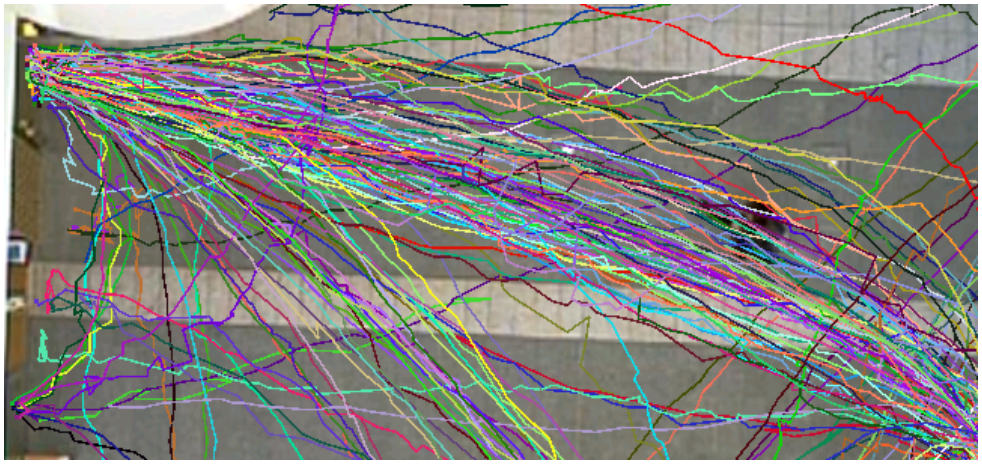


Abbildung 1.2.: Trajektorien vieler verschiedener Personen, die einen Platz überqueren

1.2. Problemstellung

Menschliches Fehlverhalten bildet oft den Auslöser zu Gefahrensituationen, so dass eine Auswertung von Personen-Trajektorien (hier: zeitabhängige Bewegungspfade von Personen, siehe Abb. 1.2) [TRA] helfen kann, Gefahrensituationen frühzeitig zu erkennen. Ist diese Trajektorien-Analyse schnell genug, können Bedrohungen in situ, also bereits im Moment ihrer Entstehung, automatisch erkannt werden. Eine Verbesserung der Detektionsleistung kann zusätzlich durch Ergänzung dieser intelligenten Videosysteme um Sensorik, wie etwa Kleinstsensoren für Licht, Erschütterung oder Temperatur, herbeigeführt werden. Weitere Schritte in der Verarbeitung der daraus erhaltenen Daten umfassen die Erkennung, Nachverfolgung und Wiederauffindbarkeit von bewegten Objekten, sowie Analysen zur Verhaltensweise der Objekte. Typische Fachgebiete, die zur Lösung dieser Aufgaben zu Rate gezogen werden, sind Bildverarbeitung [Jäh05], Mustererkennung [MUS], Künstliche Intelligenz [KI] und Datenbanken-Management [DAT].

Kommerzielle Überwachungssysteme wie *DETEC* [DET] oder *Gotcha* [GOT] sind meist nur in der Lage, eine Objekterkennung durchzuführen und bei Eintritt dieser Objekte in eine Szene die damit verbundenen Bilddaten abzuspeichern. *DETEC* erlaubt dabei zusätzliche eine Vernetzung von 'Workstations', welche jeweils bis zu 12 Kameras unterstützen [VV05].

DETER [PMTH01] [PM02] ist ein aus der Forschung stammendes System, welches auf die automatische Erkennung von unüblichen Bewegungsmustern von Personen oder Autos abzielt. Während das System eine gute Objekterkennungsleistung vorweisen kann, ist die einsetzbare Anzahl von Kameras durch die teuren Berechnungsalgorithmen zur Mustererkennung jedoch begrenzt. [VV05]

CamInSens [CAM] ist ein seit 2009 bestehender, vom Bundesministerium für Bildung und Forschung (*BMBF* [BMB]) geförderter Forschungsverbund mit dem Ziel, ein praxis- und rechtskonformes [HDP10] [HOR], intelligentes Videosystem aufzubauen, das auf Gefährdungssituationen unmittelbar und automatisch aufmerksam macht.

Das System soll einerseits, ähnlich zu *DETER*, auffällige Bewegungsmuster erkennen, andererseits aber zusätzlich die Kameras automatisch nachführen, so dass relevante Szenen im Sichtbereich bleiben. Dies bietet auch den Vorteil der Kostenersparnis, denn aufgrund der automatischen Nachführung ist es möglich, einen Bereich mit weniger Kameras zu überwachen, als es bei starr eingestellten Kameras der Fall wäre. Bei der Mustererkennung sollen in erster Linie automatisierbare Verfahren erprobt werden, um die Vorgänge der manuellen Sichtung von Videomaterial auf ein Minimum zu reduzieren. Zusätzlich soll es möglich werden, Personen über mehrere Kamera-Sichtfelder hinweg verfolgen zu können. Eine Herausforderung stellen hierbei Szenen mit vielen Personen oder Verdeckungen dar.

Die Objekterkennung wurde in diesem Projekt durch die Verwendung von neuronalen Netzen und Hintergrund-Subtraktion gelöst [JPWH10] [MP11]. Um Objekte auch über Kamera-Sichtfelder hinweg verfolgen und wiedererkennen zu können, wurden Techniken zur kameraübergreifenden Lokalisierung und multisensoriellen Auswertung entwickelt [Mon11]. Auf Basis dieser Daten konnten schliesslich Ansätze zur Mustererkennung von Trajektorien entworfen werden, welche dem Endbenutzer Informationen über auffällige Trajektorien oder überlastete Bereiche liefern können. [COLb]

Die Schnittstelle zwischen diesen Technologien und dem Überwachungspersonal soll einerseits in einem gesicherten Kontrollraum, und andererseits am Ort des Geschehens, also durch mobile Kräfte, gebildet werden. Das mobile Personal soll mit Geräten und der passenden Software ausgestattet werden, um diese Projektdaten jederzeit vor Ort abrufen zu können. Die Visualisierung soll dabei im Kontrollraum sowie auf den mobilen Geräten 'ständig aktualisiert' und 'situationsgemäß visualisiert' werden [PRO09]. Diese Informationen sollen den Benutzern im weiteren Verlauf als Basis dienen, um Maßnahmen zur Schlichtung einer potenziellen Gefahrensituation ergreifen zu können. Dabei soll es beispielsweise möglich sein, Alarme zu senden, welche die Aufmerksamkeit des Sicherheitspersonals auf bestimmte Trajektorien oder Objekte richten.

1.3. Ziele der Arbeit

Diese Diplomarbeit hat zum Ziel, auf Basis von Daten aus dem *CamInSens*-Verbund eine effektive Interaktion und eine Visualisierung dieser Daten auf mobilen Geräten zu ermöglichen. Die daraus gewonnenen Informationen werden von Sicherheitspersonal genutzt, um korrekte Entscheidungen hinsichtlich verschiedener Bedrohungssituationen treffen zu können. Die Eingabedaten bestehen hierbei aus automatisch bewerteten Personenbewegungstrajektorien sowie assoziierten Zusatzinformationen, die bereits durch *CamInSens* bereitgestellt werden. Dabei handelt es sich etwa um die Bildinformationen aller an einer Trajektorie beteiligten Kameras oder weiterer Datenquellen (Glasbruch-, Schloss-, Lichtsensoren etc).

Die Herausforderung dieser Aufgabenstellung ist, durch Anzeige der Informationen in einer geeigneten Form eine Informationsüberflutung zu vermeiden. Durch diese Darstellung soll es dem Sicherheitspersonal möglich werden, korrekte Entscheidungen zu treffen, um Gefahrensituationen zu entschärfen oder zu verhindern.

Weitere Beachtung gilt dem Formfaktor und der Interaktionsmöglichkeiten gängiger Multitouch-Tablets. Sowohl müssen Visualisierungstechniken abgeändert oder neu entwickelt werden, als auch entsprechende Multitouch-Interaktionslogiken gefunden werden, da die meisten bisherigen Forschungsergebnisse in dieser Richtung auf sehr viel größere *MultiTouch*-Darstellungsflächen abzielen (u.a. [MJGJ11], [WCRI09] und [SvdCIS09]). Die stark verkleinerten Oberflächen eines Tablet-PC bilden somit eine ausreichend großen Anlass, um den Einsatz der üblichen *MultiTouch*-Gesten für diesen Einsatzzweck zu überdenken.

1.4. Struktur der Arbeit

Das folgende Kapitel beleuchtet das *CamInSens*-Einsatzszenario als Beispiel für die Überwachung eines sicherheitskritischen Bereichs. Verschiedene Anwendungsfälle werden vorgestellt, aufgrund derer Anforderungen an eine mobile Sicherheitssoftware entwickelt werden. Das Kapitel schliesst mit der Sichtung eines geeigneten Geräts und der Auswahl einer geeigneten Softwarebasis, wie etwa eines Betriebssystems oder einer Grafikbibliothek.

Die Visualisierung der Daten bildet im anschliessenden Kapitel den Schwerpunkt. Es wird beschrieben, welche Daten für die Entwicklung benutzt wurden, wie Trajektorien bisher in anderen Forschungsprojekten visualisiert wurden und wie eine Gruppierung der Daten ermöglicht werden kann, um dominante oder auch anormale Bewegungsströme erkennen zu können. Es folgt eine Bewertung, welche Gruppierungsart in diesem Kontext Sinn macht, und schliesslich werden die damit verbundenen Teile der Implementierung vorgestellt.

Die Realisierung der Benutzerschnittstelle und eine Vorstellung des Bedienkonzepts folgt im Anschluss. Dabei wird im Speziellen erläutert, welche Touch-Logiken für den Projektrahmen gut geeignet sind, welche vermieden werden sollten und wie das Anwendungsszenario strukturiert sein sollte, um den Benutzer nicht zu unter- oder überfordern. Es werden außerdem verschiedene wissenschaftliche Studien zur *MultiTouch*-Interaktion ausgewertet, deren Ergebnisse ebenfalls in das Konzept einfliessen.

Abschliessend erfährt diese Arbeit eine Bewertung mittels eines Benutzertests und entsprechender Analyse; ein darauf folgender Ausblick trägt alle Verbesserungsmöglichkeiten zusammen, die während des Projekts oder durch den Benutzertest gewonnen wurden.

2. Das CamInSens - Anwendungsszenario

Das *CamInSens*-Projekt bot bereits Anforderungen an eine Sicherheitssoftware, die im Rahmen eines Projekt-Arbeitspaketes gebildet wurden. Diese reichten aber bei weitem nicht aus, um zu einer endgültigen Entscheidung hinsichtlich einer Plattform- und Gerätewahl für eine Sicherheitslösung gelangen zu können. Folglich wurde der Kontext des *CamInSens*-Projekts nur als Beispiel genommen, um schliesslich zu weiteren Anforderungen an eine Sicherheitssoftware für mobile Einsatzkräfte zu gelangen.

Darüberhinaus wurden Kontrollräume der Sicherheitszentralen im Kölner Hauptbahnhof sowie in der Justivollzugsanstalt Hünfeld besucht, um eine Vorstellung des Arbeitsalltags der Mitarbeiter und des technischen Stands der verwendeten Gerätschaften zu erhalten. In der 3S-Zentrale im Kölner Hauptbahnhof laufen, wie in allen anderen 3S-Zentralen, alle wichtigen Informationen zum Betriebsablauf im Bahnhof zusammen. [3S] Die 3S-Zentrale in Köln verwaltet dabei zusätzlich auch den Betriebsablauf von 130 kleineren Bahnhöfen rund um Köln/Bonn. Neben der Koordinierung eines sicheren Eisenbahnbetriebes haben die Mitarbeiter zusätzlich die Aufgabe, hilfesuchende Reisende mit Informationen zu versorgen. Es können alle Bereiche der Bahnhöfe mit Hilfe von Kameratechnik überblickt werden, so dass bei einer Gefahr hinsichtlich Sicherheit oder Sauberkeit ein dementsprechender mobiler Trupp aktiviert wird. Bei Ausfall von technischen Anlagen jedweder Art ist es auch hier die 3S-Zentrale, in der die Fehlermeldung aufläuft und kanalisiert wird.

Die Überwachung sicherheitskritischer Bereiche wird hier durch Anzeige der Videodaten von 80 (teilweise steuerbaren) Kameras auf 5 Kameramonitoren realisiert, wobei die Kameras entweder direkt angewählt oder in einem vordefinierten Muster durchgeschaltet werden können. Die einzige Verbindung zum mobilen Personal besteht auf einer rein audiellen Ebene durch Nutzung von handelsüblichen Mobiltelefonen. Verbesserungspotential wurde neben trivialen Dingen, wie Erneuerung von Hardware, durchaus auch in dem Einsatz von automatisierter Bilderkennung gesehen, wobei jedoch bereits ein System getestet und als nicht praxistauglich empfunden wurde. [GOL]

Die ebenfalls besuchte Justizvollzugsanstalt Hünfeld [JVA] ist die bisher einzige teilprivatisierte JVA im Bundesgebiet. Die Privatisierung bezieht sich hier nur auf weitgehend unkritische Bereiche wie Gebäudemanagement oder Videoüberwachung; aufgrund von verfassungsrechtlichen Vorgaben verbleibt die Organisationshoheit weiterhin bei Polizeibeamten. Uns eröffnete die Privatisierung im Rahmen von *CamInSens* die seltene Chance, einen JVA-Kontrollraum in Aktion betrachten zu können, denn ansonsten wird der Zutritt generell verwehrt. Ähnlich zur 3S-Zentrale Köln besteht die Strategie der Kameraüberwachung auch hier darin, auf einer aus 12 Monitoren bestehenden Wand verschiedene Kamerabilder einzublenden. Die Kommunikation zu den mobilen Kräften erfolgt über ein spezielles Personennotfallgerät (*PNG*), was eine bidi-

rektionale Sprachverständigung per Analogfunk anbietet. Zusätzlich können auf dem *PNG* Textnachrichten empfangen und Alarme abgesetzt werden. Als mögliche Verbesserungen wurden auch hier eine automatische Bildverarbeitung sowie darüberhinaus ein Austausch der analogen Funktechnik angeführt. Eine Erweiterung des aktuellen Konzepts um eine Bildempfangskomponente der mobilen Sicherheitskräfte oder die Einführung eines neuen mobilen Geräts wurde jedoch abgelehnt, da die Einarbeitungszeit in neue Techniken immer mit Sicherheitsrisiken verbunden ist.

Von den aus diesen Besuchen gewonnenen Informationen wurden im weiteren Verlauf der Arbeit Softwareanforderungen sowie Entscheidungen betreffend der Hardware (siehe auch Kapitel 2.4) abgeleitet.

2.1. Nähere Betrachtung des CamInSens-Systems

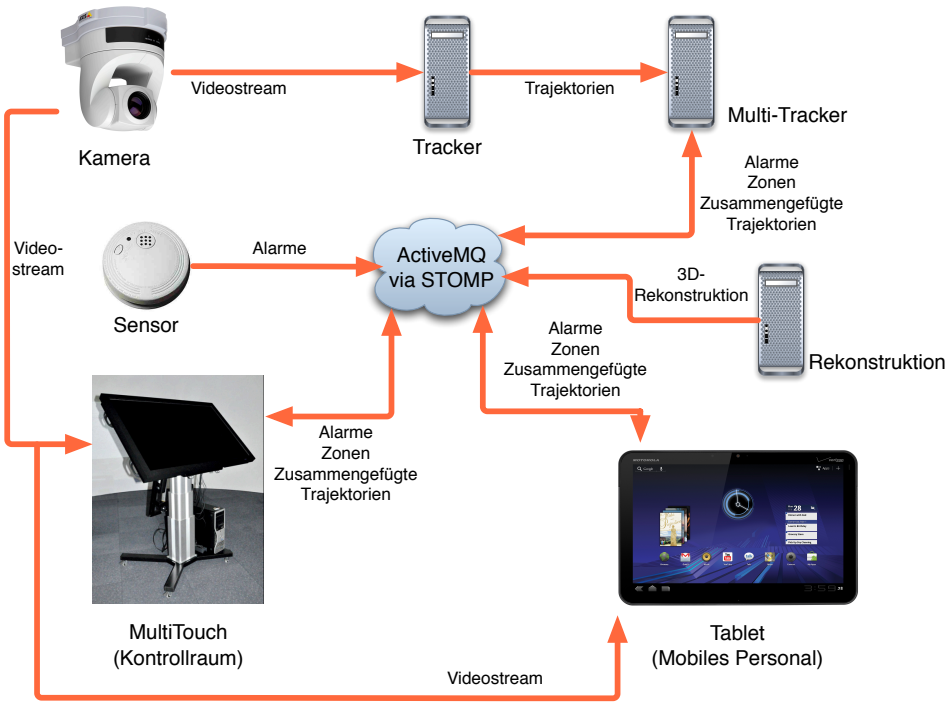


Abbildung 2.1.: Überblick über das CamInSens-System

Abb. 2.1 zeigt einen Überblick über die *CamInSens*-Architektur. Ausgangspunkte sind immer die Kameras, deren Videobild von einem *Tracker* verarbeitet wird. Dieser extrahiert menschliche Bewegungen aus den Kamerabildern und generiert erste Teiltrajektorien. Die Ergebnisse werden weitergereicht zu einem *Multi-Tracker*, welcher die Trajektorienteile zu kameraübergreifenden Trajektorien zusammenfügt. Auf diese Weise können Personenbewegungen über mehrere Kamera-Sichtfelder hinweg betrachtet werden, wobei jederzeit sichergestellt sein soll, dass eine Trajektorie jeweils eine Person symbolisiert. In der Software des Kontrollraums ist es möglich, *Zonen*

zu definieren, die nicht betreten werden dürfen, oder innerhalb einer gewissen Zeitspanne wieder verlassen werden müssen. Der *Multi-Tracker* empfängt diese Zonendefinitionen aus der Software des Kontrollraums und generiert daraus Alarme, sofern eine vorher definierte Zone betreten wird; in einem späteren Stadium des Projekts soll der *Multi-Tracker* darüberhinaus imstande sein, in den Trajektorienverläufen auffällige Verhaltensmuster zu erkennen und diese als Alarme zu melden. Ebenso generieren Kleinstsensoren wie Glasbruch- oder Feuersensoren eigene Alarme und senden diese direkt ohne Zwischenstationen in das Netzwerk. Es besteht außerdem die Möglichkeit, im Alarmfall eine *3D-Rekonstruktion* (siehe Abb. 6.3) des Täters anzufordern.

Die Software auf den mobilen Geräten besitzt die Möglichkeit, all diese Informationen abzurufen und zusätzlich Live-Videostreams direkt an den Kameras abzugreifen.

Abbildung 2.2.: Axis 214 PTZ, das im Projekt verwendete Kameramodell

2.1.1. Live-Kamerabild und Kamerasteuerung

Im Projekt wurden professionelle, schwenk- und zoombare Überwachungskameras (siehe Abb. 2.2) [AXIa] verwendet, die eine Vielzahl an Möglichkeiten anbieten, ihr Videobild ins Netzwerk auszuliefern. Die zwei für den Einsatzzweck interessantesten sind dabei *MPEG4* [MPE] sowie *MJPEG* [MJP]. *MPEG4* bietet den Vorteil, dass es ein Streamingverfahren ist, und dadurch auch bei niedrigen Netzwerk-Bandbreiten ein gutes Bild erreicht. Bei diesen Techniken wird neben der verwendeten Komprimierung oftmals ausgenutzt, dass sich der größte Teil einer Bildszene sich nicht von einem Bild zum nächsten komplett ändert. Daher werden volle Bilder, die so genannten *Keyframes*, nur in gewissen periodischen Abständen übertragen, die restlichen Daten beinhalten nur die inkrementellen Veränderungen von Bild zu Bild, woraus sich eine erneute Verringerung der zu übertragenden Daten ergibt. Nachteilig ist jedoch, dass es sich bei *MPEG4* um ein lizenzpflichtiges Produkt handelt, weswegen es nicht möglich war, eine kostenlose Abspielvariante auf dem mobilen Endgerät zu finden. Zusätzlich besitzt jede Kamera nur eine *MPEG4* Lizenz, so dass eine Streamauslieferung an zwei oder mehr mobile Geräte nicht möglich gewesen wäre [AXIb].

MJPEG, oder Motion JPEG ist hingegen ein Verfahren zur Bildübertragung, bei dem komplette JPG-codierte Bilder [JPGb] [JPGa] nacheinander übertragen werden. Es wird bei Anfang der Übertragung eine Verbindung geöffnet und die Bilder dann nacheinander über den gleichen Datenstrom gesendet, was das Videobild trotz kompletter Bilddaten flüssig erscheinen lässt.

2.1.2. Abrufen der Daten

Um die Informationen anzubieten, stand ein *ActiveMQ* - Nachrichtenserver [ACT] bereit, welcher den Nachrichtenaustausch zwischen den bereits vorhandenen Komponenten des Projekts bereitstellte. Als eine einfach zu realisierende Verbindungsmöglichkeit zu dieser *ActiveMQ*-Installation stellte sich das schlanke STOMP-Protokoll [STO] heraus, wodurch eine Kommunikation über XML-Nachrichten [XML] möglich wurde. Diese textbasierten Nachrichtenformate wurden zentral definiert und iterativ unter Einbeziehung aller Projektpartner (IVE mbH [IVE], Vitracom AG [VIT], Fraunhofer IOSB [IOSb] und Uni Hannover Arbeitsgruppen: SRA [SRAa], SIM [SIM], IPI [IPI],IKG [IKG]) an den Verwendungszweck angepasst.

2.2. Anwendungsfälle (Use-Cases)

Der erste Meilenstein des *CamInSens*-Projekts implizierte für die Software der mobilen Geräte vor allem die Erfüllung von drei Szenarien (Forbidden Zone, Restricted Zone, Diebstahl), auf welche bei der Entwicklung Rücksicht genommen wurde, da sie auch in einem allgemeinen Kontext durchaus realistische Anwendungsfälle abbilden. Im Folgenden werden die Szenarienabläufe erläutert.

2.2.1. *Forbidden Zone Szenario*

1. Es wird im Kontrollraum eine Zone definiert.
2. Eine Trajektorie (symbolisiert eine reale Person) läuft in die Zone.
3. Der Multi-Tracker generiert eine Alarmmeldung.
4. Der Kontrollraum empfängt eine Alarmmeldung.
5. Der Kontrollraum prüft, ob ein Fehlalarm vorliegt und leitet die Alarmmeldung ggf. an das mobile Personal weiter.
6. Das mobile Personal empfängt den Alarm und nutzt alle ihm vorliegenden Informationen zur Klärung.
7. Das mobile Personal bestätigt ggf. die Klärung des Alarms.
8. Der Kontrollraum empfängt eine Bestätigungsmeldung

Dieser Ablauf wird in Abb. 2.3 dargestellt.

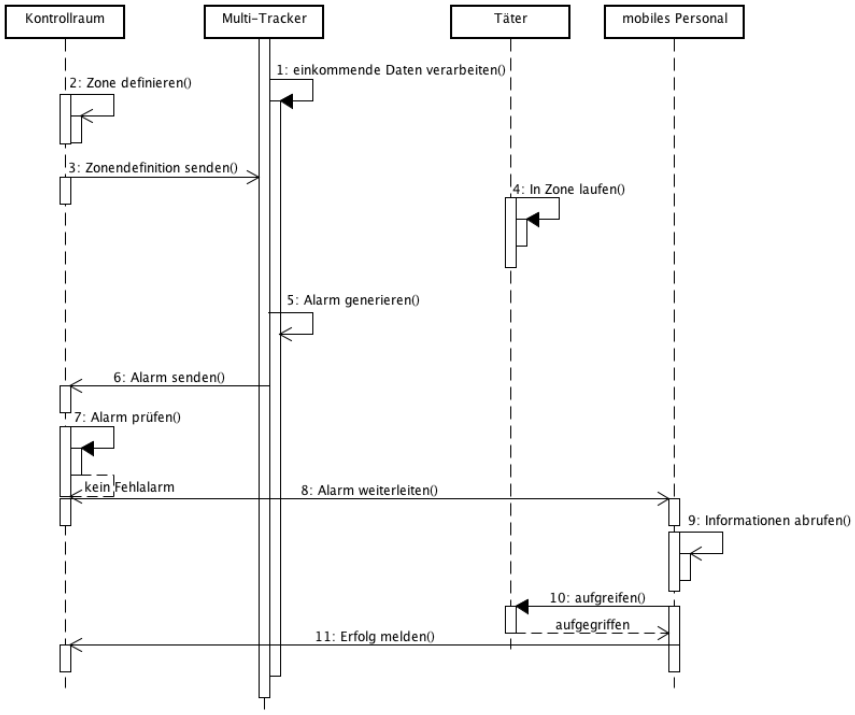


Abbildung 2.3.: Sequenzdiagramm Forbidden Zone Szenario

2.2.2. *Restricted Zone Szenario*

Prinzipiell wird im *Restricted Zone Szenario* das Gleiche abgehandelt wie im *Forbidden Zone Szenario*, nur mit dem Unterschied, dass sich eine Trajektorie für eine gewisse Zeitspanne in einer Zone aufhalten muss, bevor der Alarm auslöst. Da sich daraus keine neue oder veränderte Anforderung an die Realisierung der mobilen Sicherheitslösung ergibt, kann dieser Use-Case hier ausgelassen werden.

2.2.3. Kleinstsensor Alarm Szenario

1. Ein Kleinstsensor sendet einen Alarm
2. Der Kontrollraum empfängt eine Alarmmeldung.
3. Der Kontrollraum prüft, ob ein Fehlalarm vorliegt und leitet diesen ggf. an das mobile Personal weiter.
4. Das mobile Personal empfängt den Alarm und nutzt alle ihm vorliegenden Informationen zur Klärung.
5. Das mobile Personal bestätigt ggf. die Klärung des Alarms.
6. Der Kontrollraum empfängt eine Bestätigungsmeldung.

Im weiteren Verlauf dieser Arbeit wird nur noch das *Forbidden Zone Szenario* behandelt, da die Arbeitswege für eine Realisierung der Visualisierung und Interaktion bei allen drei Szenarien identisch sind.

Aus den gesammelten Informationen und den Randbedingungen des Projekts ergab sich nun schliesslich eine Anforderungliste.

2.3. Anforderungsliste

2.3.1. Anforderungen aus projektinternem Arbeitspaket (AP 1200) [ANF]

- Das mobile Gerät muss sich mit dem *CamInSens*-System verbinden können.
- Der Benutzer muss über unerlaubte Handlungen informiert werden können.
- Der Benutzer muss Informationen zur unerlaubten Handlung abrufen können.
- Der Benutzer muss einen erfolgreichen Einsatz zurückmelden können.

2.3.2. Weiterführende Anforderungen aus dem Kontext

- Das mobile Gerät muss sich mit dem System über das STOMP-Protokoll verbinden.
- Das mobile Gerät muss im Projekt definierte Nachrichten verarbeiten und versenden können.
- Das mobile Gerät muss Trajektorien und assoziierte Informationen darstellen können.
- Das mobile Gerät muss Alarme und assoziierte Informationen darstellen können.
- Das mobile Gerät muss Sensoren und assoziierte Informationen darstellen können.
- Das mobile Gerät muss von der Sicherheitszentrale definierte Zonen darstellen können.
- Das mobile Gerät muss 3D-Daten (z.B. der Rekonstruktion) darstellen können.
- Das mobile Gerät muss Trajektorien innerhalb einer angemessenen Berechnungszeit gruppieren können.
- Das mobile Gerät muss anormale Trajektorien innerhalb einer angemessenen Berechnungszeit erkennen können.
- Das mobile Gerät sollte Videostreams der Kameras darstellen können.

2.4. Anforderungen an die Hardware

Da kein äusserer Rahmen bezüglich Hardware, Plattform oder Visualisierung und Interaktion vorgegeben war, mussten die aktuellen Alternativen gesammelt und bewertet werden. Es wurden dann die jeweiligen Komponenten ausgewählt, die optimal die Zielführung des Projekts unterstützen.

Während der bereits erwähnten Besuche in mehreren Sicherheitszentralen wurden sowohl Interviews mit den Mitarbeitern der Kontrollräume sowie mit dem mobilem Personal geführt (siehe Anfang dieses Kapitels). Bei der Auswertung der Interviews fiel auf, wie wenige Daten das mobile Personal während ihrer Einsätze zur Verfügung hat. Dies ist vor allem den verwendeten Geräten, sowie deren mangelnden Sensoren und Schnittstellen geschuldet - die Kommunikation mit dem Kontrollraum bestand ausschliesslich in einer sprachlichen Komponente. Etwaige weitere Fähigkeiten der Geräte, wie beispielsweise *GPS* [GPS] in Mobiltelefonen zur Ortung der Mitarbeiter, wurden nicht genutzt.

2.4.1. Formfaktor und Interaktionsmöglichkeiten

Um diesen Mißstand zu beseitigen, und den Sicherheitsbeauftragten möglichst viele Informationen bereitstellen zu können, wurde die aktuelle Marktlage hinsichtlich einer für den Einsatzzweck geeigneten Gerätegattung untersucht. Um die zahlreichen per Netzwerk empfangenen Informationen möglichst ohne Wartezeiten visualisieren und bearbeiten zu können, musste ein Gerät mit einem schnell arbeitenden Prozessor und Grafikchip gewählt werden. Ebenso wichtig war die Anforderung, dass das Gerät eine schnelle, drahtlose Verbindung zu dem Projekt-Netzwerk aufbauen kann und insgesamt dabei noch derart kompakte Ausmaße besitzt, dass es komfortabel tragbar ist. Die Interaktion mit dem Gerät sollte möglichst intuitiv verlaufen, um eine schnelle Bearbeitung gewährleisten zu können. Aktuelle *MultiTouch*-Techniken mit kapazitiven Touchscreens [KAP] bilden in dieser Hinsicht die momentane Speerspitze in der massenproduzierten Unterhaltungselektronik, so dass im Endeffekt nur die Gerätegattung der neuartigen *MultiTouch*-Smartphones oder -Tablet-PCs übrig blieb. Diese sind genau darauf spezialisiert, bei kompaktem Formfaktor trotzdem relativ aufwendig gestalteten Grafikinhalt in Form von Spielen oder Videos ruckelfrei wiederzugeben, so dass die Rechen- und Grafikleistung ausreichend für den benötigten Kontext ist. Darüberhinaus ist durch die Selbstverständlichkeit einer ständigen Funknetzverbindung, die diese Geräte beispielsweise zum Internet herstellen, auch die Datenschnittstelle robust und schnell genug, wobei eine Verbindung wahlweise per *WLAN* [WLA] oder über Mobilfunknetze wie *UMTS* [UMT] und *GPRS* [GPR] erfolgen kann.

Der Unterschied zwischen Smartphone und Tablet-PC besteht vor allem in der Rechenleistung, der Bildschirmoberfläche und der Akkukapazität. Hinsichtlich der gewünschten Ziele fiel die Wahl auf Tablet-PCs, da diese generell mehr Rechen- und Akkuleistung als ihre kleineren Smartphone-Pendants besitzen. Durch die größere Bildschirm- und somit Interaktionsfläche ist es ebenfalls besser möglich, intuitivere und robustere Gesten zu finden als auf einem kleinen Gerät, auf dem meist bei Gesteneingaben nur maximal zwei Finger Platz finden; nachteilig bleibt jedoch der größere Formfaktor eines Tablets.

2.4.2. Betriebssystem

Es blieb nun die Wahl des Betriebssystems und somit auch indirekt, welche konkreten Geräte eingesetzt werden sollen. Zu Startzeit dieser Arbeit teilte sich die Tabletwelt lediglich in zwei große Lager auf: *iOS* [IOSa] und *Android* [AND].

Abbildung 2.4.: Panasonic ToughPad: Ein Android-basiertes Tablet, was speziell für die Outdoor-Nutzung konzipiert ist

iOS bietet mit dem *iPad* ein optisch sehr ansprechendes Gerät, was auch traditionell im Bedienkonzept bisher allen Android-Benutzeroberflächen überlegen ist. Für die Installation von eigenen programmierten Apps wird allerdings immer ein Entwicklerzugang fällig, der stetige Kosten mit sich bringt. Im Gegensatz dazu können bei *Android* die Applikationen einfach auf einem privaten Webspace abgelegt und dann per Browser installiert werden. Es wird dazu im Gegensatz zu *iOS* keine Prüfung von einer übergeordneten Instanz oder eine Signierung der Applikation benötigt. Signierung bedeutet in diesem Kontext, dass die Applikation mit einem Schlüssel versehen wird, der bei der Installation auf dem Gerät überprüft wird. Schlägt diese Prüfung fehl oder ist keine Signierung vorhanden, wird die Installation entweder komplett verworfen (*iOS*), oder kann nur durch Aktivieren von Entwickler-Einstellungen im Betriebssystem erfolgen (*Android*). Android glänzt ebenfalls in anderen Disziplinen durch Flexibilität: Es werden sehr viel mehr Video- und Bildformate als in *iOS* unterstützt, es gibt verschiedenste Möglichkeiten, eine Benutzeroberfläche nach eigenen Wünschen zu erstellen und schliesslich gibt es sehr viel mehr verschiedene Geräte zu kaufen. Hier kann nicht nur ausgewählt werden, wieviel Leistung, Speicher oder Bildfläche das Gerät besitzen soll, sondern es sind auch beispielsweise *Tough Devices* [TOU] erhältlich, die unter anderem im militärischen Bereich eingesetzt werden. Diese halten ohne Weiteres auch dem täglichen Gebrauch als Handwerkszeug stand, was mit dem *iPad* wohl, wenn überhaupt, nur durch Verwendung von Schutztaschen möglich wäre. Als letztes Argument gilt zu erwähnen, dass *Android* die populäre und komfortable Programmiersprache *Java* verwendet, und daher die meisten Programmierer keine neue Sprache wie *Objective-C* für *iOS* erlernen müssen.

Aufgrund der klaren Vorteile in Bezug auf die Anforderungen dieses Projekts fiel die Entscheidung schlussendlich auf die Verwendung von Android-Tablets mit dem neuesten Betriebssystem 3.0 (*Honeycomb*).

2.5. Sichtung einer geeigneten Grafikengine

Bedingt durch die Tatsache, dass das mobile Personal auch 3D-Rekonstruktionen aus dem System anfordern kann, wurde klar, dass zur Darstellung aller Informationen der dreidimensionale Grafikraum benötigt wird. Es war also nicht möglich, *Android*s hauseigene 2D-Grafik [CAN] zu verwenden, und es wurde sich für die Verwendung der ebenfalls unterstützten Grafikschnittstelle *OpenGL ES* [OPE], der mobilen Version von *OpenGL* entschieden.

Da es abzusehen war, dass man die zahlreichen Informationen nicht nur darstellen, sondern auch direkt mit ihnen interagieren will, wurde eine Möglichkeit benötigt, um bei Auswahl einer Bildschirmkoordinate die darunterliegenden 3D-Objekte abfragen zu können. Diese Technik wird in der Computergrafik meist als *Picking* beschrieben [SG09a]. Um *Picking* betreiben zu können, wird der *OpenGL* Selection-Modus benötigt, welcher aber in den OpenGL ES-Varianten zum Zeitpunkt dieser Arbeit nicht enthalten war. Somit blieb nur die Möglichkeit, entweder *Color Picking* [COLa] zu betreiben oder eine externe Bibliothek mit Picking-Funktionalität zu benutzen. Da externe Bibliotheken meist ausser *Picking* auch noch einen *Szenegraph* [SZE] und somit unter anderem auch die Vererbung von Transformationen mitbringen, wurde sich entschieden, eine externe Bibliothek zu benutzen. Die Vererbung von Transformationen ist ein wichtiges Mittel, um viele Objekte in einem Arbeitsgang zu transformieren, indem nur deren gemeinsames Elternobjekt transformiert wird. Genau dieser Fall trat in diesem Projekt auf, da alle zu visualisierenden Objekte immer relativ zu einem Referenzobjekt, wie zum Beispiel einer Kartendarstellung des Areals, positioniert werden mussten, was schliesslich die Entscheidung für einen Szenegraphen manifestierte.

Zusätzlich wurde klar, dass aus Gründen der Flexibilität und einer Verbesserung der Erwartungshaltung des Benutzers trotz Einsatzes einer Grafikbibliothek dennoch die Benutzeroberflächenelemente des *Android*-Frameworks benutzt werden sollten. Auf diese Weise kann der Nutzer seine bisherigen Erfahrung mit *Android* direkt auf die Realisierung der Sicherheitsapplikation anwenden, ohne die Benutzeroberfläche eines Drittanbieters kennenlernen zu müssen. Darüber hinaus kann zukünftig die 3D-Visualisierung einfach ausgetauscht werden, ohne dabei die komplette Interaktionsebene austauschen zu müssen, welche meist eng mit der jeweiligen Grafikbibliothek verknüpft ist. Diese Anforderung ist lösbar, indem die 3D-Grafiken in einem *Android View* [VIE] dargestellt werden, welcher variabel innerhalb einer *Android*-Applikation positionierbar ist.

Mit diesen gewonnenen Informationen wurde eine Sichtung von 3D-Grafikbibliotheken durchgeführt, bei der es galt, zuerst eine grobe Vorauswahl zu treffen. Die Vorauswahlkriterien waren hierbei:

- 3D-Unterstützung
- Gute Integrierbarkeit in Android
- Nutzung in Android View möglich
- Picking-Funktionalität
- Szenegraph
- Kostenlos

Übrig blieben nach dieser Vorauswahl die Bibliotheken *OSG*, *jmonkey*, *libgdx* und *jpct-AE*, welche nachfolgend verglichen werden.

2.5.1. OSG

Auch bekannt als OpenSceneGraph [OSG] bietet diese große Bibliothek einiges mehr als das, was im Rahmen dieser Arbeit benötigt wird. Interessant an dieser Bibliothek ist auch, dass das Fraunhofer-eigene *Virtual Reality* [VR]-Framework *Avango* [AVA] ebenfalls auf *OSG* aufsetzt und man somit auf lange Sicht *Avango* auf *Android*-Tablets etablieren könnte. *OSG* benutzt jedoch *C++* als Programmiersprache der Wahl und kann somit auf Android-Geräten nur durch Nutzung des *NDK* (Native Development Kit) [NDK] eingesetzt werden, welches durch das *JNI* (Java Native Interface) [JNI] und Cross-Kompilieren einer statischen Bibliothek die Möglichkeit bietet, auch beispielsweise *C* oder *C++*-Code auf einem *Android*-Gerät auszuführen.

Durch die Verwendung von *JNI* und der damit benötigten Kompilierzeit (die komplette Bibliothek muss statisch gelinkt werden) waren schnelle Iterationszyklen nicht möglich - ein Kompiliervorgang der Bibliothek dauerte bis zu drei Stunden. Das Debuggen des Quellcodes innerhalb der so erstellen Bibliothek stellte sich ebenfalls als schwer heraus, da bei Fehlern innerhalb des portierten Codes nur die Zeilen im *JNI*-Code angezeigt wurden, welche den Fehler verursachten. Da ein Rückschluss von *JNI*-Codezeile auf die wirklich fehlerhafte Zeile im *C++*-Code unmöglich ist und aufgrund der langen Kompilierzeiten wurde der Versuch der *OSG*-Implementierung schlussendlich abgebrochen.

2.5.2. jmonkey Engine

jmonkey [JMO] ist in *Java* implementiert und somit sehr gut in *Android* integrierbar. Die Bibliothek bietet alle benötigten Funktionen (3D, Picking, Szenegraph), es besteht aber nur die Möglichkeit, den 3D-Grafikinhalt bildschirmfüllend und nicht in einem *View* anzuzeigen. Es ist also nicht ohne Weiteres möglich, die *Android*-Benutzeroberflächenelemente zu verwenden oder die Grafikdarstellung neben anderen Informationsinhalten darzustellen.

2.5.3. libgdx

Diese Bibliothek [GDX] brachte auf den ersten Blick ebenfalls alle Funktionen von *jmonkey* mit, konnte jedoch in einem *Android View* [VIE] gestartet werden, was die Nutzung der Android GUI ermöglichte. Es stellte sich aber nach kurzen Implementierungsversuchen heraus, dass entgegen den ersten Informationen bei dieser Bibliothek kein *Szenegraph* und kein *Picking* implementiert war.

2.5.4. jpct-AE

jpct [JPC] ist eine sehr schlanke Bibliothek, die schliesslich alles mitbrachte, was erforderlich zur Durchführung der gestellten Aufgaben ist. Sie bietet *Picking*, einen Szenegraph mit Transformationenvererbung und integriert sich sehr gut in den Android-Kontext durch Anzeige in einem *View*. Es war zusätzlich möglich, auch 3D-Modelle in vielen gebräuchlichen Formaten zu importieren, weswegen in Summe die Wahl der Grafikengine auf jpct-AE fiel.

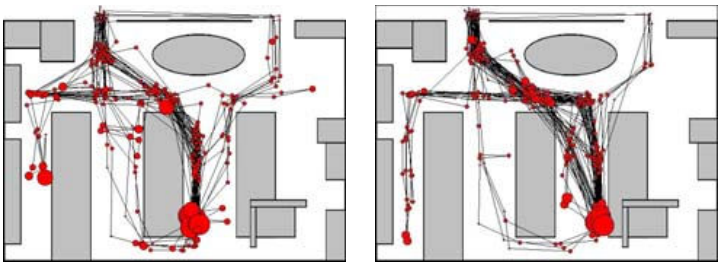
3. Visualisierung

Nach Festlegung der äusseren Rahmenbedingungen ging es nun darum, ein Konzept zu entwickeln, um die später erhältlichen Daten möglichst gut zu präsentieren. Herausforderungen bestanden insbesondere darin, die begrenzten Ressourcen auf Mobilgeräten zu beachten und effektiv zu nutzen. Zum Einen spiegeln sich diese Begrenzungen in der Leistungsfähigkeit der Hardware nieder, wie etwa bei Rechenoperationen, zum Anderen in der kleineren und, bedingt durch die verwendete Touch-Bedienung, anders als bei normalen Desktop-Rechnern zu nutzenden Bildfläche des Geräts. Hinzukommend bleibt zu erwähnen, dass das Tablet im Einsatz hauptsächlich im Gehen oder Stehen verwendet werden wird, was ebenfalls das Anforderungsprofil von einem normalen Büroarbeitsplatz-Szenario unterscheidet.

Auf Details bezüglich der Umsetzung der Benutzerschnittstelle wird im folgenden Kapitel 4 eingegangen; in diesem wird nun die Visualisierung der erhaltenen Daten behandelt.

3.1. Vorhandene Forschungsergebnisse

Um die Visualisierung möglichst effektiv zu gestalten, wurden zunächst bestehende Ergebnisse der Forschung hinsichtlich von Trajektorien-Darstellungen im Fundus der *ACM Digital Library* [ACM] gesichtet, die sich ebenfalls mit diesem oder einem ähnlichen Thema beschäftigten.



(a) Subject a (b) Subject b

Abbildung 3.1.: Visualisierung von Trajektorien zweier Personen nach [KKS+09]

Kurita et al. [KKS+09] zeichneten für diesen Zweck die Laufwege der Kollegen in ihrem Büro auf, um sie dann später auszuwerten. Ihr Ansatz besteht darin, zuerst Kreise zu zeichnen, deren Durchmesser abhängig von der Aufenthaltsdauer an einem Ort ist. Anschliessend entstehen durch eine Verbindung der Kreise die Trajektorien. Ein großes Problem dieser Lösung ist, dass eine Darstellung der Trajektorien von mehreren Personen in einem Bild nicht möglich ist, da diese nicht auseinandergehalten wer-

den können. Zusätzlich gehen bei diesem Ansatz viele Informationen durch Okklusion verloren (siehe Abb. 3.1).

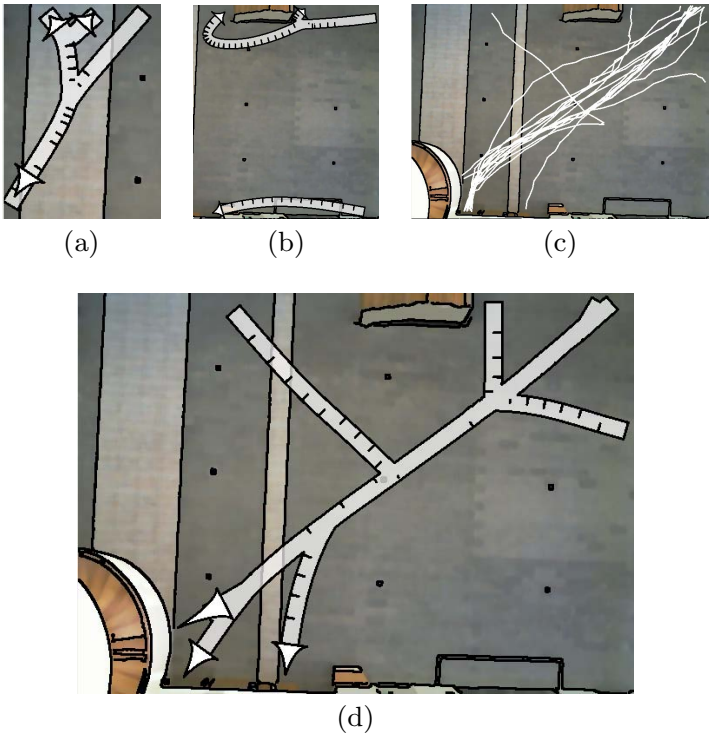


Abbildung 3.2.: Visualisierung von Trajektorien nach Höferlin et al.

Einen anderen Weg gehen Höferlin et al. [HHWH11], die mit ihrer Visualisierung versuchen, ähnliche Trajektorien zusammenzufassen, sowie Ausreißer zu entfernen und die Darstellung dadurch möglichst übersichtlich zu gestalten. In Abb. 3.2 können die einzelnen Stufen dieser Verarbeitung erkannt werden: c) beschreibt die Rohdaten ohne jegliche Optimierung oder Zusammenfassung, d) zeigt das Endprodukt mit zusammengefassten und von Störungen befreiten Trajektorien.

Einen Ausflug in die Welt des Sports unternahmen Pingali et al. [ea01], die Spielerbewegungen während Tennisspielen auswerteten (siehe Abb. 3.3). Durch die Farbe der Bildpunkte lässt sich dabei erkennen, welcher Spieler sich am Längsten an welchem Ort auf dem Spielfeld aufgehalten hat, so dass eine Aussage über den Stil des Spielers getroffen werden kann, wobei rot ein häufig und grün ein weniger häufig frequentierter Aufenthaltsort ist. Nachteilig stellt sich heraus, dass auch hier keine Darstellung mehrerer Personen auf einem Bild erfolgen kann, wenn sich die Personen nicht auf einem Spielfeld mit getrennten Spielbereichen aufhalten.

Zhong et al. [ZZTM10] benutzen einen Raum-Zeit-Würfel, um eine eventuell vorhandene Zeitachse zusätzlich zu visualisieren (siehe Abb. 3.4). Dies bietet natürlich den Vorteil, dass auch zeitliche Abhängigkeiten visuell dargestellt werden können, so dass ein Kreuzen zweier Trajektorien in der Visualisierung auch wirklich ein reales Aufeinandertreffen dieser zwei Personen symbolisiert. Diese Darstellungsdimension

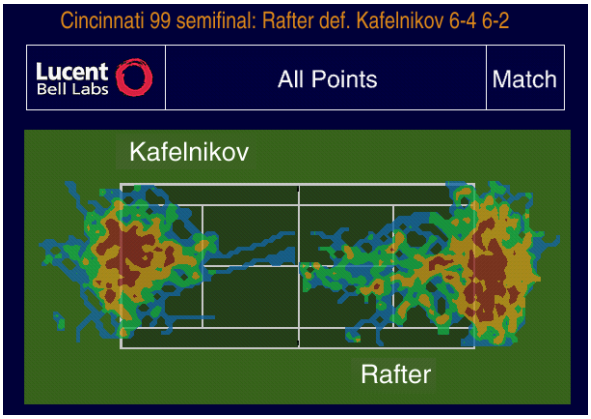


Abbildung 3.3.: Visualisierung von Trajektorien nach Pingali et al.

bieten alle anderen Visualisierungen nicht, jedoch wird der Raum-Zeit-Würfel sehr schnell unübersichtlich, wenn sich viele Personen zu unterschiedlichen Zeiten am gleichen Ort aufhalten.

Schlussendlich musste eine Visualisierung gewählt werden, die die Gegebenheiten des Projekts und die zusätzlichen Anforderungen gut berücksichtigt. Das Hauptmerkmal der Eingangsdaten ist, dass es einen abgeschlossenen Bereich gibt, den viele Personen durchkreuzen. Die Daten sollten von den Mitarbeitern schnell erfassbar sein und, wenn nötig, auch eine einzelne Person gut von anderen unterscheidbar sein.

Der Raum-Zeit-Würfel kommt daher nicht in Frage, da bei den erhaltenen Daten genau der Fall eintritt, dass viele Personen zu unterschiedlichen Zeiten den gleichen Raum durchqueren, was die Visualisierung sehr unübersichtlich werden lässt. Zudem erfordert die Zeit als dritte Dimension einen zusätzlichen Denkschritt bei der Erfassung des Sachverhalts, was einem Mitarbeiter in einem kontrollierten Arbeitsumfeld wie dem Kontrollraum zuzumuten ist, nicht aber einem mobilen Mitarbeiter, welcher äusseren Einflüssen und somit einem Konzentrationsdefizit ausgesetzt ist.

Kurzgefasst leisten es die übrigen Visualisierungen von Pingali und Kurita nicht, mehrere Trajektorien unterscheidbar auf einem Bild darzustellen, so dass schliesslich der Höferlin-Ansatz bedingt durch die gute Übersicht und damit schnelle Erfassung der Daten als die am Besten geeignete Vorlage für die Datendarstellung ausgewählt wurde.

3.2. Verwendete Daten

Als Testdatensatz mussten vergleichbare Daten gefunden werden, um die Entwicklung starten zu können. Zahlreiche Daten wurden in den IEEE Workshops 'Performance Evaluation of Tracking and Surveillance' [PETa] [PETb] [PETc], sowie in den IEEE Konferenzen 'Advanced Video and Signal based Surveillance' [AVS07] [AVS09] [AVSa] [AVSb] gefunden, jedoch bestanden die Daten aus diesen Konferenzen nur aus rohem Bild- oder Videomaterial. Die Extrahierung von Trajektorien aus Bildmaterial

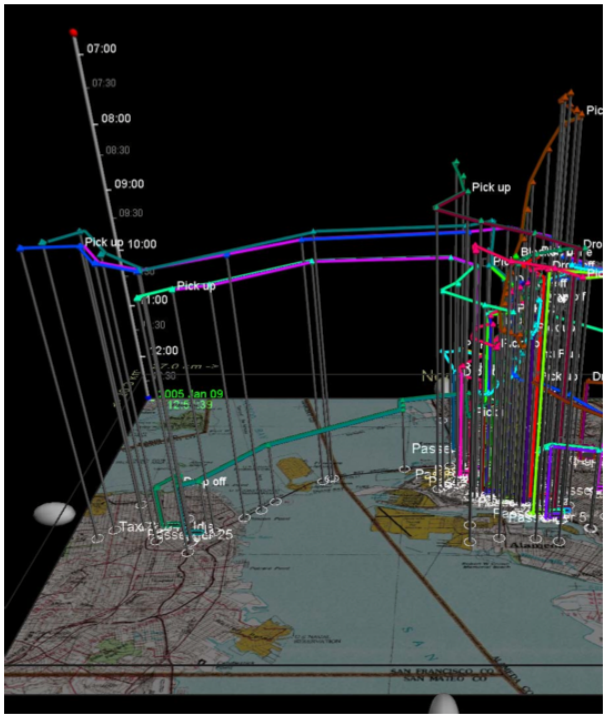


Abbildung 3.4.: Visualisierung von Trajektorien in einem Raum-Zeit-Würfel

ist jedoch nicht Teil dieser Arbeit, weshalb in anderen Quellen weiter recherchiert und schliesslich ein sehr geeigneter Datensatz gefunden wurde. [Edi] [Maj09]

Dieses Datenpaket umfasst erkannte Trajektorien von Personen, die in 2009 und 2010 durch den Innenhof (Forum) des Informatikgebäudes der Universität von Edinburgh liefen (siehe Abb. 3.5). Die Daten beinhalten mehrere Beobachtungsmonate mit rund 1000 Trajektorien pro Tag, was insgesamt einen Datenbestand von mehr als 92000 Trajektorien bildet. Wie diese Trajektorien aus dem reinen Bild- und Videodaten extrahiert wurden, wird in dieser Arbeit nicht näher untersucht.

3.2.1. Datenformat

Das Datenformat (siehe Abb. 3.6) ist in einer Textdatei erhältlich, und besteht aus einer Startzeile, die über die Anzahl der enthaltenen Trajektorien informiert, gefolgt von jeweils zwei Variablen für jede Trajektorie, 'Properties' und 'TRACK'. Jede Trajektorie besitzt einen eindeutigen Bezeichner, wie zum Beispiel R1 bis Rn.

Abbildung 3.5.: Der Innenhof des Informatikgebäudes der Universität Edinburgh

In der 'Properties'-Variable sind in folgender Reihenfolge, jeweils durch Leerzeichen getrennt, enthalten:

- Anzahl der Punkte in dieser Trajektorie
- Startzeit
- Endzeit
- Durchschnittsgröße des getrackten Gegenstands
- Durchschnittsbreite
- Durchschnittshöhe
- Durchschnittshistogramm

In der 'TRACKS'-Variable befindet sich:

- X-Wert Punkt 1
- Y-Wert Punkt 1
- Zeit Punkt 1
- X-Wert Punkt 2
- Y-Wert Punkt 2
- usw.

Es wird somit klar, dass die 'TRACKS'-Variable die wichtigste Information, nämlich eine Zeit-Weg Funktion für jede Trajektorie bereitstellt.

Dieses Format musste im Verlauf der Implementierung eingelesen und verarbeitet werden, jedoch so abstrahiert an die Visualisierung weitergegeben werden, dass die später per Netzwerk empfangenen Daten sich ebenfalls leicht integrieren lassen. Die Umsetzung dieser Anforderung wird in Kapitel 3.6: Implementierung beschrieben.

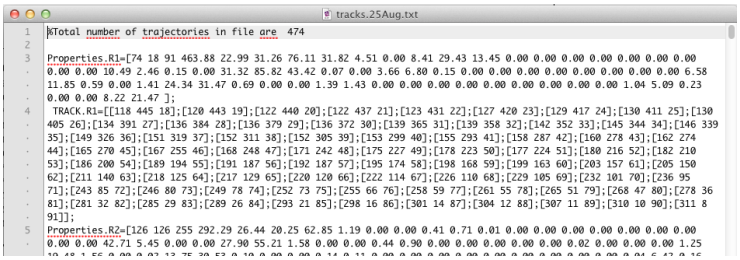


Abbildung 3.6.: Datenformat der Edinburgh-Forum Daten

3.3. Gruppierung von Daten

Der wichtigste Schritt, um Bewegungsdaten übersichtlich darzustellen und schnell erfassbar zu machen, ist eine Gruppierung von ähnlichen Bewegungen. Eine gängige Methode, um Gruppierungen von Daten zu erstellen, bei denen die Gruppenparameter im Vorfeld nicht klar sind, ist *Clustering*.

3.3.1. Clustering

Clustering ist der Vorgang, Objekte in sogenannte Cluster zu gruppieren, so dass innerhalb des Clusters eine hohe Ähnlichkeit und eine möglichst geringe Ähnlichkeit zu anderen Clustern besteht [HKP06]. Prinzipiell muss bei einem Clustervorgang immer entschieden werden, wie weit zwei Objekte voneinander entfernt sind. Diese Aufgabe wird mit Hilfe von verschiedenen Distanzalgorithmen gelöst, bei denen die euklidische Distanz [EUK] die wohl geläufigste ist. Stark vereinfacht werden nach der Distanzberechnung Objekte mit geringem Abstand untereinander zu Clustern zusammengefasst, eventuell mit bestehenden Clustern vereint und Clusterzentren gegebenenfalls angepasst.

Die Auswahl einer oder mehrerer geeigneter Clustermethoden für den betrachteten Anwendungsfall hängt von den Eigenschaften der Eingangsdaten, den Präferenzen der Distanzalgorithmen und der zur Verfügung stehenden Rechenleistung ab.

Die Trajektorien-Datenvektoren besitzen eine ungleiche Länge, was daraus folgt, dass manche Pfade länger sind oder, bedingt durch eine niedrigere Bewegungsgeschwindigkeit, öfter abgetastet werden. Diese Eigenschaft bereitet den meisten Distanzalgorithmen Probleme, so dass entweder:

1. Features aus den Trajektorien extrahiert werden müssen (beispielsweise geometrische)

   oder

2. die Datenvektoren auf gleiche Länge transformiert werden müssen (Resampling, Extra-/Interpolation)

   oder

3. ein Distanzmaß gefunden werden muss, was mit ungleicher Featurelänge umgehen kann (Hausdorff-Distanz oder andere)

Eine weitere Eigenschaft der Eingabedaten ist, dass die Trajektorien, und somit die Cluster, keine kreisähnliche Figur, sondern längliche Pfade formen. Viele Clusteralgorithmen können jedoch nur kreisförmige und keine arbiträr geformten Cluster erkennen, wodurch entweder geeignete Methoden gewählt oder die Eingangsdaten in andere Repräsentationen umgeformt werden müssen. Höferlin et. al. [HHWH11] arbeiten beispielsweise in ihrem Verfahren zur Trajektorienvisualisierung mit einer Transformation, welche die Trajektorien zuerst in einem Gitter quantisiert und dadurch in ein Datenwort umformt (siehe Abb. 3.7). Auf diese transformierte Daten wird schliesslich eine Funktion angewandt, mit der die Änderungskosten von einem Datenwort zum anderen berechnet werden. Diese Kosten dienen im Endeffekt als Distanzmaß, mit welchem verschiedene Clustermethoden umgehen können.

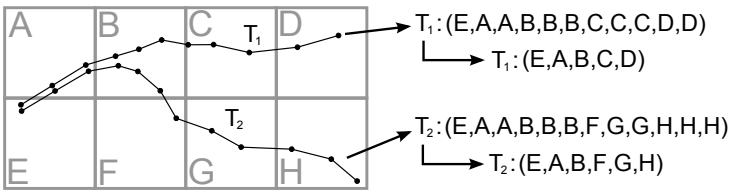


Abbildung 3.7.: Generieren einer String-Repräsentation durch Quantisierung einer Trajektorie in Zellen, [HHWH11]

Der zu clusternde Datenbestand soll während der Laufzeit ständig erneuert werden, was bedeutet, dass neue Trajektorien hinzukommen werden und alte eventuell gelöscht werden müssen. Dieser Prozeß sollte nicht zur Folge haben, dass der gesamte Datenbestand neu durchforstet werden muss, sondern er sollte in einer ökonomischen Weise erneuert werden können oder generell so schnell sein, dass die Ergebnisse in einer annehmbaren Zeit geliefert werden.

Als letzte, aber nicht weniger wichtige Einschränkung spielt es eine große Rolle, dass die Rechenleistung auf einem mobilen Tablet-PC einerseits sehr begrenzt ist, das Clustering aber möglichst schnell erfolgen soll. Es können folglich nur die Methoden mit der geringsten Komplexität (und somit die schnellsten) implementiert werden.

3.3.2. Auswahl Gruppierungsmethode

Es wurden verschiedene Clusteralgorithmen gesichtet, von denen *OPTICS* als sehr gut passend erkannt wurde, da der Algorithmus nur einen logarithmischen bis quadratischen Aufwand besitzt, aber trotzdem arbiträr geformte Cluster erkennt und updatefähig ist. Im Gegensatz zu *DBSCAN*, welcher ähnliche Eigenschaften besitzt, benötigt *OPTICS* jedoch keine Konfigurationsvariablen, auf die der Algorithmus bei Falschbelegung sensitiv reagieren könnte (siehe auch die im Rahmen dieser Arbeit auf Basis von [HKP06] erstellte Zusammenfassung in Tabelle 3.1).

Nach einer ersten Implementierung wurde die Clustermethode auf einen Ausschnitt eines Datensatzes angewendet, um zu testen, ob arbiträr geformte Cluster tatsächlich erkannt werden. Das Ergebnis machte jedoch schnell klar, dass dieser Algorithmus zu langsam für den späteren Produktiveinsatz sein würde. Es wurden rund einhundert Punkte einer einzelnen Trajektorie zur Berechnung übergeben, wofür *OPTICS* auf dem

Tabelle 3.1.: Vergleich Clusteralgorithmen

Algorithmus	Clusterform-erkennung	Komplexität	Updatefähig-keit
k-means k-median k-modes	kreisförmig	linear	ja
k-medoids	kreisförmig	quadratisch	ja
AGNES	kreisförmig	quadratisch	nein (nach Merge-Entscheidung kein zurück)
BIRCH	kreisförmig	linear	ja (Phase 2 muss aber wiederholt werden)
Chameleon	arbiträr	quadratisch	nein
DBSCAN	arbiträr	logarithmisch bis quadra-tisch	ja
OPTICS	arbiträr	logarithmisch bis quadra-tisch	ja (Einfügen in die Ord-nung)

mobilen Gerät jedoch rund vier Minuten benötigte. Es wurde zwar das erwartete Ergebnis ausgegeben, jedoch sind vier Minuten Wartezeit nicht tolerierbar für das spätere Einsatzszenario, vielmehr war ein in Sekunden messbares Ergebnis avisiert. Auch die Anwendung anderer Clustermethoden wie *DBSCAN* lieferten ähnliche Ergebnisse.

Um diesem Problem zu begegnen, wurde die *Hausdorff-Distanz* [HAU] implementiert, um einen Clusteralgorithmus mit linearem Aufwand, wie beispielsweise *k-means*, benutzen zu können, doch auch hier stellte sich kein Erfolg im Bereich der Performanz ein. Um die Distanzen von knapp fünfhundert Trajektorien (die Daten eines Tages) untereinander zu berechnen, wurden mit *Hausdorff* siebzehn Minuten benötigt, exklusive eines auf die Distanzwerte angewandten Clusteralgorithmus.

Aufgrund dessen wurde letzten Endes eine vereinfachte Version des Höferlin-Ansatzes implementiert, was schliesslich zu guten Ergebnissen und annehmbaren Berechnungszeiten führte. Das zu untersuchende Areal wird bei dieser Lösung in ein Gitter unterteilt und jeder entstehenden Zelle ein eindeutiger Name zugeteilt. Die Trajektorien werden nun abgelaufen und die Namen der dabei durchquerten Zellen konkateniert, wobei direkt hintereinander gelegene Mehrfachvorkommen von Zellen ausgefiltert werden (siehe auch Abb. 3.2). Schliesslich entsteht für jede Trajektorie eine stark komprimierte Repräsentation, die Aufschluss über den Verlauf der Trajektorie gibt. Der letzte Schritt besteht darin, Trajektorien mit gleichen Repräsentationen zu

gruppieren, wobei gleichzeitig nach absteigender Gruppengröße sortiert wird.

Diese Methode lässt den klassischen Clustering-Schritt aus, um die Berechnungszeit zu verkürzen. Tatsächlich können nun knapp fünfhundert Trajektorien innerhalb einer Sekunde gruppiert werden, was eine tolerierbare Wartezeit für den Benutzer darstellt. Nachteilig an diesem Algorithmus ist allerdings, dass der Nutzer die Größe des Gitters eventuell an neue Örtlichkeiten anpassen muss, um gute Ergebnisse zu erhalten. Da die Einsatzorte sich aber wohl nur extrem selten in ihrer Gestalt ändern werden, kann dies als einmalige Anfangskonfiguration angesehen werden.

3.4. Anomalienerkennung

Durch das Gruppieren der Personenbewegungen werden nicht nur dominante und häufig genutzte Pfade erhalten, sondern automatisch auch sich anormal verhaltende Trajektorien erlangt. Dies sind genau die Repräsentationen, die nicht mit anderen Trajektorien gruppiert werden konnten und somit eine Gruppe für sich allein bilden. In dieser konkreten Umsetzung wird gleichzeitig absteigend nach der Länge des Repräsentations-Strings gruppiert, denn eine Person, die nicht den direkten und kürzesten Weg geht, kann generell schon als verdächtig gekennzeichnet werden. Ein gutes Beispiel hierfür ist eine Trajektorie, die wiederholt zwischen zwei beobachteten Raumteilen pendelt oder Kreise durch das Gebäude läuft, denn dieses Verhaltensmuster könnte in einem Bahnhof beispielsweise von einem Taschendieb verwendet werden.

3.5. Visualisierung der Daten

Nach Analyse der bereits vorhandenen Visualisierungstechniken in Kapitel 3.1 wurde sich entschlossen, den Ansatz von Höferlin als Vorlage zu verwenden, was als ersten Schritt das Visualisieren von einzelnen Pfaden zum Ziel hatte.

3.5.1. Umgebungskarte

Um einen Bezug zwischen den Bewegungslinien und der Realität herzustellen, wurde eine Kartendarstellung des betrachteten Areals als texturiertes Viereck, welches immer im Hintergrund liegt, gezeichnet (siehe auch Abb. 3.8). Die Trajektorien werden dabei relativ zur Karte gezeichnet und jeweils im Szenegraph als Kindknoten an den Knoten der Umgebungskarte gehängt, wodurch bei einer gewünschten Translierung der gesamten Szene nur der Kartenknoten transliert werden muss.

3.5.2. Trajektorien

Eine gängige Technik, um Linien, und somit Pfade, in *OpenGL* zu zeichnen, ist die Verwendung des *Linestrip*-Primitivs [PRI] [SG09b]. Dieses Grafikprimitiv kann sehr schnell von der Grafikhardware gezeichnet werden und ist in seiner Breite variabel. Ebenfalls kann bei Bedarf *Anti-Aliasing* [SG09c] für diese Objekte angeschaltet werden, was die Kantenbildung bei diagonal im Bild verlaufenden Linien stark verringert. Es gab jedoch zwei Argumente, sich gegen eine Nutzung dieses Primitivs zu entscheiden: Einserseits unterstützt die gewählte Grafikbibliothek *jpct-AE* nur die Verwendung

von Triangle-Primitiven, und andererseits stellen Line-Primitive selbst bei angehobener Linienbreite keine Geometrie dar. Um eine *Picking*-Funktionalität und somit eine direkte Interaktion des Benutzers mit 3D-Objekten zu ermöglichen, ist jedoch eine Geometrie nötig, da sonst kein Schnitttest erfolgen kann.

Abbildung 3.8.: Visualisierung von Trajektorien

Es blieb entweder die Nutzung von vorgefertigten Geometrien der Bibliothek, wie zum Beispiel von Zylindern, oder das manuelle Zeichnen einer Geometrie durch mehrere zusammengefügte Dreiecke. Da das Hintereinanderhängen von vorgefertigten Zylindern bei spitzen Winkeln zwischen zwei Trajektorienteilstrecken eine Lücke mit sich bringt, die nicht ohne Weiteres sauber gefüllt werden kann, wurden die Bewegungspfade nur aus Dreieckprimitiven zusammengesetzt.

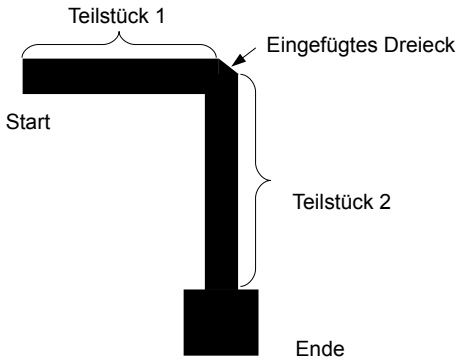


Abbildung 3.9.: Aufbau einer einzelnen Trajektorie

Ein Pfad besteht dabei prinzipiell aus einer Aneinanderreihung von Vierecken, die jeweils durch zwei Dreiecke gebildet werden, wobei auch die Breite jeweils pro Trajektorie variabel gestaltet werden kann. Die Anschlussstellen zwischen zwei Teilstücken werden durch Anfügen jeweils eines weiteren Dreiecks geschlossen, um einen glatten Verlauf des Pfades zu erreichen. Um die Laufrichtung der Trajektorien zu visualisieren,

wird ein Quadrat mit etwas breiterer Kantenlänge ans Ende der Trajektorien gesetzt. Bei Verlängerung der Trajektorie durch einen neuen Datenbestand verschiebt sich dieses Quadrat automatisch wieder ans Ende, so dass der Benutzer jederzeit informiert ist, in welche Richtung sich die beobachtete Person bewegt (siehe Abb. 3.9).

Damit jede Trajektorie möglichst gut von anderen unterscheidbar ist, wurden alle Bewegungslinien mit einer jeweils unterschiedlichen, zufällig gewählten Farbe eingefärbt (siehe Abb. 3.8). Die Priorität der Darstellung bei Objekten mit gleichem z-Wert erhält jeweils die Trajektorie, die zuletzt hinzugefügt wurde, so dass der Benutzer neben der guten Differenzierung zu anderen Trajektorien auch immer automatisch die aktuellsten Daten betrachtet.

3.5.3. Andere 3D-Objekte (Sensoren, Zonen, Rekonstruktionen)

Aus den Anforderungen (siehe 2.3) geht hervor, dass neben den Trajektorien auch noch viele andere Objekte visualisiert werden müssen, namentlich Sensoren, Zonen und Rekonstruktionen.

Sensoren sind laut Projektdefinition alle Instanzen, die einen direkten Beitrag im Netzwerk liefern [SRAb], so ist beispielsweise die Kontrollraumsoftware oder eine Überwachungskamera ebenfalls ein Sensor. Die Visualisierung als 3D-Objekt macht jedoch nur bei denjenigen Sensoren Sinn, die einerseits im beobachteten Raum liegen, und andererseits beim Verständnis der aktuellen Gefahrenlage helfen können. Es wurde sich daher entschlossen, nur alle Kleinstsensoren, mobilen Geräte und die Überwachungskameras darzustellen.

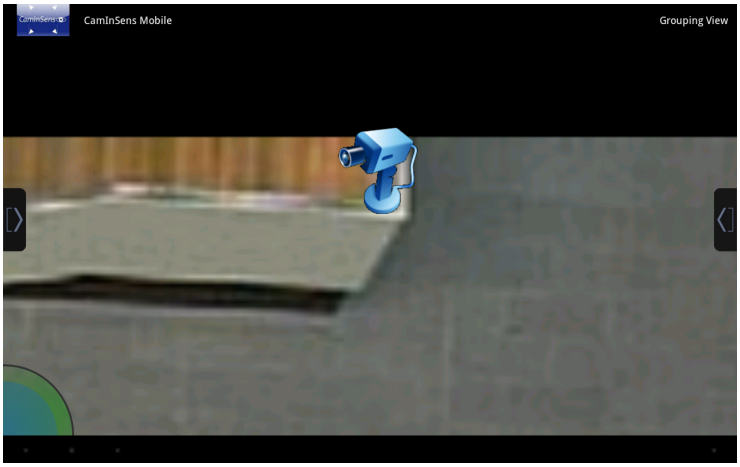


Abbildung 3.10.: Visualisierung eines Sensors, in diesem Fall eine Kamera

Die konkrete Visualisierung erfolgt dabei durch Verwendung von aussagekräftigen, teilweise transparenten Icon-Grafiken, die durch Billboard-Verfahren [SG09d] immer dem Betrachter zugerichtet sind. Es entsteht so gegenüber der Verwendung von 3D-Modellen ein Performanzvorteil, ohne die Visualisierung in einem großen Maß zu verschlechtern.

Zonen werden in der Kontrollraumsoftware durch ein Polygon definiert, und ebenso repräsentieren farbgefüllte Vielecke die Zonen in der Software auf den mobilen

Geräten. Die Farbe ist dabei abhängig davon, welche Art der Zone (Forbidden Zone, Restricted Zone) angelegt wurde, um dem Nutzer bereits implizit zu zeigen, welche Funktion das Objekt ausfüllt.

Abbildung 3.11.: Visualisierung einer *Forbidden Zone* (blaue Farbe)

Alle oben beschriebenen Objekte werden bei Bekanntwerden einer neuen Position automatisch an diese neue Stelle auf der Karte verschoben. Bedingt dadurch, dass alle Objekte auf einer Ebene und somit dem gleichen z-Wert gezeichnet werden, musste festgelegt werden, welche Objekte immer vollständig sichtbar sein sollen, und welche bei Bedarf auch überlagert werden können. Es wurde eine Reihenfolge festgelegt, in der die Arealkarte als hinterstes Objekt, danach die Zonen, gefolgt von den Trajektorien und schliesslich die Sensoren und 3D-Reproduktionen als vorderste Objekte gezeichnet werden. Auf diese Weise ist gewährleistet, dass die visualisierten Daten jederzeit schnell erfassbar sind (zum Beispiel wird eine Trajektorie, die durch eine Zone läuft, visuell über der Zone gezeichnet) und die zur Gefahrenerkennung wichtigen Objekte eher im Vordergrund zu finden sind.

3.5.4. Kippen der Szene

In einem späteren Stadium des Projekts sollen die Benutzer auch die Möglichkeit haben, 3D-Rekonstruktionen von mutmaßlichen Tätern anfordern zu können und sie auf dem mobilen Gerät darzustellen. Zum Zeitpunkt dieser Arbeit erhielt das mobile Gerät jedoch keine vollständige Reproduktion aus dem Netzwerk, sondern einen Zylinder, der den Umriß einer Person visualisieren sollte (siehe Abb. 3.12).

Dieser wurde in der Software an korrektem Ort und mit korrekter Höhe auf der Umgebungskarte dargestellt, so dass es für eine genaue Betrachtung der 3D-Rekonstruktion nötig war, die Szene entlang ihrer x-Achse kippbar zu machen. Darüberhinaus besitzt das Ankippen der Szene den Vorteil, dass der Anwender sich besser in das dargestellte 3D-Umfeld hineinversetzen kann, wodurch eine schnellere Erfassung der Szene möglich wird.

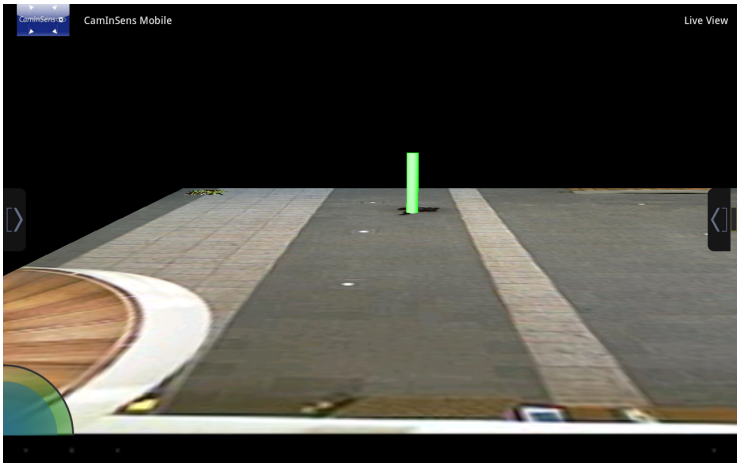


Abbildung 3.12.: 3D-Rekonstruktion bei gekippter Szene, symbolisiert durch einen Zylinder

3.5.5. Alarme

Die mitunter wichtigste Nachricht ist die Alarmnachricht, welche von der Sicherheitszentrale und dem *Multi-Tracker* an die Software auf den mobilen Geräten gesendet wird. Diese symbolisiert eine akute Gefahrensituation und muss daher sehr präsent visualisiert werden. Es wurde daher die Entscheidung gefällt, dieses Objekt immer im Vordergrund aller anderen Objekte einblenden zu können, was sehr gut mit den Benutzeroberflächen lösbar ist, die das Android-Framework mitbringt. Es ergibt sich durch Nutzung der Android-Bordmittel ausserdem die Möglichkeit, die Alarme anzuzeigen, auch wenn die 3D-Darstellung so weit skaliert oder verschoben ist, dass die Alarmposition nicht mehr im Sichtfeld liegt. Um auch mehrere Alarme gleichzeitig anzeigen zu können und dabei eine Reihenfolge erkennbar zu machen, wurde sich für eine Listendarstellung der Alarmobjekte entschieden, von wo aus dann weitere Informationen eingeholt werden können (siehe dazu Abb. 3.13 und Kapitel 3.5.6).

Damit der Benutzer auch nach vielen Alarmen weiterhin die volle Bildfläche zur Sichtung von relevanten Informationen zur Verfügung hat, sitzt die Alarmliste in einer ausfahrbaren 'Schublade' (im Folgenden *Panel* genannt) auf der rechten Seite des Bildschirms, welche sich im Alarmfall automatisch öffnet. Ein solches ausfahrbares *Panel* befindet sich ebenfalls auf der linken Seite des Bildschirms, von wo aus der Benutzer diverse Schalter betätigen oder Einstellungen durchführen kann (für näheres siehe Kapitel 4). Um dem Nutzer die Möglichkeit zu bieten, Alarme bereits anhand ihres Namens ignorieren oder bestätigen zu können, befinden sich innerhalb der jeweiligen Alarmzeile zwei Schaltflächen. Ein Berühren der Schaltfläche informiert die Sicherheitszentrale sofort über den Bearbeitungsstand des Alarms und entfernt selbigen aus der Bedienoberfläche des mobilen Benutzers.

Abbildung 3.13.: Visualisierung von Alarmen in einer Liste, die in einer ausfahrbaren 'Schublade' (Panel) sitzt

3.5.6. Assoziierte Informationen

Jedes 3D-Objekt sowie die Alarme beinhalten auch gewisse assoziierte Informationen, die zur Klärung einer Gefahrensituation hilfreich sein können. Diese Informationen können aus Gründen der Übersicht nicht alle zusätzlich zu jedem Objekt erscheinen, so dass es nur möglich ist, konkret angeforderte Informationen darzustellen. Um das zu ermöglichen, wurde sich wiederum des Android-Frameworks und dessen Dialog-visualisierung [DIA] bedient. Bei Anforderung der assoziierten Informationen (zur Methode der Anforderung siehe Kapitel 4) zu einem Objekt wird nun ein Dialog mit allen inhärenten Daten angezeigt (siehe Abb. 3.14).

Die im jeweiligen Objekt enthaltenen Daten sind:

- Trajektorie (und abgeleitete Objekte):
  - Der eindeutige Name der Trajektorie
  - sensorID: Der Sensor, der die Trajektorie zuletzt verlängert hat (typischerweise eine Kamera)
  - objectID: Das Gebäude oder der Raum, in der die Trajektorie erfasst wurde
  - timestamp: Der Zeitpunkt der letzten Verlängerung der Trajektorie
  - confidence: Die Vertrauenswürdigkeit im Sinne des Gefahrenpotentials der Trajektorie

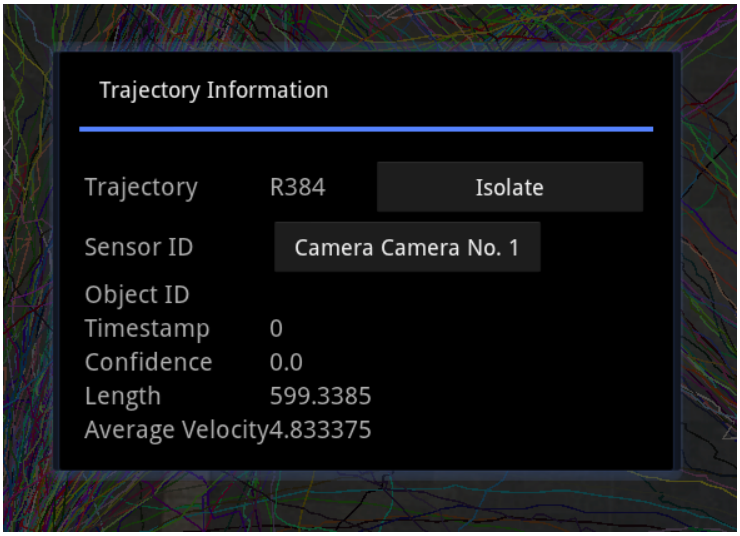


Abbildung 3.14.: Nähere Informationen zu einer Trajektorie, präsentiert in einem Dialog

- Sensor (und abgeleitete Objekte, z.B. Kamera):
  - sensor id: Der eindeutige Name dieses Sensors
  - position: Die aktuelle Position dieses Sensors
  - direction: Die aktuelle Richtung des Sensors (wenn vorhanden)
  - monitored space: Die Fläche, die dieser Sensor überwachen kann
  - ipv4 address: Die Netzwerkadresse des Sensors

Diese assoziierten Daten stammen ausschliesslich aus den Prozessen des *Tracker*s und *Multi-Tracker*s, enthielten im Zeitraum dieser Arbeit aber teilweise noch keine verwertbaren Daten, so dass beispielsweise Werte der confidence-Variable nicht zur Einfärbung von Trajektorien genutzt werden konnten.

Es war allerdings möglich, bereits bekannte Daten zu vernetzen, so dass aus einem Trajektoriendialog direkt der Dialog der aufzeichnenden Überwachungskamera aufgerufen werden kann. In diesem Dialog ist es dann wiederum möglich, den Live-Kamerastream darzustellen und sogar die Überwachungskamera fernzusteuern, wenn es unterstützt wird (siehe auch Abb. 4.4).

3.5.7. Tiled Video

Zusätzlich zu der Anzeige von Videodaten in einem einzelnen Dialog (siehe Abb. 3.15) ist es in der Anwendung möglich, eine gekachelte Übersicht aller momentan existierenden Videodaten darzustellen (siehe Abb. 3.17). In dieser Ansicht ist es nicht mehr möglich, die einzelnen Kameras fernzusteuern, da sonst das Ansichtsformat eines jeden Videostreams zu klein geraten würde; dafür ist es dem Nutzer jedoch möglich, eine effektive Übersicht der aktuellen Gefahren- und Auslastungslage zu erhalten.

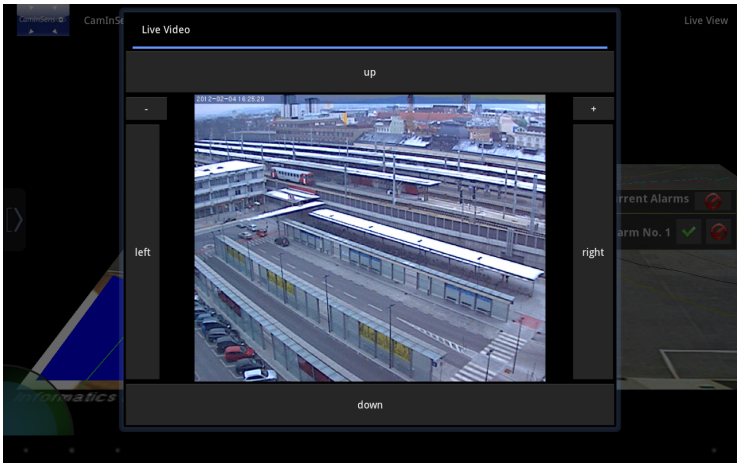


Abbildung 3.15.: Darstellung eines Videostreams mit Möglichkeit der Kamerasteuerung

3.5.8. Systemmeldungen

Es ist sinnvoll, den Benutzer immer über ablaufende Prozesse im Hintergrund zu informieren, sofern das zum Programmverständnis des Nutzers beiträgt. Als Beispiele können hier der Auf- und Abbau sowie Störungen in der Netzwerkverbindung, Bestätigungen von Gesteneingaben oder Benachrichtigungen über lang laufende Rechenprozesse genannt werden.

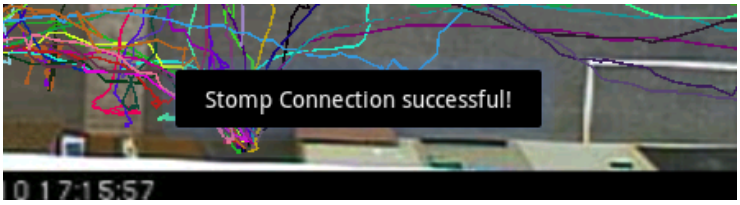


Abbildung 3.16.: Systemmeldung einer im Hintergrund hergestellten Netzwerkverbindung

Für all diese Meldungen wird jeweils ein kleines Benachrichtigungsfenster am unteren Rand des Bildschirms eingeblendet, welches auf keine Interaktion reagiert und selbständig ein- und ausblendet (siehe Abb. 3.16). Der Nutzer erhält so die Informationen, die er zur ständigen Anpassung seiner Erwartungshaltung gegenüber dem Programm benötigt, wird jedoch nicht in seiner aktuellen Tätigkeit unterbrochen oder gestört.

Abbildung 3.17.: Gekachelte Ansicht aller Videodaten

3.6. Implementierung

In den folgenden Sektionen wird näher darauf eingegangen, auf welche Art und Weise die beschriebenen Verhaltensweisen in der Programmierung umgesetzt wurden.

3.6.1. DrawObject

Um mit der benutzten Bibliothek *jpct-AE* ein 3D-Objekt zeichnen zu können, muss immer jeweils eine Instanz von Object3D [OBJ] erzeugt werden. Diese Instanziierung erfolgt, indem entweder vorgefertigte Minimalmodelle wie Zylinder oder Kugel verwendet werden, oder eine maximale Anzahl an Dreiecken festgelegt wird und das Objekt im späteren Verlauf mit Dreiecken gefüllt wird. Da diese Instanz von *Object3D* keine dynamische Anpassung der Dreiecksanzahl zulässt und das Objekt keine eigenen Zusatzparameter zur späteren Nutzung halten kann, musste ein umkapselndes Konstrukt erstellt werden, was bei Objektänderungen automatisch ein neues *Object3D* erzeugt und zusätzlich die Möglichkeit bietet, Zusatzinformationen abzuspeichern. Diese Klasse wurde *DrawObject* genannt.

Die erzeugte Struktur in Abb. 3.18 spiegelt wieder, wie alle benötigten Verhältnisse zueinander im Quellcode gelöst und optimal zusammengefasst wurden. Dabei ist zu erwähnen, dass Membervariablen, die nur für interne Zwecke benötigt werden, sowie triviale Getter und Setter in diesem Klassendiagramm ausgelassen wurden.

Die abstrakte Basisklasse *DrawObject* beinhaltet Offsetwerte, um die in Kapitel 3.5.3 festgelegte Objektüberlagerung zu erhalten. Dies ist nötig, da *jpct-AE* kein Deaktivieren des Tiefentests zulässt und somit auch kein Nacheinanderzeichnen der Objekte möglich ist, um die korrekte Verhaltensweise zu erreichen. Mit diesen Offsetwerten wird die Objektüberlagerung nun erreicht, indem die setSortOffset-Methode [SOR] der *Object3D*-Klasse benutzt wird. Diese greift dann wiederum auf die *OpenGL*-Methode glPolygonOffset [SG09e] zurück. Der Mechanismus bewirkt, dass Objekte mit einem höheren Offsetwert bei einem unentschiedenen Tiefentest automatisch im Vordergrund gezeichnet werden. Die Verwendung des Offsets hat auch den positiven Nebeneffekt,

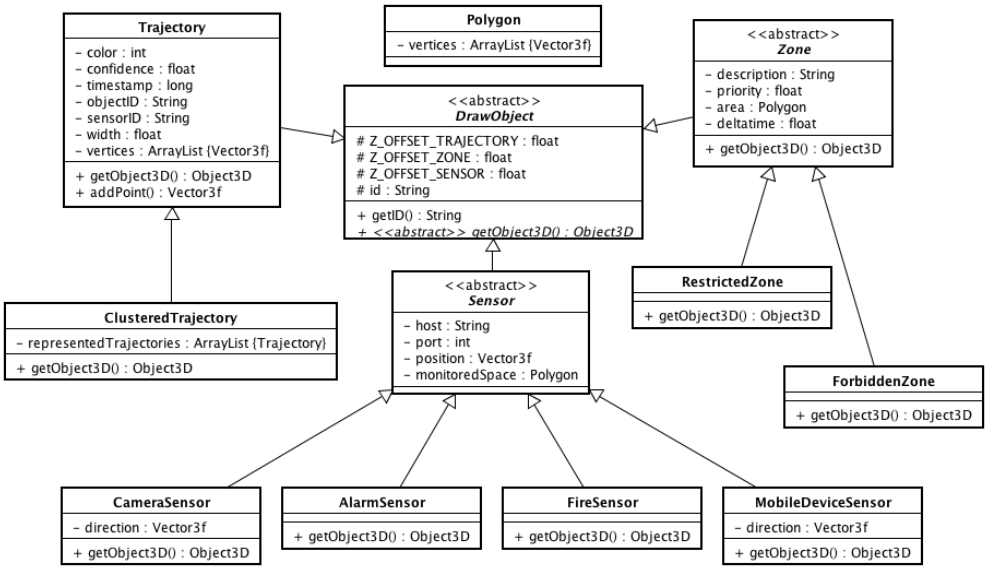


Abbildung 3.18.: Vererbung zwischen den 3D-Objekten

dass z-Fighting [ZFI] nicht mehr stattfindet. Neben den Offsetwerten verlangt die *DrawObject*-Klasse ausserdem von allen ihren Kindern, dass Sie ihren eindeutigen Namen festhalten und eine *Object3D*-Instanz ausliefern können.

Die von *DrawObject* erbenden Klassen *Trajectory*, *Sensor* und *Zone* beherbergen die jeweils assoziierten Informationen, die zur späteren Dialoganzeige, wie in Kapitel 3.5.6 erwähnt, benötigt werden. *Trajectory* verfügt zusätzlich über eine Methode *addPoint*, mit der die Trajektorie um eine Teilstrecke verlängert werden kann. Das *Object3D* wird schliesslich durch Auswertung der *ArrayList vertices* gebildet, in der alle mit *addPoint* angereichten Punkte gesammelt vorliegen.

ClusteredTrajectory beschreibt als Tochterklasse von *Trajectory* eine Gruppierung von Trajektorien. In der *ArrayList representedTrajectories* werden alle in der Gruppe enthaltenen Trajektorien festgehalten, zur Erzeugung des 3D-Objekts werden in dieser Klasse die Vertices einer repräsentierten Trajektorie benutzt.

In *Sensor* und deren Kinderklassen erkennt man, wie mit der optionalen Richtungsinformation umgegangen wird: Nur *MobileDeviceSensor* und *CameraSensor* implementieren diese, da sie im Gegensatz zu Feuer- oder Diebstahlsensoren nicht stationär sind. Die zu zeichnenden Objekte werden in den Sensorklassen jeweils nur in den untersten Ebenen der Hierarchie erzeugt, da jeder Sensor seine eigene Icongrafik besitzt.

Schliesslich folgt als Letztes die *Zone*-Klasse, bei der die *Object3D*-Elemente mit Hilfe der Variable *area* vom Typ *Polygon* als einfache, kolorierte Ebene erstellt werden. Die Farbgebung findet dabei direkt in *RestrictedZone* und *ForbiddenZone* statt, während die Ebene bereits in der Elternklasse angelegt wird.

Zusammenfassend bleibt zu sagen, dass die gesamte Hierarchie vor dem Hintergrund erstellt wurde, neue Objekte möglichst einfach hinzufügbar zu machen und den Wartungsaufwand bei Änderungen an beliebiger Stelle der Hierarchie klein zu halten. Dies ist dadurch gelungen, dass einerseits Parameter, die für alle gelten, möglichst weit oben in der Hierarchie angesiedelt sind, und andererseits objektspezifische Parameter möglichst weit unten eingegliedert wurden. Wenn nun beschlossen wird, dass der eindeutige Name aller Zeichenobjekte ab sofort eine Zahl sein soll, bedarf es nur einer Änderung des Membervariablentyps von *id* in der *DrawObject*-Klasse. Gibt es zukünftig einen neuen Sensor oder eine neue Zone, so reicht es, eine neue Klasse von *Sensor* beziehungsweise *Zone* abzuleiten und die Methode *getObject3D* zu implementieren. Ist zuletzt gewünscht, dass die Kameras in Zukunft ihre Aussentemperatur übermitteln, so muss auch hier nur eine Klasse um eine Membervariable erweitert werden.

3.6.2. DrawObjectDatabase

Da zum Zeitpunkt dieser Arbeit keine zentrale Datenbank existierte, mit der es möglich ist, alle aktiven oder vorhandenen Instanzen innerhalb des Netzwerks abzufragen, mussten diese Daten gesammelt und lokal vorgehalten werden. Die Netzwerknachrichten fliessen jeweils nur einmal an die zu diesem Zeitpunkt mit dem Nachrichtenserver verbundenen Geräte, so dass eine Datenbankstruktur notwendig wurde, die programmweit abfrag- und änderbar ist.

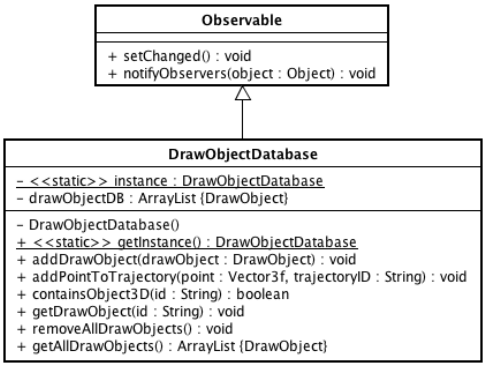


Abbildung 3.19.: Klassendiagramm mit den wichtigsten Funktionen der DrawObjectDatabase

Gelöst wurde diese Anforderung in der *DrawObjectDatabase*-Klasse. Die Implementierung erfolgte mit Hilfe eines *Singleton*-Patterns [Gam95a], um die Datenbank global verfügbar zu machen und nur eine Instanz dieser Klasse zu erlauben. Auf diese Weise können alle Aufrufer sicher sein, die richtige Datenbankinstanz zu erhalten. Es wurde eine *ArrayList* als Membervariable angelegt, welche die *DrawObject*-Instanzen beinhaltet, und von öffentlich erreichbaren Methoden aus veränderbar ist. In diesem Zusammenhang existieren auch öffentliche Komfortfunktionen, um beispielsweise mit *addPointToTrajectory(Vector3f point, String trajectoryID)* auf einfache Weise eine Trajektorie um einen Punkt zu erweitern, sogar wenn nur der eindeutige Name

und nicht die konkrete Instanz der Trajektorie vorhanden ist. Weitere Beachtung galt der Threadsicherheit, denn die Datenbank sollte später auch aus allen verschiedenen Threads [THR] der Anwendung robust abrufbar sein. Es wurden hierfür konsequent *Iterator* für das Durchlaufen der Datenbank benutzt, und an den benötigten Stellen das *synchronized*-Schlüsselwort eingesetzt.

Das *synchronized*-Schlüsselwort [SYN] gibt an, dass ein bestimmter Codeabschnitt oder eine Variable nicht von zwei Aufrufern gleichzeitig verwendet werden darf und sperrt somit die Benutzung für einen Aufrufer, bis der andere den synchronisierten Bereich verlassen hat. In diesem Fall wurde ein synchronisierter Block verwendet, um den Synchronisierungsbereich möglichst klein zu halten und nicht die Methode, sondern die Datenbankliste als kritisches Objekt zu markieren (siehe Listing 3.1). Auf diese Weise ist sichergestellt, dass ein Aufrufer, der aus einem anderen Thread und einer anderen Methode die Datenbank verändert, sich nicht mit den Tätigkeiten eines aktuellen Aufrufers überschneidet.

Der *Iterator* und das gleichnamige Pattern [Gam95b] abstrahieren die zugrundeliegende Datenstruktur weg vom Aufrufer. Es ist so möglich, die Datenstruktur zu durchlaufen, ohne Kenntnisse über die genaue Implementierung der Struktur zu haben, was ein einfacheres Abändern der Datenstruktur ermöglicht. Die Benutzung von Iteratoren in Java beim Durchlaufen einer Liste hat ausserdem den Vorteil, dass bei gleichzeitigem Zugriff aus zwei Threads eine *ConcurrentModificationException* ausgelöst wird, was einem den Rückschluss liefert, dass an einer Stelle im Code die Threadsicherheit nicht eingehalten wurde. Dieser Mangel macht sich ansonsten meist erst dann bemerkbar, wenn durch Timingprobleme unplausible Daten zustande kommen, die sehr schwer zu debuggen sind.

Listing 3.1: synchronized-Block

```
1 public void addDrawObject(DrawObject drawObject) {
2         if (drawObject != null) {
3                 synchronized (drawObjectDB) {
4                         deleteDuplicate(drawObject);
5                         drawObjectDB.add(drawObject);
6                         notify(drawObject);
7                 }
8         }
9 }
```

Damit die Datenbankkonsumenten über etwaige Änderungen sofort informiert werden, und um eine stärkere Abkapselung der Datenbank zu erreichen, wurde zusätzlich das *Observer*-Pattern [Gam95c] eingesetzt. Mit Hilfe dessen ist es einem Konsumenten möglich, sich an der Datenbank für Änderungsnachrichten zu registrieren, worauf er im weiteren Verlauf Aktualisierungen der Datenbank direkt in seiner Klasse erhält. Es ist somit nicht mehr notwendig, die Datenbank ständig abzufragen und auf Änderungen zu vergleichen. Darüber hinaus ist bei einem Austausch der Datenbank nur eine Schnittstelle nachzubilden, um das gleiche Verhalten wiederzuerlangen, was die Erweiterbarkeit des Programms verbessert. Die Implementierung dieses Patterns gestaltet sich in Java denkbar einfach: Die zu überwachende Klasse muss lediglich von *Observable* ableiten und ihre Konsumenten mit einem Aufruf von *setChanged()* und anschliessendem *notifyObservers(Object object)* über eine Änderung informieren.

Die *Observer* ihrerseits implementieren das *Observer*-Interface und müssen dadurch eine Methode *update(Object object)* bereitstellen, in welcher das in der *notifyObservers*-Methode übergebene Objekt dann empfangen und verarbeitet wird. In dem hier betrachteten Fall empfangen die *Observer* von *DrawObjectDatabase* also einzeln alle *DrawObject*s, die sich in der Datenbank in jedweder Form geändert haben.

3.6.3. Live-Kamerabild

MJPEG (siehe Kapitel 2.1.1) wurde als Methode zur Videodarstellung der Software implementiert, welche auch bei mehreren Zugriffen auf eine Kamera und schlechter Netzwerkqualität gut funktionierte. Eine Steuerung der Kameras wurde ebenfalls mit Hilfe des in der Kamera integrierten Webservers ermöglicht, welcher zu diesem Zweck speziell geformte Anfrageadressen erwartet. Diese werden bei Empfang ausgewertet und steuern die Kamera in Neig- und Schwenkrichtung oder Zoom. Die Anfragen sind nur mit fester, unkontrollierbarer Schrittweite möglich, so dass es nicht zu realisieren war, dass die Kamera mit dieser Methode sanft von einem Punkt zu einem anderen durchschwenken kann. Aufgrund dessen wurde im Bedieninterface keine Touchfunktionalität zur Auswahl bestimmter Punkte, sondern nur Schaltflächen zur Steuerung der Kamerabewegung eingebaut (siehe Abb. 3.15).

3.6.4. Systemmeldungen komfortabel benutzbar machen

Die Systemmeldungen wurden mit Hilfe der *Toast*-Funktion [TOA] von Android angezeigt. Diese benötigt allerdings eine Android *Context* Instanz [CON], um auf die globalen Umgebungsvariablen der Anwendung und somit den Grafiklayer zurückgreifen zu können. Da diese Umgebungsvariablen beispielsweise in einer Netzwerkklasse nicht erreichbar sind, es aber gerade dort sinnvoll ist, Systemmeldungen anzuzeigen, wurde ein System entwickelt, um die *Toast*-Nachrichten unabhängig von einer *Context*-Instanz aufrufen zu können.

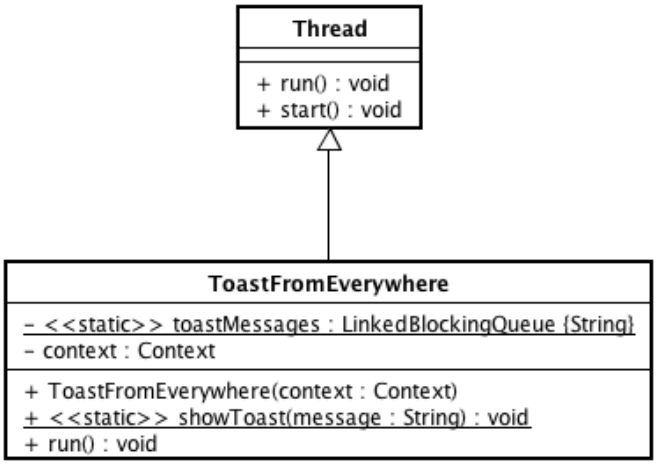


Abbildung 3.20.: ToastFromEverywhere ermöglicht das Darstellen von Systemmeldungen, ohne eine Context-Referenz zu besitzen

Das System besteht darin, bei Start der Anwendung einen Hintergrundthread zu starten, welcher in einer Endlosschleife immer wieder überprüft, ob ein neues Element in einer statischen, global erreichbaren *Queue* enthalten ist. Die *Queue* wird um ein Element erweitert, indem ein Aufruf der statischen Methode *showToast(String message)* erfolgt. Die Funktion eines Threads erfüllt eine Klasse, wenn sie von *Thread* ableitet und dann die *run()*-Methode überschreibt. Die *run*-Methode muss dabei alle Ausführungen enthalten, die in dem parallel startenden Thread laufen sollen. Anschliessend wird der Thread gestartet, indem die Klasse instanziiert und die ebenfalls von *Thread* geerbte *start()*-Methode ausgeführt wird. Da beim Start des Hintergrundthreads die Referenz zu dem aktuellen *Context* übergeben wird, kann nun bei Feststellung eines neuen Listenelementes durch *toastMessages.take()* [TAK] die Nachricht dargestellt werden.

4. Interaktion im mobilen Kontext

Neben einer effektiven Visualisierung der Daten wurde parallel ein Interaktionskonzept entwickelt und umgesetzt, was auf die Besonderheiten der eingesetzten Gerätegattung und die Randbedingungen des *CamInSens*-Projekts Rücksicht nimmt. Zu diesem Zweck wurden zuerst aktuelle Forschungsergebnisse gesichtet und auf Tauglichkeit überprüft.

4.1. Vorhandene Forschungsergebnisse

Ein Problem von *MultiTouch*-Gesten ist, dass Gestenpakete einer Software oft von Designern oder Entwicklern geschnürt werden, und somit meist mit technischen Hintergedanken oder allein durch Erfahrungswerte entstehen. So ist es nur natürlich, dass ein Programmierer bei der Wahl einer Geste zuerst an die Umsetzbarkeit auf der Softwareseite denkt, und nicht unbedingt die intuitivste Geste für einen bestimmten Zweck wählt. Ebenso ist es beispielsweise für Designer durch individuelle Vorbelastungen hinsichtlich der Erfahrungen mit *MultiTouch*-Geräten schwierig, neue und intuitive Gesten zu finden.

Um diesen Mangel zu beseitigen und komplett natürliche Gesten zu finden, werteten Wobbrock et al. [WMW09] in ihrem Beitrag über 1000 natürlich gewählte Gesten von 20 Benutzern aus, die vorher noch keinen Kontakt zu *MultiTouch*-Geräten hatten. Die Vorgehensweise des Tests bestand darin, den Probanden auf einer *MultiTouch*-Projektionsfläche eine Bewegung oder Manipulation von virtuellen Objekten vorzuspielen und die Testpersonen daraufhin eine Geste zeichnen zu lassen, welche als intuitiv für diese Manipulation empfunden wird. Es wurden 27 Animationen ('Referents') im Vorfeld festgelegt, die allesamt gängige Manipulationsoperationen wie Bewegen, Rotieren, Vergrößern/Verkleinern, Auswählen oder Vor/Zurück abbilden sollten. Durch eine statistische Auswertung der erhaltenen Ergebnisse konnte schliesslich ein von den Benutzern erstelltes Gestenpaket gewonnen werden (siehe Abb. 4.1).

Einen anderen Ansatz verfolgten Hinrichs et al. [HC11], die Benutzer und deren Gesten beobachteten, während Sie eine öffentlich zugängliche *MultiTouch*-Informationsfläche benutzten. Viele Gesten waren dabei schon im Vorfeld in der Informationsfläche implementiert, und durch gleichzeitige Analyse von Video- und Gestendaten konnten einerseits die bestehenden Gesten bewertet, und andererseits neue Gesten gewonnen werden. Vor allem durch Beobachtung der Gesichtszüge konnte dabei entnommen werden, ob der Proband mit der Wirkung der gerade durch ihn getätigten Gesten einverstanden war. Auch in dieser Studie kamen knappe 1000 Gesteneingaben zusammen, die im Nachhinein klassifiziert und klaren Intentionen zugeordnet werden konnten. Interessant an der Studie ist zusätzlich, dass die Hälfte der getesteten Nutzer sich hier aus Kindern zusammensetzte, und so die Ergebnisse eine noch bessere Repräsentation darstellen. Die erhaltenen Gesten gestalteten sich in dieser Untersuchung

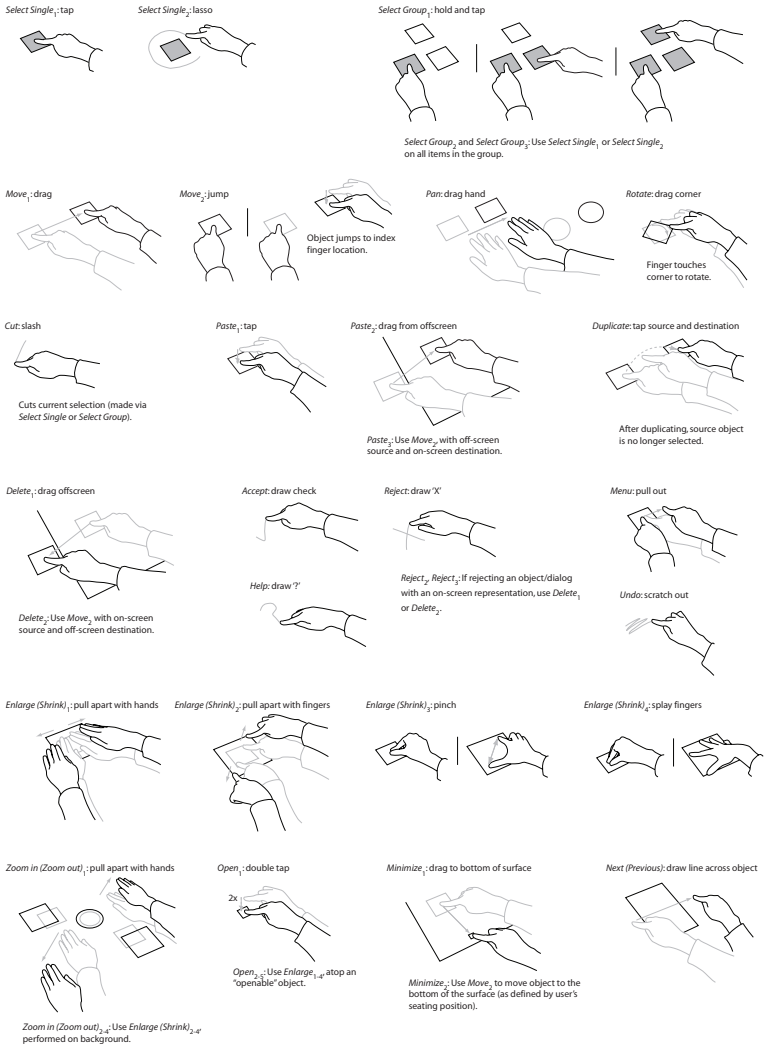


Abbildung 4.1.: Ein von Benutzern erstelltes Gestenpaket, Quelle: [WMW09]

sehr ähnlich zu denen von *Wolbrock et al.*.

Beide Untersuchungen entdeckten zudem, dass die Anzahl der auf der Touch-Oberfläche aufliegenden Finger keinen großen Unterschied für die Intention einer Geste macht, vielmehr wurde die Anzahl der Finger durch die Benutzer meist dann erhöht, wenn mehr Objekte auf einmal bewegt werden sollten und somit eine 'schwere' Tätigkeit erledigt wurde. Für die Intention einer Geste kommt es daher mehr auf andere Eigenschaften, wie die Änderungen der Finger-zu-Finger-Distanzen oder Aufsetz- und Abhebebewegungen ('Tap') an. Es ist daher sinnvoll, auf der technischen Ebene mehrere Gesten für eine Intention zu realisieren, um die verschiedenen Benutzerverhaltensweisen optimal abzudecken.

Die Ergebnisse dieser beiden Ansätze wurden nun im Folgenden dazu verwendet, um ein für den mobilen Einsatz passendes Gestenpaket zu entwickeln.

4.2. Konzept

Die wichtigsten Parameter, die es zu berücksichtigen galt, waren der konkrete Einsatzzweck sowie die Bedienmöglichkeiten des Tablets. Die Software soll es mobilem Sicherheitspersonal ermöglichen, Gefahrensituationen zu erkennen und durch Datenabruf wichtige Informationen zur Lösung des Problems zu erhalten. Da das mobile Personal die verwendeten Geräte ausschliesslich im Gehen oder Stehen benutzt, und somit eine Hand exklusive des Daumens zum Halten des Geräts benötigt wird, schieden zweihändige Gesten von Anfang an aus. Ebenso sind komplizierte MultiTouch-Gesten für den vorgesehenen Zweck ungeeignet, da die Benutzer im späteren Gebrauch unter Zeitnot und eventuell von einer Menschenmenge umgeben auf Situationen reagieren müssen. Wiederholte Eingaben von Gesten aufgrund von Fehleingaben oder falsch erkannte Gesten sollten folglich vermieden werden; vielmehr sollte das Gerät möglichst bei der ersten Benutzereingabe das gewünschte Ergebnis liefern, was vor allem dann der Fall ist, wenn möglichst wenige Finger das Gerät steuern.

Darüberhinaus war zu beachten, dass die Bildfläche eines Tablet-PCs sehr viel kleiner ist als die eines normalen Desktoprechners, so dass die angebotene Fläche effektiver sowohl für die Visualisierung als auch für die Interaktion genutzt werden muss. Eine gleichzeitige Anzeige aller Bedien- und Einstellungsoptionen des Gerätes ist daher nicht möglich, und ebenso ist nicht genügend Fläche für ausladende *MultiTouch*-Gesten vorhanden.

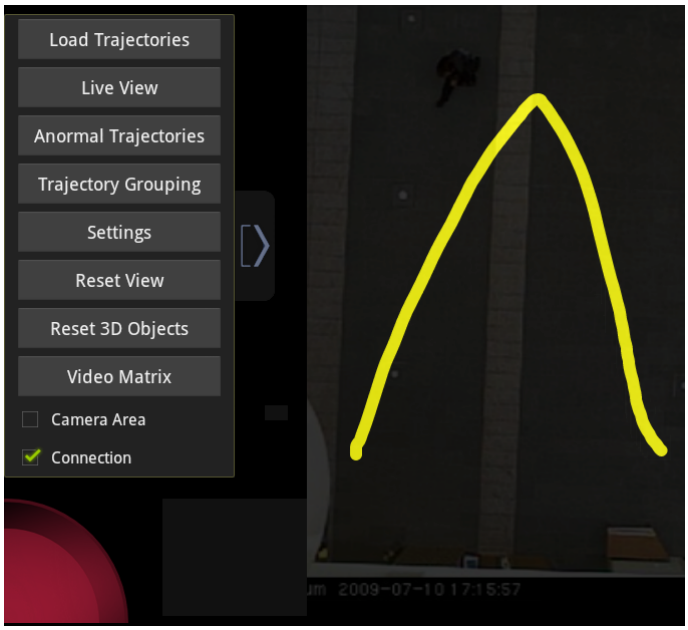


Abbildung 4.2.: Schaltflächen in einer ein- und ausfahrbaren 'Schublade' (Panel), daneben gezeichnete SingleTouch-Geste

Die beiden oben genannten Probleme wurden gelöst, indem Schaltflächen für wichtige Umschaltaktionen benutzt wurden, welche sich in einer ein- und ausfahrbaren 'Schublade' (im Folgenden: *Panel*) am Rand des linken Bildschirmrands befinden (sie-

he Abb. 4.2, linke Bildseite). Durch die Möglichkeit, die Schaltflächen komplett verstecken zu können, bleibt dem Benutzer die volle Bildfläche zur Analyse der visualisierten Daten. Bei Bedarf kann der Nutzer die Buttons wieder sichtbar machen und eine Aktion anstossen oder die Einstellungen der Software ändern. Zusätzlich ist es dem Anwender möglich, eigene *SingleTouch*-Gesten zu definieren, welche die gleichen Funktionen wie die Schaltflächen ausführen (Beispiel einer Geste siehe Abb. 4.2, rechte Bildseite). Diese Gesten sind absichtlich nur für *einen* Finger konzipiert, damit die Bedienung auch in dem oben beschriebenen Kontext noch gut möglich ist, denn je mehr Finger für eine Eingabe benutzt werden, desto höher ist die Wahrscheinlichkeit der Fehleingabe beziehungsweise Falscherkennung der Geste. Der Benutzer bestimmt bei der Festlegung der *SingleTouch*-Geste deren Pfadverlauf, so dass er den vollen Einfluss über die Einfachheit der Geste sowie über die Differenzierbarkeit des Pfadverlaufs im Gegensatz zu anderen Gesten hat. Es ist dabei logisch, die Gesten möglichst gut unterscheidbar anzulegen, damit der Detektionsalgorithmus auch bei leichter Fehleingabe die richtige Geste erkennt, denn je ähnlicher sich zwei Gesten sind, desto genauer muss die Geste nachgezeichnet werden, um die gewünschte Funktion abzurufen. Der Gestenmodus wird aktiviert, indem eine viertelkreisförmige Bildschaltfläche am unteren linken Rand des Bildschirms betätigt wird. Anschliessend zeichnet der Nutzer einen Gestenpfad, welcher einer abgespeicherten Geste zugeordnet wird und die Software führt die damit verbundene Aktion aus. Ein guter Nebeneffekt des Einsatzes der selbst definierbaren Gesten ist, dass von der Gestendefinition bis zum Abfang des damit verbundenen Schlüsselwortes im Code weitaus weniger Implementierungsaufwand nötig ist, als mit Android-Bordmitteln eine neue Schaltfläche anzulegen und deren Benutzung abzufragen. Die Gestentechnik eignet sich demzufolge ebenfalls sehr gut, um zur Fehlersuche oder zum Test von kleinen Komponenten bestimmte Codestücke per Benutzoberfläche auszuführen.

Um die Erwartungshaltung der Nutzer zu erfüllen, wurden bereits etablierte und auch in bisherigen Forschungsergebnissen ([HC11] [WMW09]) entwickelte Single- oder MultiTouch-Gesten für die Navigation innerhalb der 3D-Welt verwendet. Alle Gesten wurden dabei mit dem Ziel umgesetzt, eine möglichst robuste Eingabe mit so wenig agierenden Fingern wie möglich zu erhalten.

Für die Translation der Kartendarstellung inklusive der darauf gezeichneten 3D-Objekte wurde die sogenannte *Drag*-Geste verwendet (siehe Abb. 4.3). Diese wird erkannt, wenn ein einzelner Finger auf der Oberfläche aufsetzt, sich ein Stück bewegt und anschliessend wieder von der Oberfläche abgehoben wird. Die 3D-Objekte bewegen sich dabei derart unter dem aufsitzenden Finger mit, als ob die 3D-Szene an dem manipulierenden Finger festgeklebt wäre, wodurch ein sehr natürliches und intuitives Bearbeitungsgefühl entsteht. Dieses wird noch zusätzlich dadurch verstärkt, dass bei Abheben des Fingers die Bewegung der 3D-Objekte sanft ausläuft, so wie es beim Anschieben einer Papierseite auf einem Tisch auch passieren würde.

Für eine Vergrößerung oder Verkleinerung der Szene wurde die *Scale*-Geste eingesetzt (siehe Abb. 4.3), wie sie mittlerweile auf jedem Smartphone eingesetzt wird. Dabei werden zwei Finger auf dem Bildschirm aufgesetzt und dann zueinander oder voneinander weg bewegt, wobei die Szenenausschnitte unter den Fingern sich hier auch wieder wie festgeklebt unter den Touchpunkten mitbewegen, und dadurch der Rest der Szene entsprechend skaliert wird. Diese Geste bildet ebenfalls ein natürliches Verhalten ab, da mit diesen Fingerbewegungen auch die auf einem elastischen Materi-

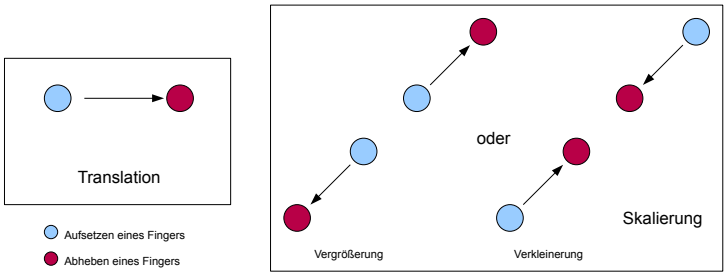


Abbildung 4.3.: Translations- und Skalierungsgeste

al aufgedruckten Elemente in dieser Weise vergrößert oder verkleinert werden könnten (beispielsweise bei einem bedruckten Luftballon). Um das Gefühl der am Finger festklebenden Szene vollständig abzubilden, wird bei Bewegung der Finger in eine gleiche Richtung die Szene zusätzlich zur Skalierung ebenfalls noch um den Mittelpunkt der fingerverbindenden Strecke transliert.

Eine ebenfalls weitverbreitete und wissenschaftlich für gut befundene Geste ist das Antippen eines Elements, um nähere Informationen zu erhalten. [HC11] [WMW09] In dem gegebenen Zusammenhang öffnet sich bei Antippen eines Objekts in der 3D-Szene ein Dialog mit allen assoziierten Informationen des Objekts (siehe dazu auch Kapitel 3.5.6). Wenn es mit dem aktuellen Datenstand der *DrawObjectDatabase* möglich ist, werden die Daten so zusammengeführt, dass eine weitere Navigation in die jeweiligen Informationsdialoge der assoziierten Elemente möglich ist. So lässt sich etwa bei Aufruf eines Trajektoriendialogs ebenfalls ein Dialog mit den Informationen der Kamera öffnen, welche die Trajektorie zuletzt verlängert hat. Von dort ist es dann möglich, auf den Videobilddialog zuzugreifen, in dem es nun wiederum möglich ist, die Kamerabewegung zu steuern (siehe Abb. 4.4). Hierfür wurde eine auf Visualisierung und Interaktion bezogene *Drill-Down*-Metapher [DRI] verwendet, welche in vielen mobilen Betriebssystemen (z.B. iOS und Android) und Frameworks verwendet wird, um dem User gleichzeitig viele Informationen bereitzustellen, dabei aber trotzdem die Übersicht auf einer kleinen Bildfläche zu behalten.

Die Metapher besteht darin, dass eine schrittweise Navigation durch hierarchische Daten erfolgt, wobei jeder Schritt tiefer in die Hierarchie eine immer detailliertere Auskunft über den betrachteten Sachverhalt ergibt. Wenn ein neuer Dialog geöffnet wird, überlagert er die jeweils unter ihm liegenden vollständig, so dass der Benutzer nicht durch zu viele Informationen verwirrt und von seinem Ziel abgelenkt wird. Bei Antippen des Bildschirms ausserhalb des Dialogs, also auf die Karte mit überlagerten 3D-Objekten, wird schrittweise wieder in der Navigationshierarchie zurück navigiert (*Drill-Up*), so dass es jederzeit möglich ist, einen falsch gewählten Navigationsschritt rückgängig zu machen und die Navigation auf dieser Hierarchiestufe fortzusetzen. Auf diese Weise muss bei einer Fehleingabe der Auswahlprozess nicht von vorne gestartet werden, was dem Nutzer eine Zeitersparnis beschert.

Während der Benutzung des Geräts fiel auf, dass durch Skalieren und Verschieben der Szene oft ein Punkt erreicht wurde, an dem die Übersicht über die Szene verloren war. Um diese Übersicht wiederzuerlangen, wurde zuerst für diesen Zweck eine neue

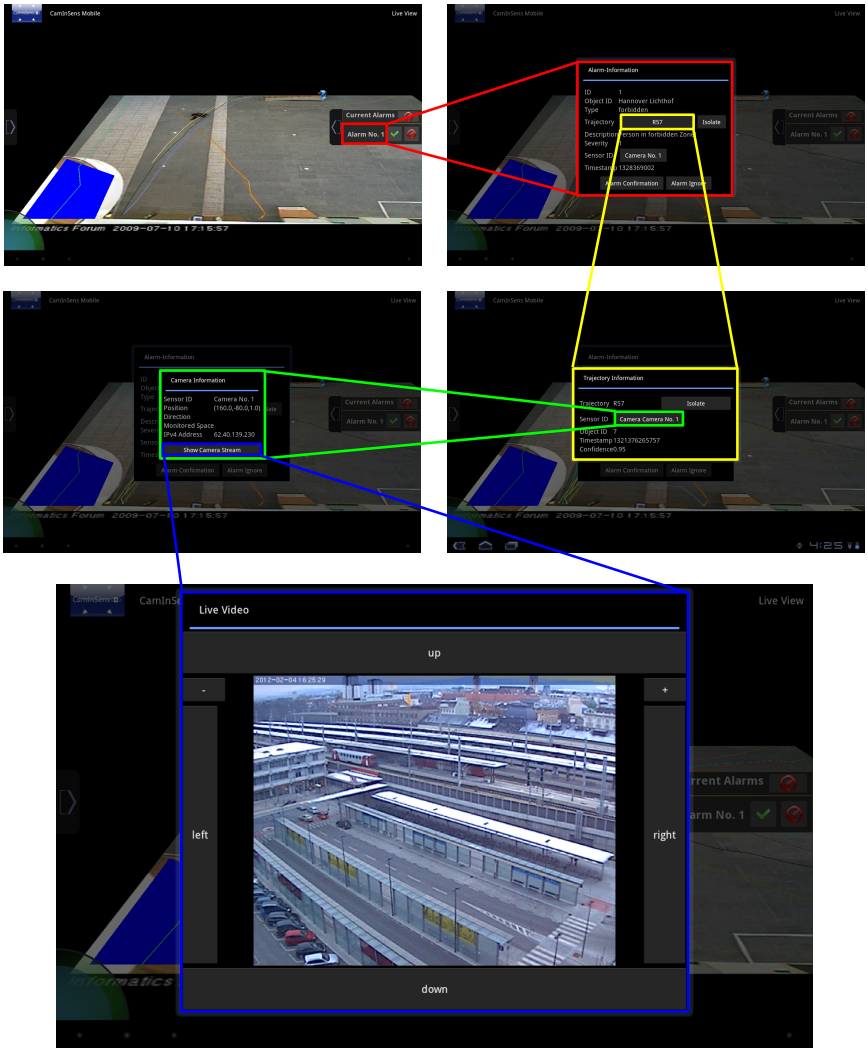


Abbildung 4.4.: Drill Down-Metapher, wie in der Software umgesetzt

Schaltfläche in dem linken, ausfahrbaren *Panel* angelegt, die die Szenentransformation wieder auf einen definierten Anfangszustand zurücksetzte. Es wurde jedoch schnell klar, dass ein immer damit verbundenes Öffnen und Schliessen des *Panel*s zur Folge hatte, dass entweder das *Panel* ständig geöffnet war, oder dass versucht wurde, mit der *Scale*- und *Drag*-Geste wieder zum Ausgangszustand zu gelangen. Um diesem Zustand zu begegnen, wird dem Nutzer, wie oben bereits erwähnt, als Alternative zu den Schaltfläche auch eine *SingleTouch*-Gestensteuerung ermöglicht, jedoch wurde diese Funktion so oft gebraucht, dass sich für eine eigene, im Programm festgeschriebene Geste entschieden wurde. Die Wahl fiel auf die *Double-Tap*-Geste (siehe Abb. 4.5), die analog zu einem Doppelklick auf einer Computermaus abläuft: Man tippt zweimal in einem kurzen Zeitabstand auf den Bildschirm. Diese Geste konnte in diesem Kontext nicht, wie bei einem Desktopcomputer, zur Aktivierung eines Objekts verwendet werden, da es auf einem Touch-Bildschirm nahezu unmöglich ist, zweimal

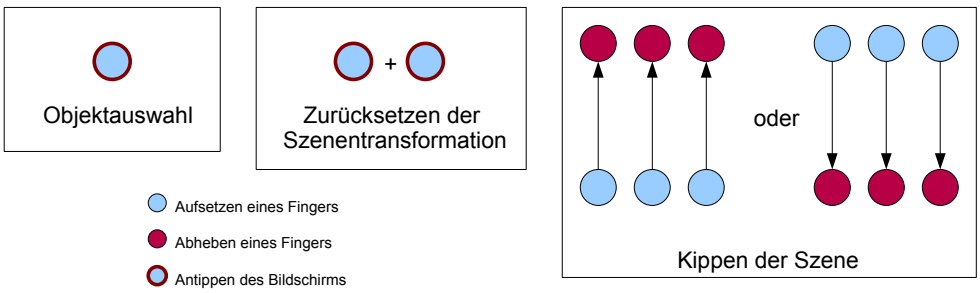


Abbildung 4.5.: Tap, DoubleTap und 3-Finger Kippgeste

auf den gleichen Punkt zu tippen. Bei naher Platzierung zweier Objekte zueinander wäre es dann nicht mehr möglich zu differenzieren, welches Objekt vom Benutzer bei seinem Auswahlversuch gemeint war. Ein weiterer Vorteil des *DoubleTap* ist, dass diese Geste sehr leicht auslösbar ist, aber dennoch eine sehr geringe Fehlauslösequote besitzt. So kann diese Geste einerseits komplett ohne Kenntnisse des Bildschirminhalts verwendet werden, da kein spezielles Objekt oder ein Bereich getroffen werden muss, andererseits ist die Chance sehr gering, dass der *DoubleTap* durch Fehleingabe einer anderen Geste ausgelöst wird. Beispielsweise wird kein Nutzer in derart kurzen Zeitabständen Objektinformationen per Antippen anfordern oder die Szene translieren, dass ein *DoubleTap* dabei getriggert werden würde. Die Verwendung des *DoubleTap* für diese Funktion bildete somit einen Kompromiss zwischen intuitiver Bedienung und technischer Umsetzbarkeit, denn intuitiv ist diese Geste für jeden Benutzer, der jemals mit einem gängigen Betriebssystem und Tastatur/Maus-Eingabegeräten gearbeitet hat, bereits belegt. Es konnte im Arbeitsrahmen jedoch keine absolut natürliche und gleichzeitig robust detektierbare Geste für eine Intention des Zurücksetzens entwickelt werden, so dass schliesslich der *DoubleTap* zum Zurücksetzen der Szenentransformation genutzt wurde, was sehr gut funktionierte und schnell verinnerlicht wurde.

Wie in Kapitel 3.5.4 beschrieben, wurde auch ein Kippen der Szene benötigt, um dem Benutzer einen besseren Blick auf die eventuell vorhandene 3D-Rekonstruktion bereitzustellen, und um ihm eine bessere Immersion in die Szene zu ermöglichen. Eine geeignete Geste für diese Funktion wurde durch Implementierung einer Drei-Finger Wischbewegung gefunden. Im Gegensatz zum Rücksetzen der Szenentransformation wurde diese Funktion weitaus weniger häufig benutzt, meist wurde der Kippwinkel nur einmal beim Start der Anwendung eingestellt, was den Einsatz einer *Multi-Touch*-Geste rechtfertigt. Hinzukommend ist diese Funktion nicht gut dafür geeignet, um mit einer Schaltfläche oder *SingleTouch*-Geste gesteuert zu werden, da der Nutzer möglichst schon bei Ausführung der Geste ein Feedback erhalten will, wie die Szene im aktuellen Kippwinkel aussieht. Diese Argumente und die Tatsache, dass sowohl Ein- als auch Zwei-Finger-Gesten bereits durch Translation und Skalierung belegt waren, wurde sich für die Verwendung einer Drei-Finger-Geste entschieden. Diese sollte dabei allerdings nicht zu kompliziert zu bedienen sein, weswegen nur die Aufwärts- und Abwärtsbewegung der Finger auf dem Bildschirm, also in y-Richtung, für das Kippen der Szene verwendet werden. Die Bedienungsschwierigkeit der Geste

verhält sich daher ähnlich zu der Translationsgeste, nur mit dem Unterschied, dass drei Finger auf dem Bildschirm aufgesetzt sein müssen.

4.3. Implementierung

Die Implementierung der in Kapitel 4.2 erwähnten Gesten gestaltete sich durch Verwendung der Android-Klassen *GestureDetector* [GES] sowie *ScaleGestureDetector* [SCA] sehr einfach. Bevor diese Klassen eingesetzt werden können, müssen alle Touch-Events abgefangen werden, die auf dem 3D-View getätigt werden. Mit *setOnTouchListener (OnTouchListener onTouchListener)* [SET] kann in jedem View definiert werden, welche Klasse die Touch-Meldungen dieses Views empfangen soll. Die Empfängerklasse, in der hier betrachteten Software die Klasse *MapTouchListener*, muss dabei das Interface *OnTouchListener* [ONT] implementieren, wodurch eine Methode *onTouch (View v, MotionEvent event)* in der Klasse enthalten sein muss. Diese Methode empfängt fortan alle Touch-Events in Form eines *MotionEvent*-Objekts [MOT], welches detaillierte Informationen über jede Berührung des Bildschirms im Anzeigebereich des Views beherbergt. Bei Eintreffen der *MotionEvent*s werden diese direkt an die vorher instanziierten Klassen *ScaleGestureDetector* sowie *GestureDetector* weitergereicht, welche die empfangenen Informationen in erkannte Gesten umsetzen. Zum Empfang der ausgewerteten Gesten muss die *MapTouchListener*-Klasse zusätzlich die Interfaces *OnGestureListener* [ONG], *OnDoubleTapListener* [ONDa] und *OnScaleGestureListener* [ONSd] implementieren, wodurch zahlreiche Methoden zur Annahme von Gesteninformationen entstehen.

Im Folgenden wird nun erklärt, mit Hilfe welcher Callback-Methoden die gewünschten Gesten implementiert werden konnten.

4.3.1. Drag-Geste

Die Drag-Geste besteht im wesentlichen aus der Bewegung eines Fingers über den Bildschirm, was in der Callback-Methode *onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)* [ONSb] abgebildet wird. *e1* beinhaltet die Informationen über das Touch-Event beim Aufsetzen des Fingers, *e2* ist das aktuell empfangene Event. Die Werte in *distanceX* bzw. *distanceY* beschreiben die zurückgelegten Distanzen des Fingers seit dem letzten Touch-Event in x- beziehungsweise y-Richtung. Diese beiden Distanzwerte werden an die Methode *translate(screenX, screenY)* in die Klasse *MapRenderer* weitergereicht, welche den *OpenGL-View* bereitstellt. Wie schon anhand der Methodensignatur zu erkennen, befinden sich die Translierungswerte noch in Bildschirmkoordinaten und werden daher in der *translate*-Methode in Weltkoordinaten der Szene umgerechnet.

Dazu wird, wie in Abb. 4.6 dargestellt, zuerst die Seitenlänge der momentan betrachteten Szene berechnet. l ist in diesem Fall der Abstand der Kamera zur Szenen-Ebene, der *fov*-Winkel (Field-of-View) kann aus den aktuellen Kameraparametern ausgelesen werden, und c ist die Anzahl der Pixel des Geräts in x-Richtung. Durch Dividieren der erhaltenen Szenen-Seitenlänge durch die Zeilenpixelanzahl des Geräts wird ein Transformationsfaktor erlangt, mit dessen Hilfe durch Multiplizieren mit einer Bildschirmkoordinate schliesslich eine Szenenkoordinate produziert wird. Eine Alter-

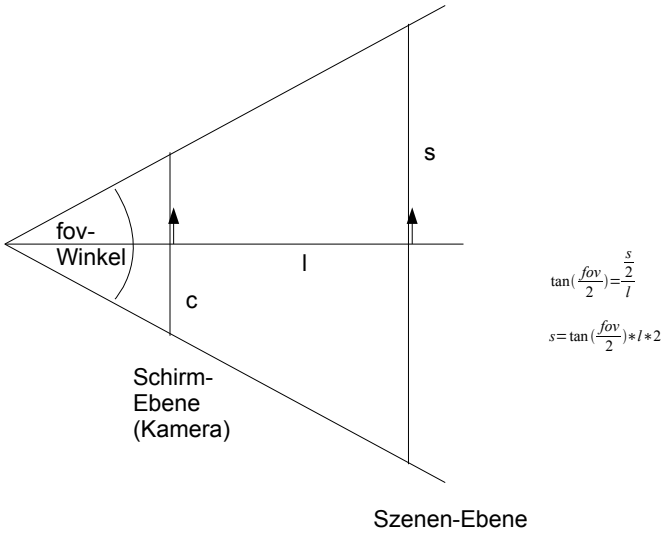


Abbildung 4.6.: Umrechnungsschritt von Bildschirm- in lokale Koordinaten

native zu dieser Umrechnung ist, einen Strahl entlang der Kameraachse zu schicken, welcher durch den Aufsetzpunkt des Fingers läuft. Dort, wo dieser Strahl die Hintergrundebene der Szene schneidet, liegt dann schliesslich die Szenekoordinate. Aufgrund dessen, dass die Koordinatenumwandlung aber sehr häufig benutzt wird, wurde die Umrechnung anhand einer Formel bevorzugt, da diese erhebliche Performanzvorteile gegenüber der vorstehenden *Picking*-Methode mit sich bringt.

Natürliche Translation

Da zusätzlich nach Abheben des Fingers eine natürliche Weiterbewegung der Translation implementiert werden sollte, wurde dies mit Hilfe einer gedämpften Federfunktion gelöst. Beim Abheben des Fingers wird vom *GestureDetector* die Funktion *onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)* angesteuert, die in *velocityX* und *velocityY* die Geschwindigkeiten des Fingers in x- und y-Richtung zum Zeitpunkt des Abhebens bereitstellt. Mit Hilfe einer *Runnable* [RUN] und eines *Handler*s [HAN] werden nun kontinuierlich sechzig Mal pro Sekunde die ausklingenden Translationswerte berechnet und an die *translate*-Methode in *MapRenderer* weitergereicht, bis ein Translations-Schwellwert unterschritten wird.

Eine Java-*Runnable* beinhaltet ein Codestück, das mit dessen *run()*-Methode jederzeit abgespielt werden kann und es wird oft verwendet, um beispielsweise einen Thread zu starten, ohne dafür eine von *Thread* erbende Klasse implementieren zu müssen. Stattdessen kann auch im Konstruktor *Thread(Runnable target)* einfach das Codefragment überreicht werden, was dann nach Aufruf von *start()* in einem separaten Thread ausgeführt wird. Ähnlich verhält sich die Android-eigene Klasse *Handler*, welcher ebenfalls eine *Runnable* überreicht werden kann. Der *Handler*-Mechanismus funktioniert dabei so, dass ein mit *post(Runnable r)* an eine *Handler*-Instanz übereichtes Codestück danach im gleichen Thread ausgeführt wird, in dem der *Handler*

instanziiert wurde. So ist es in *Android* sehr einfach, eine Kommunikation zwischen Threads, wie etwa zwischen einem Berechnungs- und dem OpenGL-Thread, zu erreichen. Da die verwendete Grafikbibliothek *jpct-AE* nicht threadsicher ist, und somit alle die Grafik betreffenden Sachen auch im OpenGL-Thread durchgeführt werden müssen, wurde die *Handler*-Funktionalität im Rahmen dieser Arbeit sehr häufig eingesetzt.

4.3.2. Scale-Geste

Die Skalierungsgeste konnte vollständig mit Hilfe der Methoden *onScaleBegin (ScaleGestureDetector detector)*, *onScale (ScaleGestureDetector detector)* und *onScaleEnd (ScaleGestureDetector detector)* (siehe [ONSd]) umgesetzt werden. Die den Methoden als Parameter übergebene *ScaleGestureDetector*-Instanz liefert alle benötigten Informationen wie etwa den Skalierungsfaktor oder den Mittelpunkt zwischen den beiden aufgesetzten Fingern. Dieser wird dafür benötigt, die Skalierung nicht etwa um den Ursprung der Szene, sondern um den aktuell durch die Geste fokussierten Punkt erfolgen zu lassen, so wie es der Benutzer auch erwartet. Umgesetzt wurde diese Anforderung durch ein Verschieben des aktuellen Gestenmittelpunkts in den Ursprung der Welt, darauffolgendes Skalieren und schliesslich die Rücktranslierung des Mittelpunkts an die ursprüngliche Position. Die Umrechnung der Punkte von Bildschirm- in Szenekoordinaten wurde auch hier durch die in Kapitel 4.3.1 vorgestellte Umrechnung gelöst.

4.3.3. Antipp-Gesten

Durch Verwendung des *OnDoubleTapListener*s und dessen Callback-Funktion *onSingleTapConfirmed(MotionEvent e)* [ONSa] konnte die Funktion zur Auswahl von Objektinformationen, welche durch das Antippen eines Objekts erfolgen sollte, komfortabel ausgelöst werden. Das mitgelieferte *MotionEvent* beinhaltet die Bildschirmkoordinaten, die für das darauffolgende *Picking* des 3D-Objekts nötig sind. In diesem Kontext war es wichtig, nicht die *OnGestureListener*-Funktion *onSingleTapUp(MotionEvent e)* [ONSc] zu verwenden, da nur *onSingleTapConfirmed* garantiert, dass anschliessend an das erste Abheben des Fingers vom Schirm kein *DoubleTap* mehr erfolgt. Diese Unterscheidung ist wichtig, denn ein *DoubleTap* wurde verwendet, um die Szenentransformation zurückzusetzen, was in der Methode *onDoubleTap(MotionEvent e)* [ONDb] angestossen werden konnte. Eine Fehlauslösung der Antipp-Geste wäre somit bei Verwendung der ungeeigneten Funktion *onSingleTapUp(MotionEvent e)* sehr wahrscheinlich gewesen.

Wenn am angetippten Punkt der Szene mehrere 3D-Objekte vorhanden sind, wird das *Picking* weiter ausgeführt, bis schliesslich die Hintergrundkarte erreicht wird, was die Erreichbarkeit von vollständig oder teilweise okkludierten Objekten garantiert. Die auf diesem Weg gesammelten Objekte werden dem Benutzer nacheinander als Dialog angezeigt, so dass er sich für die gewünschte Information entscheiden und seine Datensuche weiter verfolgen kann.

4.3.4. Drei-Finger Geste

Im Gegensatz zu den anderen Gesten benötigte die Drei-Finger-Geste zum Ankippen der Szene mehr Implementierungsaufwand. Da kein in *Android* mitgelieferter Gestenerkennungsmechanismus für mehr als zwei Finger integriert ist, mussten alle einzelnen Touch-Events abgefangen und hinsichtlich ihrer Relevanz bezüglich der Drei-Finger-Geste überprüft werden, noch bevor die *MotionEvent*s an die beiden *GestureDetector*-Instanzen weitergegeben werden.

Ein *MotionEvent* beinhaltet alle Informationen eines aktuellen Touch-Events, und folglich können daraus auch die Informationen gewonnen werden, wieviele Finger gerade auf dem Bildschirm aufliegen, ob die Finger gerade aufgesetzt oder abgehoben werden, und ob die Finger gerade eine Geste ausführen. Diese Auskunft erhält man, indem die Methode *getAction()* ausgeführt wird, welche eine durch Bitmasken codierte Zahl zurückliefert, die die ausgeführte Aktion sowie den Index des aktuell aufgesetzten oder abgehobenen Fingers zurückliefert. Um die aktuelle Aktion (POINTER_DOWN, POINTER_UP oder MOVE) herauszufinden, muss diese Zahl dementsprechend mit der *ACTION_MASK* demaskiert werden, darüberhinaus kann mit *getPointerCount()* die Anzahl der aufliegenden Finger abgefragt werden.

Wenn nun der Start der Geste durch *POINTER_DOWN* und durch drei aufliegende Finger erkannt wird, wird das Flag *tiltActionRunning* auf *true* gesetzt, was anschliessend verhindert, dass die *GestureDetector*-Instanzen weitere Touch-Events erhalten. Dieses Flag ist nötig, denn der *ScaleGestureDetector* erkennt auch bei drei aktiven Fingern eine Scale-Geste, was zu unerwünschten Nebeneffekten bei der Bedienung führen kann. Schliesslich wird bei Empfang jedes erhaltenen Touch-Events nun berechnet, wie weit sich die Finger entlang der y-Achse bewegt haben und ein daraus berechneter Winkelwert an die *tilt(float angleDeg)*-Funktion des *MapRenderer* übergeben. Diese Methode addiert den erhaltenen Winkelwert zur aktuellen Drehung der Szene hinzu, wobei jedoch eine Ober- und Untergrenze der Drehung beachtet wird, damit die Szene nicht für den Benutzer unsichtbar wird.

5. Bewertung

Nachdem in den vergangenen Kapiteln der Projektrahmen, die Anforderungen, die Auswahl einer geeigneten Hardware- und Softwareplattform und schliesslich die Konzeption und Umsetzung der Visualisierungs- und Interaktionskomponenten beleuchtet wurden, soll nun dieses Kapitel aufgrund eines Benutzertests die Güte des entwickelten Produkts bewerten. Anbetracht des Einsatzzweckes sind dabei die wichtigsten zu stellenden Fragen:

- Kann in einem Alarmfall der Täter mit Hilfe der vorgestellten Lösung erfasst werden?
- Ist die Visualisierung zielführend und möglichst effektiv für die Erfassung der Gefahrensituation?
- Ist die Interaktion intuitiv und effizient, so dass Informationen möglichst schnell abgerufen werden können?
- Ist es dem mobilen Personal auch möglich, autonom zu arbeiten, wenn beispielsweise die Sicherheitszentrale ausfällt?

Zur Untersuchung dieser Fragen wurde eine Benutzerstudie durchgeführt, die in diesem Kapitel präsentiert wird. Als erstes wird das Konzept und die Methode der Studie vorgestellt, welche eine Trainingsphase, die eigentlichen Aufgabenstellungen und eine Testdurchführung inklusive Ausfüllen eines Fragebogens umfasst. Schließlich werden die Ergebnisse der Studie ausgewertet, Vorschläge der Benutzer aufgeführt und eigene Beobachtungen eingebracht.

5.1. Konzeption der Benutzerstudie

Die Applikation soll später eingesetzt werden, um mobile Sicherheitskräfte bei der Überwachung eines potenziell gefährlichen Bereichs zu unterstützen. Die Software wird immer nur in diesem speziellen Kontext eines professionellen Sicherheitsdienstes eingesetzt werden, so dass davon ausgegangen werden kann, dass die Benutzung der Anwendung immer mit einer vorausgehenden Schulung stattfinden wird - im Gegensatz zu einer Applikation, die frei auf einer Webseite verfügbar ist und dann auf eigene Faust erkundet werden muss. Daher findet eine kurze Einführung statt, nach welcher dem Probanden Aufgaben gestellt werden, die die obenstehenden Fragen möglichst gut überdecken. Jeweils nach Durchführung einer Aufgabe und am Ende des gesamten Tests wird der Benutzer gebeten, einen Fragebogen zu beantworten.

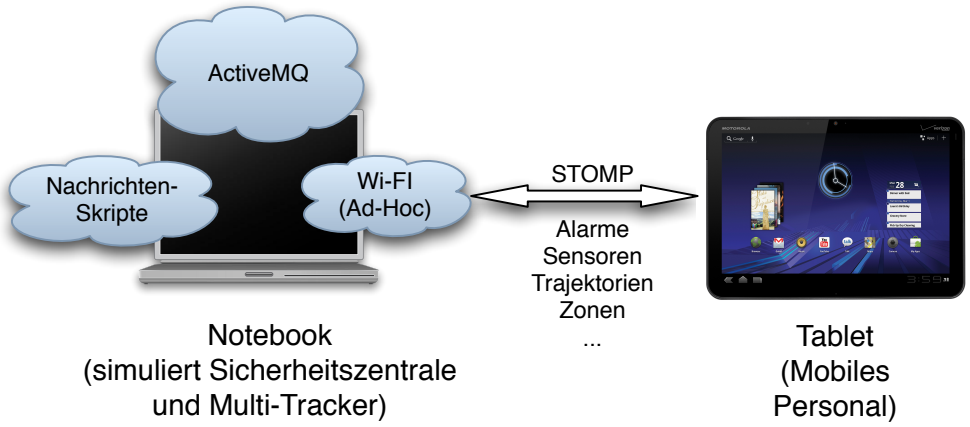


Abbildung 5.1.: Testumgebung

5.1.1. Testumgebung

Ein lokaler *ActiveMQ*-Server wurde mit den Projekt-Konfigurationsdaten auf einem tragbaren Computer aufgesetzt. Aufgrund dessen, dass der verwendete Tablet-Computer lediglich über eine Funk-Netzwerkverbindung verfügt, wurde auf dem Laptop eine Ad-Hoc Funkverbindung angelegt, auf welche sich das Tablet verbinden und somit die Verbindung zum Nachrichtenserver herstellen kann (siehe Abb. 5.1). Auf diese Weise war es schliesslich möglich, den Testort völlig frei zu wählen.

Natürlich war es nicht machbar, ein Testsetup mit allen datenliefernden Instanzen des Projekts zu erstellen, so dass realistische Abläufe mit Hilfe von Skripten nachgeahmt wurden. Diese beherrschen alle im Projekt definierte Nachrichtenformate und können so durch Versenden der entsprechen Nachrichten an den Server vollständig die Instanzen eines *Multi-Tracker*, einer Sicherheitszentrale oder eines Kleinstsensoren nachbilden. Mit diesen Mitteln wurde eine Umgebung geschaffen, die einen Einsatz des mobilen Geräts in einem Überwachungskontext möglichst gut nachahmt.

5.1.2. Trainingsphase

Der Ablauf einer Trainingsphase wurde schriftlich fixiert und immer gleich ausgeführt, um allen Benutzern die gleichen Voraussetzungen für die Lösung der Aufgaben zu geben. Es wird zuerst die Applikation ohne Inhalte gestartet und im Hintergrund per Netzwerk die Edinburgh-Daten (siehe Kapitel 3.2) eines Tages mit doppelter Geschwindigkeit abgespielt. Der Einsatzbereich der Anwendung, also die Überwachung der Personenbewegungen in einem bestimmten Bereich, wird dem Benutzer erklärt. Anschliessend wird kurz erläutert, was eine Trajektorie begrifflich und visuell in der Anwendung darstellt, und zusätzlich wird einerseits erklärt, dass ein Trajektorienende durch ein großes Viereck dargestellt wird, und andererseits dass die Farbe der Trajektorie keine Rolle hinsichtlich ihres Gefahrenpotentials oder anderer Eigenschaften spielt. Ebenso wird darauf eingegangen, dass im Hintergrund eine Kartendarstellung des überwachten Areals angezeigt wird, und dass die Trajektorien maßstabsgetreu zur Karte eingezeichnet sind.

Als Nächstes erhält der Proband eine Beschreibung der Interaktionsmöglichkeiten, es werden die *Drag-*, *Scale-* und *3-Finger*-Geste aufgezeigt, gefolgt von *DoubleTap* und dem Antippen von 3D-Objekten. Ausgehend davon werden die Informationsdialoge der verschiedenen 3D-Objekte erkundet und es wird erklärt, wie diese miteinander vernetzt sind. Danach werden die ausfahrbaren *Panel*s auf der linken und rechten Bildschirmseite betrachtet und deren Inhalte veranschaulicht. Die verschiedenen Modi, die im linken *Panel* getriggert werden können, werden durchlaufen und es wird dabei erklärt, dass im Gruppierungsmodus eine dickere Trajektorienbreite eine Aussage über die Größe der Gruppe trifft. Durch Abspielen eines Alarms per Netzwerk wird das rechte *Panel* geöffnet und die mit dem Alarm zusammenhängenden Möglichkeiten, wie beispielsweise Trajektorienisolation oder Kamerastram, eröffnet. Abschliessend erfolgt eine Einführung in die *SingleTouch*-Gesten und es wird ein Ausdruck (siehe Abb. 5.2) bereitgestellt, auf dem die aktuell abrufbaren Gesten zu finden sind. Es wird dabei klargemacht, dass entweder die Schaltflächen im ausfahrbaren *Panel* oder die *SingleTouch*-Gesten verwenden werden können, wobei beide Methoden in der Software das gleiche Kommando ausführen.

Anschliessend wird dem Benutzer etwas Zeit gegeben, um sich ohne Hilfe mit dem Gerät und der Software vertraut zu machen.

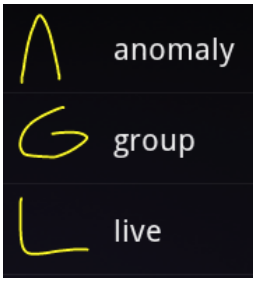


Abbildung 5.2.: Bei Benutzertest vorhandene, dem Benutzer ausgedruckt vorliegende SingleTouch-Gesten

5.1.3. Zu lösende Aufgaben

Die Aufgaben wurden so konzipiert, dass sie gleichzeitig ein reales Szenario nachstellen und die in Kapitel 5 entwickelten Fragen abdecken. Als erste Aufgabe wurde ein Szenario ausgearbeitet, welches einen Alarmfall simuliert und sehr nah an das *Forbidden Zone* Szenario (siehe Kapitel 2.2.1) angelehnt ist. Es werden wieder Trajektoriendaten mit doppelter Geschwindigkeit abgespielt und zusätzlich eine *ForbiddenZone* sowie ein Kamerasensor angelegt. Im folgenden Verlauf werden durch abgespielte Skripte Trajektoriendaten so generiert und im Netzwerk verbreitet, dass auf dem mobilen Gerät eine Trajektorie in die *ForbiddenZone* läuft. Anschliessend wird ein Alarm abgespielt, welcher von der Software empfangen und ausgewertet wird. Der Benutzer erhält nun die Aufgabe, den Täter möglichst schnell ausfindig zu machen und anschliessend die Bearbeitung des Alarms zu bestätigen. Die genaue Aufgabenstellung lautet dabei folgendermassen:

1. Es wird nun ein Szenario abgespielt.
Untersuchen Sie jeweils die Trajektorie, die den Alarm auslöst.
a) Versuchen Sie, die Trajektorie isoliert von den anderen zu betrachten.
b) Betrachten Sie den dazugehörigen Videostream.
c) Bestätigen Sie die Bearbeitung des Alarms.
d) Kehren Sie danach zurück in die Live-Ansicht.

Abbildung 5.3.: Erste Aufgabe des Benutzertests

Dieser Weg durch die Applikation simuliert einen mögliche Abfolge von Schritten, um möglichst viele Informationen über den Sachverhalt zu erlangen. Es wird davon ausgegangen, dass nach Sichtung aller dieser Informationen der Täter ausfindig gemacht wurde und somit der Alarm bestätigt werden kann.

Die zweite Aufgabe soll ermitteln, wie gut eine Gefahrenerkennung auch ohne Sicherheitszentrale, das heisst ohne empfangenen Alarm, möglich ist. Dazu werden die Daten eines gesamten Tages auf einmal in die Kartendarstellung eingespielt, worauf der Benutzer das Ziel erhält, jeweils eine Trajektorie aus einer dominanten Bewegungsrichtung und eine sich anormal verhaltende Trajektorie zu betrachten (siehe Abb. 5.4). Dabei sollen, wie in der ersten Aufgabe, alle Möglichkeiten der Informationssichtung genutzt werden.

4. Sie sehen nun die Trajektoriendaten eines gesamten Tages.
Versuchen Sie, Gefahren mit Hilfe des Programmes zu erkennen und untersuchen Sie die entsprechenden Trajektorien.
Versuchen Sie dabei insbesondere, dominante Bewegungsrichtungen und sich anormal verhaltende Trajektorien zu untersuchen.
a) Versuchen Sie, jeweils eine anormale und eine Trajektorie aus einer dominanten Bewegungsrichtung isoliert von den anderen zu betrachten.
b) Betrachten Sie den dazugehörigen Videostream.
c) Kehren Sie danach zurück in die Live-Ansicht.

Abbildung 5.4.: Zweite Aufgabe des Benutzertests

5.1.4. Idealer Weg zum Ziel

Um zu verstehen, wie sich das vom Programmierer erwartete Verhaltensmuster von dem des Probanden unterscheidet, wurden Wege zum Ziel definiert, die aus Sicht des Programmierers das Optimum darstellen.

Der optimale Weg, um die erste geforderte Aufgabe zu lösen ist (siehe auch Abb. 5.5):

- Berühren der *Alarmzelle*
- Berühren des *Isolate*-Buttons
- Berühren der *Alarmzelle*
- Berühren des *Camera*-Buttons
- Berühren des *Stream*-Buttons
- Zurücknavigieren zur *Alarminformation*
- Betätigen der *Alarm Confirmation* Schaltfläche

Aufgabe 2 ist optimal durch folgende Vorgänge zu lösen (siehe Abb. 5.6):

- Geste für Gruppierung ausführen oder Schaltfläche in linkem *Panel* betätigen
- Berühren einer gruppierten Trajektorie
- Anzeige aller in der Gruppe enthaltenen Trajektorien
- Auswahl einer Trajektorie aus Liste
- Berühren der *Isolate*-Schaltfläche
- Berühren der isolierten Trajektorie
- Auswahl der Kamerainformationen
- Darstellen des Kamerastreams
- Zurücknavigieren zur Kartendarstellung
- Geste oder Schaltfläche für *Live View* betätigen

In der Auswertung (Kapitel 5.2) kann mit Hilfe dieser Festlegungen ausgewertet werden, inwiefern Benutzer intuitiv einen anderen Weg als der Entwickler einschlagen, wodurch mögliche Optimierungsbereiche sichtbar werden.

5.1.5. Antwortmöglichkeiten des Fragebogens

Wie zu Anfang dieses Kapitels erwähnt, soll die Software in erster Linie dazu dienen, mobilem Sicherheitspersonal bei der Lösung von Gefahrensituationen zu helfen. Diese Gefahrensituationen werden durch die gestellten Aufgaben nachempfunden, so dass bei der erfolgreichen Lösung der Aufgabe auch von einer Beseitigung der Gefahrenquelle in einem realen Szenario ausgegangen werden kann. Als erste Frage nach Bearbeitung einer Aufgabe muss der Proband daher beantworten, ob er die Aufgabe gelöst hat. Nach Bearbeitung der ersten Aufgabe muss er zusätzlich in einer fünfstufigen Wertung beantworten, wie gut er jeweils haptisch, visuell und audiell über den

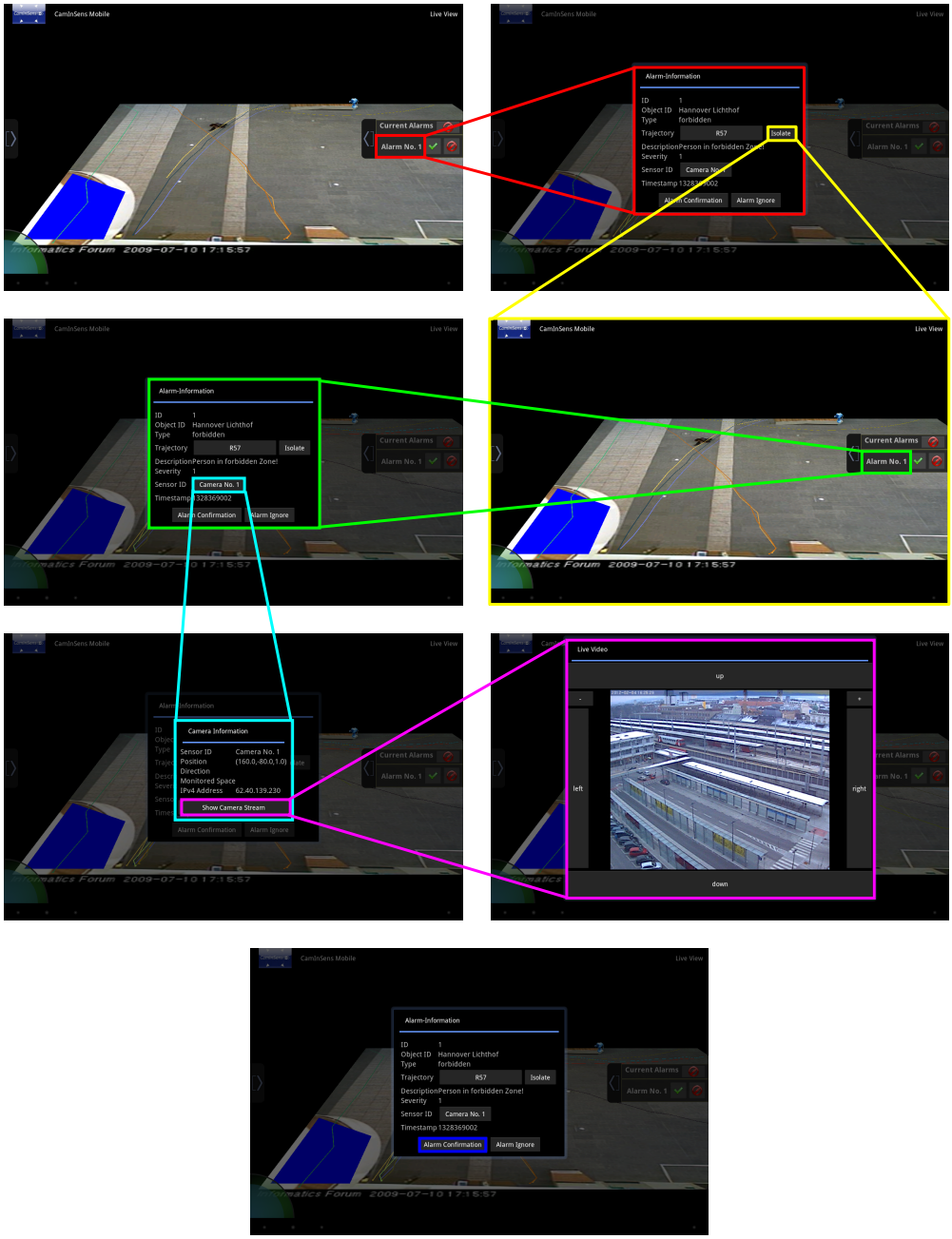


Abbildung 5.5.: Optimaler Lösungsweg Aufgabe 1

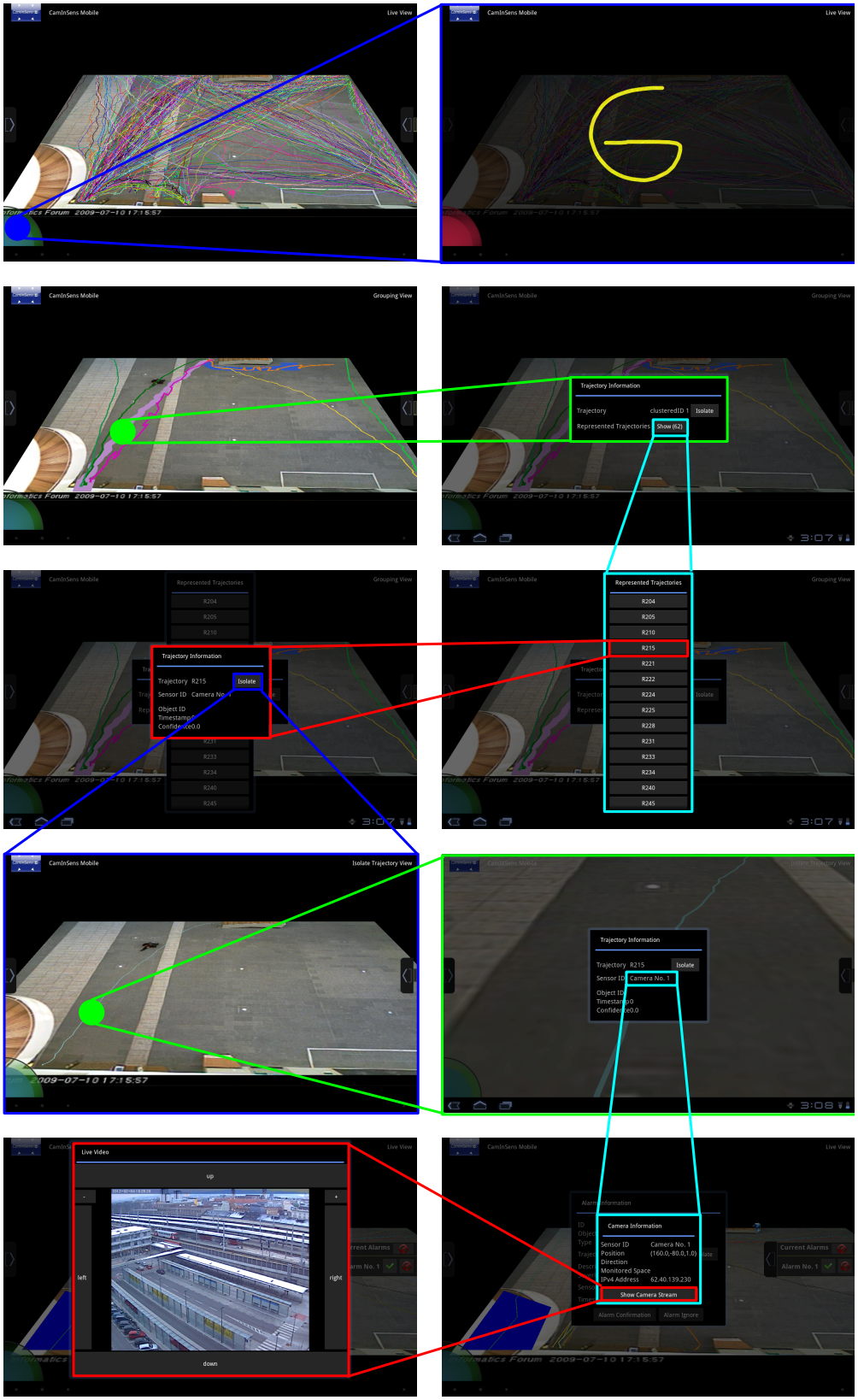


Abbildung 5.6.: Optimaler Lösungsweg Aufgabe 2: Dominante Bewegungsrichtung

Alarm informiert wurde, denn ohne eine gute Perzeption des Alarms könnte eine Gefahrensituation einfach verpasst werden.

Im Anschluss daran folgen ebenfalls fünfstufige Fragen zur Visualisierung und Interaktion, die für beide Aufgaben gelten. Die Abstufungen lauten dabei jeweils:

- Trifft absolut zu
- Trifft zu
- Weder noch
- Trifft nicht zu
- Trifft absolut nicht zu

Es wird einerseits abgefragt, ob die Interaktion vom Benutzer als effektive und effiziente Hilfe bei der Lösung der Aufgabe empfunden wurde, andererseits soll der Proband beurteilen, ob die Interaktionsmöglichkeiten für ihn intuitiv waren.

Schliesslich folgen noch einige allgemeine Fragen, um über die Kenntnisse der Probanden in Bezug auf MultiTouch- und kartenbasierter Interaktion, die verwendete Interaktionsart (Schaltflächen oder Gesten) und über allgemeine Verbesserungsvorschläge Auskunft zu erhalten. Interessant ist dabei insbesondere, ob ein möglicher Zusammenhang zwischen den Erfahrungen mit bestimmten Technologien und den eingebrachten Bewertungen oder verwendeten Interaktionsarten besteht.

Zu allen fünfstufigen Fragen existieren unmittelbar unter der Frage Freitextfelder, in denen die Tester direkt Verbesserungsvorschläge abgeben können. Auf diese Weise ist sichergestellt, dass Ideen, die am Anfang des Tests auftauchen, nicht bis zum Ende des Tests wieder vergessen werden. Die Erfassung der Testergebnisse wurde über eine Webseite [KWI] gelöst, was den Vorteil der automatischen statistischen Auswertung ohne Mehraufwand bedeutete. Zusätzlich dazu war es auf diese Weise nicht nötig, immer ausgedruckte Exemplare der Fragebögen bei sich zu tragen, so dass auch sehr spontan Testkandidaten akquiriert werden konnten.

Der Fragebogen kann in kompletter Länge in Anhang A.1 eingesehen werden.

5.2. Auswertung der Studie

Tabelle 5.1.: Altersverteilung der Probanden

Proband Nr.	1	2	3	4	5	6	7	8	9	10	11	Ø
Alter	31	32	23	28	27	27	28	29	28	24	23	27,3

Nach Durchführung des Tests mit 11 männlichen Personen, welche einen Altersdurchschnitt von 27,3 Jahren (siehe Tabelle 5.1) besaßen, konnten die ersten Trends ausfindig gemacht werden. Erfreulich ist, dass beide Aufgaben von allen Probanden vollständig gelöst wurden, womit eindrucksvoll die Erfüllung der im Projekt nötigen Ziele aufgezeigt wird.

5.2.1. Alarmwahrnehmung

Die dreigeteilte Frage zur Perzeption des Alarms zeigte viele Verbesserungsmöglichkeiten auf - so wünschten sich etwas mehr als ein Drittel der Befragten eine bessere haptische Information. Erreicht werden kann dies aber nur durch Verwendung eines anderen Tablets oder externen Geräts, welches ein stärkeres haptisches Feedback liefert, denn das *Android*-Framework erlaubt keine Steuerung der Vibrationsstärke.

Während es zum audiellen Feedback keine Beanstandungen gibt, bestehen bei der Visualisierung einer Alarmnachricht die meisten Optimierungsvorschläge. So wünschen sich 4 der 11 Testpersonen eine visuell auffälligere Benachrichtigung beim Eintreffen eines Alarms, und ebenso schlagen knapp 30% eine direkte Visualisierung der salienten und mit dem Alarm verbundenen Trajektorie nach Anwahl des Alarms vor, ohne dabei den Zwischenschritt über den Alarminformations-Dialog gehen zu müssen.

5.2.2. Visualisierung

Die Bewertung der Visualisierung im Gesamten fällt allerdings sehr positiv aus; es wurde keine schlechtere Bewertung als die zweitbeste Möglichkeit (*Trifft zu*) abgegeben. Verbesserungen wurden sehr oft zum Handling der Dialoge vorgeschlagen; hier wird es als irritierend empfunden, dass durch das streng umgesetzte *DrillDown*-Konzept nach Navigation in die Tiefe erst alle hierarchisch darüberliegenden Dialoge geschlossen werden müssen, um an die Kartenansicht oder einen Dialog in einer hohen Hierarchiestufe zu gelangen. Ebenso wird gewünscht, dass nach Ausführung eines Befehls zur Isolierung der Trajektorie automatisch alle Dialoge ausgeblendet werden und eine sofortige Sicht auf die ausgewählte Personenbewegung erhalten wird. Bei Anwahl einer gruppierten Trajektorie wünschen sich zwei getestete Personen die Möglichkeit, alle in dieser Gruppe enthaltenen Trajektorien isoliert betrachten zu können, um einen besseren Eindruck der Gruppe zu erhalten. Genauso häufig wird es als Vorteil empfunden, eine Alarmhistorie auch nach dem Bestätigen oder Ignorieren des Alarms betrachten zu können. Einer der Benutzer erwartet eine deutlichere Visualisierung des derzeit aktiven Darstellungsmodus.

5.2.3. Interaktion

Noch besser als die Visualisierung schnitt die Interaktion in der Bewertung ab; nur zwei Tester gaben nicht die bestmögliche Wertung, wobei einer der Tester in einer anderen Frage des Tests angibt, dass er sehr wenig Erfahrungen mit Multitouch-Anwendungen hat. Ohne Kenntnis von gebräuchlichen Multitouch-Bedienungsmustern ist es daher logisch, dass für diesen Probanden die Bedienung nicht sofort intuitiv war. Dieselbe Testperson gibt zusätzlich an, dass sie gerne zur Auswahl von Trajektorien ein Rechteck ähnlich der Auswahlmetapher in Desktop-Betriebssystemen ziehen möchte, was außerdem bestätigt, dass dieser Tester versucht, seine Desktop-Erfahrungen direkt auf ein mobiles Betriebssystem abzubilden. Andere Vorschläge zielen in die Richtung, das *Picking* der Trajektorien zu verbessern, indem die Trajektorie nicht exakt mit dem Finger getroffen werden muss. Derzeit ist die Auswahl eines 3D-Objekts meist nur per vorheriger Skalierung möglich, da der Finger das

(meist kleine) Objekt genau treffen muss. Bei mehreren Treffern soll ein Auswahlbildschirm erscheinen, in dem die Trajektorie anhand ihrer Farbe ausgewählt werden kann. Eine kleine Schwäche sah ein Proband darin, dass die mit einem *DoubleTap* angezoomte Stelle nach der Skalierung nicht im Mittelpunkt des Bildschirms liegt. Darüberhinaus wurde von einer anderen Testperson eine Möglichkeit gesucht, um mit einer festen Geste zum letzten eingestellten Modus zurückzukehren.

5.2.4. Allgemeine Fragen

In der Auswertung der allgemeinen Fragen zur Software und zum Kenntnisstand der Benutzer kann belegt werden, dass die Benutzer bei einer Wahl zwischen Gesten- oder Schaltflächensteuerung von Funktionen eher eine Gestensteuerung bevorzugen. Dies ist darin begründet, dass die Zeit, um eine Schaltfläche in dem ausfahrbaren Menü zu betätigen, meist länger ist, als eine Geste auszuführen. Sogar diejenigen Tester, die während des Tests vorwiegend die Schaltflächen benutzten, gaben bei Bearbeitung dieser Frage mündlich zu, dass sie bei besserer Kenntnis des Funktionsumfangs der Applikation auch eher die Gestensteuerung verwendet hätten. Diesen Fakt nutzte ein Kandidat für den Vorschlag, dass während der Gesteneingabe die Information eingeblendet werden könnte, welche Gesten insgesamt zur Verfügung stehen.

Während alle Testpersonen berichteten, dass sie regelmässig kartenbasierte Anwendungen benutzen, offenbarte ein Viertel der Benutzer, dass sie bisher keine großen Erfahrungen mit MultiTouch-Interaktionsformen gesammelt haben. Konsistent lässt sich in der Umfrage auswerten, dass genaue diese Benutzergruppe diejenige ist, die hauptsächlich versuchten, Desktop-Verhaltensweisen auf das Tablet anzuwenden. So kommt etwa der Vorschlag aus dieser Gruppe, alle möglichen Gesten dauerhaft einblendbar zu machen, was auf eine mangelnde Erfahrung beim Erlernen von Gesten zurückzuführen ist, und ebenso stammt der vorstehend erläuterte Vorschlag zum Trajektorienpicking per Auswahlrechteck aus dieser Personenmenge.

Die vollständigen Ergebnisse des Benutzertests inklusive aller Benutzerkommentare können in Anhang A.2 betrachtet werden.

5.2.5. Beobachtungen

Solange die Benutzer den Test ausführten, wurden sie aus unbeeinflussender Distanz beim Lösen der Aufgabe beobachtet und die Aufgabenlösungen mit der optimalen Lösung der jeweiligen Aufgabe (siehe Abb. 5.5 und 5.6) verglichen. Im Laufe dessen fiel hauptsächlich auf, dass viel mehr *Picking* als angenommen benutzt wurde; beispielsweise wurden Videobilder durch Anwählen der 3D-Repräsentation des Kamerasensors angezeigt oder im dritten Schritt der optimalen Lösung der ersten Aufgabe (Abb. 5.5, gelb rumrandet) nicht die Alarmzelle angetippt, um wieder Informationen der isolierten Trajektorie zu öffnen, sondern stattdessen die Trajektorie in der 3D-Szene gepickt. Darüberhinaus konnten jedoch keine großen Differenzen zwischen den optimalen und den durchgeführten Lösungswegen erkannt werden, alle Probanden konnten die Aufgaben ohne größere Überlegungen flüssig lösen.

6. Fazit und Ausblick

In dieser Arbeit wurde die Realisierung einer mobilen Sicherheitslösung für Überwachungszwecke vorgestellt, welche unter Zuhilfenahme des staatlich geförderten Forschungsprojekts *CamInSens* entwickelt wurde. *CamInSens* soll erreichen, in überwachten Gefahrenbereichen die erhaltenen Video- und Sensorendaten so zu analysieren, dass Bedrohungen möglichst frühzeitig erkannt und behandelt werden können. Das Ziel dieser Arbeit war, auf Basis von verarbeiteten Daten aus *CamInSens* eine Interaktion und Visualisierung zu entwickeln, die bei einem späteren Praxiseinsatz mobilem Sicherheitspersonal dabei hilft, in Bedrohungssituationen fundierte Entscheidungen treffen zu können. Zu diesem Zweck wurde nicht nur eine Software implementiert, sondern auch eine Marktsichtung hinsichtlich geeigneter Geräte und einsetzbarer Softwarebibliotheken durchgeführt. Die Software bietet den Benutzern die Möglichkeit, übermittelte Personenbewegungen und Sensordaten zu betrachten, sowie nötige Detailinformationen abzufragen. Bei Anhäufung zu vieler Daten im Sichtbereich ist es möglich, ähnliche Trajektorien zu gruppieren oder sich anormal verhaltende Trajektorien anzuzeigen. Darüberhinaus werden Alarmnachrichten empfangen und visualisiert, die in einer Sicherheitszentrale oder einer anderen Instanz des Projekts erzeugt werden. Das verwendete mobile Gerät wird einerseits mit bekannten und etablierten MultiTouch-Logiken, und andererseits mit speziell für den Einsatzzweck ausgearbeiteten Bedientechniken gesteuert.

Zum Abschluss der Arbeit wurde ein Benutzertest mit der Zielstellung durchgeführt, eine Bewertung der Applikation hinsichtlich Visualisierung, Interaktion und dem Erreichen der in Kapitel 2.3 festgelegten Anforderungen zu erhalten. Die Durchführung des Tests ergab allgemein ein äusserst positives Feedback, es konnten allerdings auch zahlreiche Verbesserungsvorschläge gewonnen werden, welche im Folgenden erläutert werden.

Picking

Eine wichtige Verbesserung kann in der *Picking*-Technik erfolgen, damit es Nutzern möglich wird, auch bei kleinen Objekten ohne starkes Hineinzoomen in die Szene eine erfolgreiche Auswahl tätigen zu können. Eine Lösung wäre, die gesamte Szene ein zweites Mal mit deutlich vergrößerten Objektausmaßen in einen zweiten, nicht dargestellten Framebuffer zu rendern. Das *Picking* erfolgt dann in diesem Framebuffer, wodurch auch bei ungenau platziertem Finger noch Treffer mit Objekten aus der eigentlichen Szene zustande kommen. Die dadurch steigenden Trefferzahlen erfordern eine neue Methodik der Ergebnisdarstellung - denkbar wäre, die Informationsdialoge der gepickten Objekte nicht, wie bisher, nacheinander darzustellen, sondern eine Übersicht aller Treffer dazwischen zu schalten (siehe Abb. 6.1). In dieser Übersicht sollten die Szenenobjekte nicht nur aufgrund ihres Namens, sondern möglichst auch mit ihrer Farbe beziehungsweise ihrem Icon dargestellt werden, da sich einige Proban-

den des Nutzertests bei solchen Auswahlmöglichkeiten mehr Information als nur den Objektnamen wünschten.

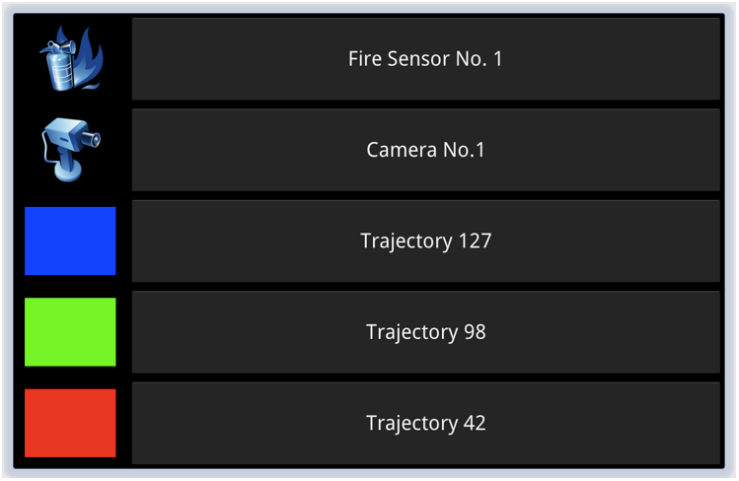


Abbildung 6.1.: Vorschlag Übersicht nach Picking-Mehrfachtreffern

Alarmvisualisierung

Die Visualisierung eines ankommenden Alarms bot einigen Test-Probanden nicht die nötige Aufälligkeit, um ohne den haptischen und audiellen Anteil eine Alarmnachricht zu bemerken. Eine Optimierung könnte hier erreicht werden, indem der entsprechende Dialog mit Alarminformationen direkt bei Eintreffen eines Alarms geöffnet wird. Ein dazu generiertes Schwarz/Weiss-Blinken des Bildschirms würde den Benutzer darüberhinaus auch auf den Alarm aufmerksam machen, wenn das Tablet nur im peripheren Erfassungsbereich der Augen vorhanden ist. Einige Testkandidaten schlugen ausserdem vor, bei Anwahl eines Alarms sofort die damit verbundene Trajektorie isoliert darzustellen, um nicht den Umweg über einen Alarmdialog gehen zu müssen, was ebenfalls als eine angebrachte Optimierung angesehen werden kann.

Im jetzigen Stand der Software wird ein Alarm komplett aus der Benutzeroberfläche entfernt, sobald er ignoriert oder bestätigt wird, was in den Augen vieler Nutzer irritierend war. Damit auch alle vergangenen Alarmaktionen betrachtet werden können, ist es daher sinnvoll, eine jederzeit betrachtbare Alarmhistorie auf dem Gerät vorzuhalten, wodurch es den Anwendern auch nach einer Fehlbedienung möglich wäre, die Verfolgung eines Alarms wieder aufzunehmen. In der gleichen Weise wäre es dadurch für den Benutzer realisierbar, seine vergangenen Tätigkeiten zu betrachten und auszuwerten.

Drilldown-Metapher

In vielen Situationen reklamierten die Benutzer eine zu starre Einhaltung der *Drill-Down*-Metapher, was bei Navigation aus einer tiefen Hierarchiestufe zur Kartendarstellung dazu führt, dass viele Dialoge hintereinander geschlossen werden müssen,

obwohl das Ziel schon im Hintergrund zu sehen ist. Eine Verbesserung dieser Situation kann zum Beispiel erreicht werden, indem in einer oberen Leiste jederzeit der Ausgangspunkt und die letzten Stationen der getätigten Navigationsschritte sichtbar sind. Durch Berühren eines dieser Felder kann der Benutzer dann direkt zum gewünschten Dialog oder zum Ausgangspunkt springen (für einen Vorschlag zur grafischen Umsetzung siehe Abb. 6.2), was die Geste zum Schliessen eines Dialogs (ausserhalb der Dialogfläche tippen) somit überflüssig machen wurde. Stattdessen kann davon ausgegangen werden, dass ein Nutzer bei Antippen des Hintergrunds auch wirklich zu diesem zurückkehren möchte, so dass diese Geste mit der Funktionalität des Zurückkehrens zur Kartendarstellung verbunden werden sollte. Im Rahmen der Navigationsoptimierungen könnte zusätzlich eine Funktion eingeführt werden, mit der ein Benutzer programmweit per Geste oder Schaltfläche zur letzten angewählten Darstellung gerät. Dies vermindert durch eine jederzeitige Wiederherstellungsfähigkeit die Hemmung, eine Falscheingabe durchzuführen und macht den Benutzer mutiger bei Erkundung und Benutzung der Software.

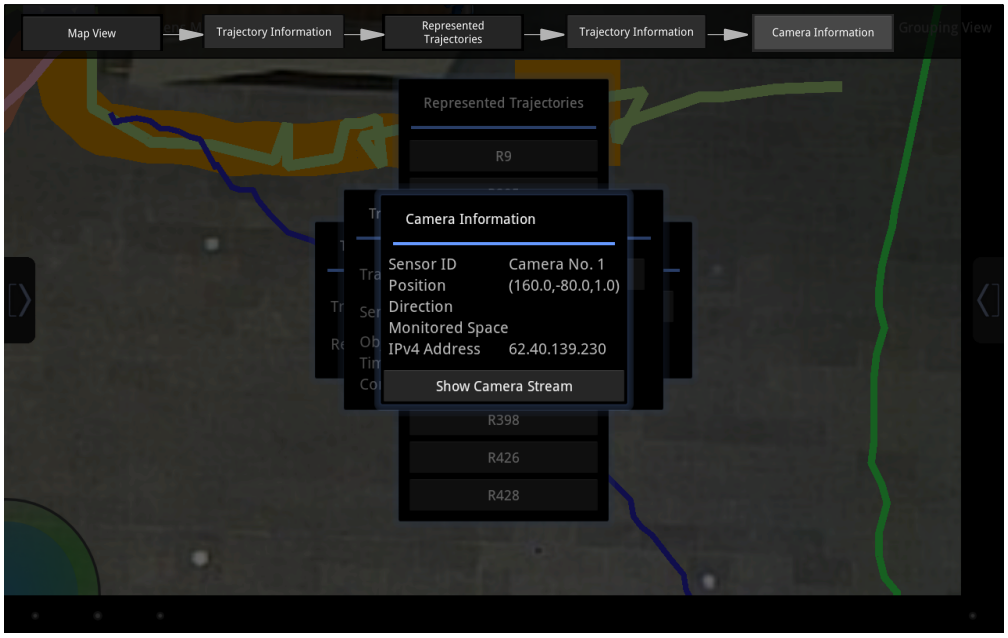


Abbildung 6.2.: Erweiterung des DrillDown durch Möglichkeit der direkten Schaltflächen-Navigation

Zusätzlich kann das *DrillDown* logischerweise abgebrochen werden, wenn der Nutzer die Kartenansicht als Folge einer gerade ausgeführten Aktion erwartet. So sollten deshalb etwa alle geöffneten Dialoge geschlossen werden, sobald der Benutzer eine Trajektorie isoliert oder eine andere Aktion ausführt, die ein 3D-Objekt manipuliert.

Visualisierung

Eine weitere Optimierung der Software kann erzielt werden, indem die Grafikbibliothek durch eine besser geeignete ersetzt wird. Im Laufe der Implementierung entstan-

den Situationen, in denen die schlechte Struktur von *jpct-AE* klar wurde. So werden etwa für Rotation, Translation und Skalierung verschiedene Matrizen bzw. Variablen verwendet und nicht etwa, wie in *OpenGL* üblich, eine zentrale Transformationsmatrix (siehe [OBJ]). Daher ist es aufgrund von Unkenntnis der internen Multiplikationsreihenfolge nicht möglich, die Abfolge von hintereinander ausgeführten Translationen, Rotationen oder Skalierungen festzulegen, was aber im Falle der Skalierung um einen bestimmten Punkt notwendig ist (siehe Kapitel 4.3.2). Dieses eine Beispiel dient nur als Repräsentation für viele weitere kleine Strukturschwächen, die nach und nach bekannt wurden, so dass sich innerhalb der Fraunhofer-Arbeitsgruppe in einem ähnlichen Projekt für *libgdx* [GDX] entschieden wurde. Der fehlende Szenegraph und die Picking-Funktionalität dieser Bibliothek wurden nachträglich implementiert, so dass diese modifizierte Software nun auch für die in dieser Arbeit vorgestellte Lösung einsetzbar wäre.

Abbildung 6.3.: Beispiel einer 3D-Rekonstruktion, Quelle: [REC]

Die Gruppierung von Trajektorien bildet eine weitere Optimierungsmöglichkeit, da sie derzeit nur mit einem stark vereinfachten Algorithmus stattfindet, um den Benutzern das Ergebnis trotz geringer Prozessorkraft in akzeptabler Zeit präsentieren zu können. Zukünftig könnten stattdessen die Gruppierungen per Netzwerk vom *Multi-Tracker* angefragt werden, denn dieser soll im weiteren Verlauf des Projekts selbständig Alarme generieren, wenn anormale Trajektorien auftreten (siehe auch [COLb]). Zu diesem Zweck werden in der *Multi-Tracker*-Instanz aufwendigere Gruppierungsalgorithmen benutzt, deren Ergebnisse dann auch gleichzeitig in der Sicherheitszentrale und dem mobilem Endgerät abgefragt werden könnten, was zudem eine bessere Kon-

sistenz zwischen den Visualisierungen der Sicherheitszentrale und des Tablets ergibt.

In einem späteren Stadium des Projekts sollen realistische 3D-Modelle (siehe Abb. 6.3) die Zylinderrepräsentation der derzeitigen 3D-Rekonstruktion ersetzen, und ebenso könnte die Software an vielerlei Stellen hinsichtlich ihrer 3D-Darstellung erweitert werden. Neben der Importierung und Anzeige von 3D-Modellen der Rekonstruktion, wäre es denkbar, ebenso ein 3D-Modell der Umgebung anstatt der aktuellen, zweidimensionalen Kartendarstellung zu importieren, um ein zusätzliches Plus an Immersion zu erreichen (siehe Abb. 6.4). In diesem Kontext wäre es auch möglich, die Bewegungspfade dreidimensional (beispielsweise durch aneinander gehängte Quader) darzustellen, um auch bei flachem Blickwinkel in die Szene eine bessere Übersicht zu erhalten.

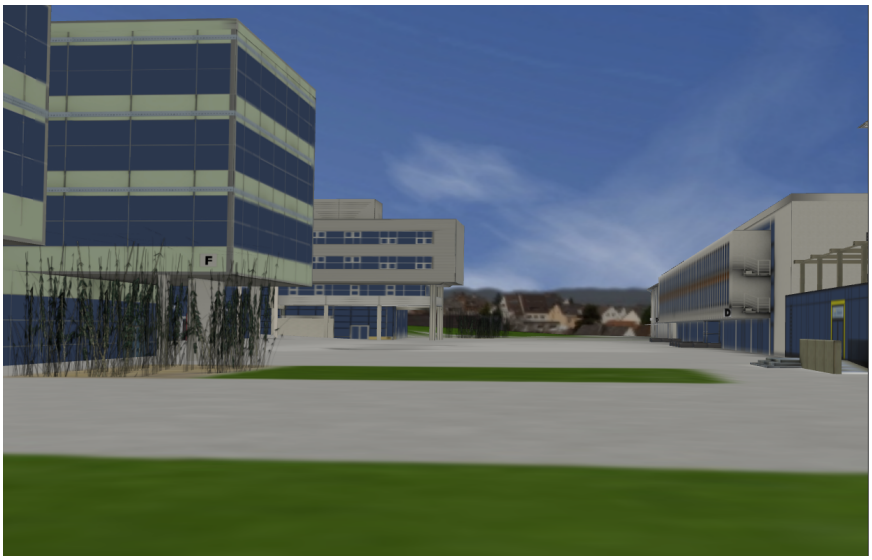


Abbildung 6.4.: 3D-Umgebungskarte, hier: Uni Koblenz, Quelle: [UNI]

Darüberhinaus sind noch viele kleinere Vorschläge aus den Ergebnissen des Nutzertests umsetzbar, wie etwa eine bessere Visualisierung des aktuell aktiven Betrachtungsmodus oder eine Einblendung aller verfügbaren *SingleTouch*-Gesten während der Durchführung einer Geste. Ebenso ist die Umsetzung des Vorschlags sinnig, die in einer Gruppe enthaltenen Trajektorien nicht nur in einer Liste, sondern auch visuell auf der Karte isoliert von den anderen betrachten zu können.

Andere Verbesserungen

Die Kommunikation zwischen Sicherheitszentrale und mobilem Personal spielt eine wichtige Rolle, welche bisher jedoch recht einseitig ausgelegt ist. So kann die Sicherheitszentrale in jeder Alarmnachricht zusätzlich eine Beschreibung des Alarms eintragen, ein genaueres Nachfragen des mobilen Mitarbeiters wird allerdings derzeit noch nicht unterstützt - er besitzt lediglich die Möglichkeit der Bestätigung oder Ignorierung. Zukünftig wäre es denkbar, einen Informationsaustausch zwischen Tabletbenutzer und Sicherheitszentrale erfolgen zu lassen, indem beispielsweise Sprach-

oder Textnachrichten ausgetauscht werden. Dieser Schritt könnte dazu führen, dass die gegenwärtig benutzten Funkgeräte oder Mobiltelefone komplett durch ein Tablet inklusive Software ersetzt werden.

Die Erweiterung des derzeitigen Softwarestands um diese vorgestellten Ideen wird zeigen, wie groß der daraus zusätzlich gewonnene Nutzen für den Anwender ist. Die sehr guten Ergebnisse des durchgeführten Nutzertests bestätigen in jedem Fall die Richtigkeit vieler getroffener Entscheidungen und dass sich eine Weiterentwicklung auf Basis der in dieser Arbeit beschriebenen mobilen Sicherheitslösung lohnt.

Literaturverzeichnis

[3S] *http://www.bahnhof.de/site/bahnhoefe/de/west/koeln/bahnhofsteam/bahnhofsteam.html*. 17.2.2012.

[ACM] *http://dl.acm.org/*. 23.1.2012.

[ACT] *http://activemq.apache.org*. 20.1.2012.

[AND] *http://www.android.com/*. 20.1.2012.

[ANF] *AP 1200: Anforderungsliste*. Internes Projektdokument.

[AVA] *http://www.avango.org/*. 20.1.2012.

[AVSa] *http://www.avss2010.org/, Year = 2010*. 23.1.2012.

[AVSb] *http://www.avss2011.org/, Year = 2011*. 23.1.2012.

[AVS07] *http://www.eecs.qmul.ac.uk/~andrea/avss2007.html*. 23.1.2012, 2007.

[AVS09] *http://www.avss09.org/*. 23.1.2012, 2009.

[AXIa] *http://www.axis.com/de/products/cam_214/*. 27.1.2012.

[AXIb] *http://www.axis.com/files/manuals/um_214_37546_en_0911.pdf*. 27.1.2012, Seite 11.

[BMB] *http://www.bmbf.de*. 15.8.2011.

[CAM] *http://www.caminsens.org, Förderkennzeichen 13N10809-814*. 12.2.2012.

[CAN] *http://developer.android.com/guide/topics/graphics/2d-graphics.html*. 20.1.2012.

[CCT] *http://en.wikipedia.org/wiki/Closed-circuit_television#Networking_CCTV_cameras*. 17.2.2012.

[COLa] *http://wiki.delphigl.com/index.php/Color_Picking*. 20.1.2012.

[COLb] *http://www.ikg.uni-hannover.de/index.php?id=620&L=1*. 16.2.2012.

[CON] *http://developer.android.com/reference/android/content/Context.html*. 30.1.2012.

[DAT] *http://de.wikipedia.org/wiki/Datenbank*. 16.2.2012.

[DET] *http://www.detec.no*. 16.2.2012.

[DIA] *http://developer.android.com/guide/topics/ui/dialogs.html*. 26.1.2012.

[DRI] *http://de.wikipedia.org/wiki/Drill-Down*. 31.1.2012.

[ea01] AL., PINGALI ET: *Visualization of Sports using Motion Trajectories: Providing Insights into Performance, Style, and Strategy*. 2001.

[Edi] *http://homepages.inf.ed.ac.uk/rbf/FORUMTRACKING/, Year = 2009*. 16.8.2011.

[EUK] *http://de.wikipedia.org/wiki/Euklidischer_Abstand*. 23.1.2012.

[Gam95a] GAMMA, E.: *Design patterns: elements of reusable object-oriented software*, Seiten 127–134. Addison-Wesley professional computing series. Addison-Wesley, 31 Auflage, 1995.

[Gam95b] GAMMA, E.: *Design patterns: elements of reusable object-oriented software*, Seiten 257–271. Addison-Wesley professional computing series. Addison-Wesley, 31 Auflage, 1995.

[Gam95c] GAMMA, E.: *Design patterns: elements of reusable object-oriented software*, Seiten 293–303. Addison-Wesley professional computing series. Addison-Wesley, 31 Auflage, 1995.

[GDX] *http://code.google.com/p/libgdx/*. 20.1.2012.

[GES] *http://developer.android.com/reference/android/view/GestureDetector.html*. 1.2.2012.

[GOL] *http://goldeneye.co.in*. 17.2.2012.

[GOT] *http://www.gotchanow.com*. 16.2.2012.

[GPR] *http://de.wikipedia.org/wiki/GPRS*. 20.2.2012.

[GPS] *http://de.wikipedia.org/wiki/Global_Positioning_System*. 5.2.2012.

[HAN] *http://developer.android.com/reference/android/os/Handler.html*. 1.2.2012.

[HAU] *http://de.wikipedia.org/wiki/Hausdorff-Metrik*. 23.1.2012.

[HC11] HINRICHS, UTA und SHEELAGH CARPENDALE: *Gestures in the wild: studying multi-touch gesture sequences on interactive tabletop exhibits*. In: *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, Seiten 3023–3032, New York, NY, USA, 2011. ACM.

[HDP10] HORNUNG, GERRIT, MONIKA DESOI und MATTHIAS POCS: *Biometric Systems in Future Preventive Scenarios - Legal Issues and Challenges*. In: BRÖMME, ARSLAN und CHRISTOPH BUSCH (Herausgeber): *BIOSIG*, Band 164 der Reihe *LNI*, Seiten 83–94. GI, 2010.

[HHWH11] HÖFERLIN, MARKUS, BENJAMIN HÖFERLIN, DANIEL WEISKOPF und GUNTHER HEIDEMANN: *Interactive Schematic Summaries for Exploration of Surveillance Video*. In: *Proceedings of the ACM International Conference on Multimedia Retrieval (ICMR '11)*, Seiten 9:1–9:8. ACM, 2011.

[HKP06] HAN, JIAWEI, MICHELINE KAMBER und JIAN PEI: *Data Mining: Concepts and Techniques, Second Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, 2 Auflage, 1 2006.

[HOR] `http://solrtest.bsz-bw.de/vufind/Record/DE-208338418237`. 16.2.2012.

[IBM] `http://www.ibm.com`. 17.2.2012.

[IKG] `http://www.ikg.uni-hannover.de`. 17.2.2012.

[IOSa] `http://www.apple.com/de/ios/`. 20.1.2012.

[IOSb] `http://www.iosb.fhg.de`. 17.2.2012.

[IPI] `http://www.ipi.uni-hannover.de`. 17.2.2012.

[IVE] `http://www.ivembh.de`. 17.2.2012.

[Jäh05] JÄHNE, B.: *Digitale Bildverarbeitung*. Springer, 2005.

[JMO] `http://jmonkeyengine.com/`. 20.1.2012.

[JNI] `http://docs.oracle.com/javase/6/docs/technotes/guides/jni/index.html`. 5.2.2012.

[JPC] `http://www.jpct.net/jpct-ae/`. 20.1.2012.

[JPGa] `http://goethe.ira.uka.de/seminare/redundanz/vortrag11/`. 20.2.2012.

[JPGb] `http://www.jpeg.org`. 20.2.2012.

[JPWH10] JÄHNEN, U., C. PAUL, M. WITTKE und J. HAHNER: *Multi-object tracking using feed-forward neural networks*. In: *Soft Computing and Pattern Recognition (SoCPaR), 2010 International Conference of*, Seiten 176 –181, dec. 2010.

[JVA] `http://www.jva-huenfeld.justiz.hessen.de/`. 17.2.2012.

[KAM] `http://www.zeitgeschichte-online.de/zol-Videoueberwachung-09-2010`. 15.8.2011.

[KAP] *http://de.wikipedia.org/wiki/Touchscreen#Kapazitive_Touchscreens*. 19.2.2012.

[KI] *http://de.wikipedia.org/wiki/Künstliche_Intelligenz*. 16.2.2012.

[KKS+09] KURITA, YUICHI, JUNYA KOBAYASHI, TSUYOSHI SUENAGA, KENTARO TAKEMURA, YOSHIO MATSUMOTO und TSUKASA OGASAWARA: *Personal identification and visualization of relationships by using human trajectories*. In: *Proceedings of the 2008 IEEE International Conference on Robotics and Biomimetics*, Seiten 548–553, Washington, DC, USA, 2009. IEEE Computer Society.

[KON] *http://online.wsj.com/article/SB10001424052748704538404574539910412824756.html*. 17.2.2012.

[KWI] *http://www.kwiksurveys.com*. 7.2.2012.

[Maj09] MAJECKA, B.: *Statistical models of pedestrian behaviour in the Forum*. Diplomarbeit, University of Edinburgh, 2009.

[MJGJ11] MARQUARDT, NICOLAI, RICARDO JOTA, SAUL GREENBERG und JOAQUIM A. JORGE: *The continuous interaction space: interaction techniques unifying touch and gesture on and above a digital surface*. In: *Proceedings of the 13th IFIP TC 13 international conference on Human-computer interaction - Volume Part III*, INTERACT'11, Seiten 461–476, Berlin, Heidelberg, 2011. Springer-Verlag.

[MJP] *http://de.wikipedia.org/wiki/Motion_JPEG*. 27.1.2012.

[Mon11] MONARI, EDUARDO: *Dynamische Sensorselektion zur auftragsorientierten Objektverfolgung in Kameranetzwerken*. Nummer 978-3-86644-729-5 in *Karlsruher Schriften zur Anthropomatik / Lehrstuhl für Interaktive Echtzeitsysteme, Karlsruher Institut für Technologie ; Fraunhofer-Inst. für Optronik, Systemtechnik und Bildauswertung IOSB Karlsruhe*. KIT Scientific Publishing, http://digbib.ubka.uni-karlsruhe.de/volltexte/1000024187, 1 Auflage, 2011.

[MOT] *http://developer.android.com/reference/android/view/MotionEvent.html*. 1.2.2012.

[MP11] MONARI, E. und T. POLLOK: *A real-time image-to-panorama registration approach for background subtraction using pan-tilt-cameras*. In: *Advanced Video and Signal-Based Surveillance (AVSS), 2011 8th IEEE International Conference on*, Seiten 237 –242, 30 2011-sept. 2 2011.

[MPE] *http://mpeg.chiariglione.org/standards/mpeg-4/mpeg-4.htm*. 27.1.2012.

[MUS] *http://de.wikipedia.org/wiki/Mustererkennung*. 16.2.2012.

[NDK] http://developer.android.com/sdk/ndk/index.html. 20.1.2012.

[OBJ] http://www.jpct.net/doc/com/threed/jpct/Object3D.html. 26.1.2012.

[ONDa] http://developer.android.com/reference/android/view/GestureDetector.OnDoubleTapListener.html. 1.2.2012.

[ONDb] http://developer.android.com/reference/android/view/GestureDetector.OnDoubleTapListener.html#onDoubleTap(android.view.MotionEvent). 1.2.2012.

[ONG] http://developer.android.com/reference/android/view/GestureDetector.OnGestureListener.html. 1.2.2012.

[ONSa] http://developer.android.com/reference/android/view/GestureDetector.OnDoubleTapListener.html#onSingleTapConfirmed(android.view.MotionEvent). 1.2.2012.

[ONSb] http://developer.android.com/reference/android/view/GestureDetector.OnGestureListener.html#onScroll(android.view.MotionEvent,android.view.MotionEvent,float,float). 1.2.2012.

[ONSc] http://developer.android.com/reference/android/view/GestureDetector.OnGestureListener.html#onSingleTapUp(android.view.MotionEvent). 1.2.2012.

[ONSd] http://developer.android.com/reference/android/view/ScaleGestureDetector.OnScaleGestureListener.html. 1.2.2012.

[ONT] http://developer.android.com/reference/android/view/View.OnTouchListener.html. 1.2.2012.

[OPE] http://www.khronos.org/opengles/. 20.1.2012.

[OSG] http://www.openscenegraph.org/. 20.1.2012.

[PETa] http://www.cvg.rdg.ac.uk/PETS2006/data.html, *Year = 2006*. 16.8.2011.

[PETb] http://www.cvg.rdg.ac.uk/PETS2007/data.html, *Year = 2007*. 16.8.2011.

[PETc] http://www.cvg.rdg.ac.uk/PETS2009/, *Year = 2009*. 16.8.2011.

[PM02] PAVLIDIS, I. und V. MORELLAS: *Two examples of indoor and outdoor surveillance systems*. In: P. REMAGNINO, J. A. GRAEME, N. PARAGIOS C. REGAZZONI (Herausgeber): *Video Based Surveillance Systems: Computer Vision and Distributed Processing*, Seiten 39–50. Kluwer Academic Publishers, January 2002.

[PMTH01] PAVLIDIS, I., V. MORELLAS, P. TSIAMYRTZIS und S. HARP: *Urban surveillance systems: from the laboratory to the commercial world*. Proceedings of the IEEE, 89(10):1478 –1497, oct 2001.

[PRI] `http://wiki.delphigl.com/index.php/glBegin#Beschreibung`. 24.1.2012.

[PRO09] *Projektantrag*. Internes Projektdokument, 11 2009.

[REC] `http://www-home.htwg-konstanz.de/~umlauf/publications.html`. 12.2.2012.

[RUN] `http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Runnable.html`. 1.2.2012.

[SCA] `http://developer.android.com/reference/android/view/ScaleGestureDetector.html`. 1.2.2012.

[SET] `http://developer.android.com/reference/android/view/View.html#setOnTouchListener(android.view.View.OnTouchListener)`. 1.2.2012.

[SG09a] SHREINER, DAVE und THE KHRONOS OPENGL ARB WORKING GROUP: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, Seiten 614–624. Addison-Wesley Professional, 7th Auflage, 2009.

[SG09b] SHREINER, DAVE und THE KHRONOS OPENGL ARB WORKING GROUP: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, Seiten 42–53. Addison-Wesley Professional, 7th Auflage, 2009.

[SG09c] SHREINER, DAVE und THE KHRONOS OPENGL ARB WORKING GROUP: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, Seiten 267–280. Addison-Wesley Professional, 7th Auflage, 2009.

[SG09d] SHREINER, DAVE und THE KHRONOS OPENGL ARB WORKING GROUP: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, Seiten 259,504. Addison-Wesley Professional, 7th Auflage, 2009.

[SG09e] SHREINER, DAVE und THE KHRONOS OPENGL ARB WORKING GROUP: *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Versions 3.0 and 3.1*, Seite 294. Addison-Wesley Professional, 7th Auflage, 2009.

[SIM] *http://www.sim.uni-hannover.de*. 17.2.2012.

[SOR] *http://www.jpct.net/doc/com/threed/jpct/Object3D.html#setSortOffset(float)*. 26.1.2012.

[SRAa] *http://www.sra.uni-hannover.de*. 17.2.2012.

[SRAb] *http://www.sra.uni-hannover.de/mediawiki/, internes Projektwiki*. 17.2.2012.

[STO] *http://stomp.github.com/*. 20.1.2012.

[SvdCIS09] SCHICK, ALEXANDER, FLORIAN VAN DE CAMP, JORIS IJSSELMUIDEN und RAINER STIEFELHAGEN: *Extending touch: towards interaction with large-scale surfaces*. In: *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, ITS '09, Seiten 117–124, New York, NY, USA, 2009. ACM.

[SYN] *http://www.iks.hs-merseburg.de/~uschroet/Literatur/Java_Lit/JAVA_Insel/javainsel_11_005.htm#mj7c6c2a3b6c349b93c6b963acf3f596f2*. 27.1.2012.

[SZE] *http://de.wikipedia.org/wiki/Szenengraph*. 20.1.2012.

[TAK] *http://docs.oracle.com/javase/1.5.0/docs/api/java/util/concurrent/LinkedBlockingQueue.html#take()*. 6.2.2012.

[THR] *http://openbook.galileodesign.de/javainsel5/javainsel09_000.htm#Rxx747java09000040002C41F02A100*. 20.2.2012.

[TOA] *http://developer.android.com/guide/topics/ui/notifiers/toasts.html*. 30.1.2012.

[TOU] *http://www.panasonic.com/business/toughpad/us/best-android-rugged-tablet-overview.asp*. 20.1.2012.

[TRA] *http://de.wikipedia.org/wiki/Trajektorie_(Physik)*. 15.8.2011.

[UED] *http://de.wikipedia.org/wiki/Überwachungsdruck*. 12.2.2012.

[UMT] *http://de.wikipedia.org/wiki/UMTS*. 20.2.2012.

[UNI] *http://erkunden.uni-koblenz.de/icw/*. 12.2.2012.

[VIE] *http://developer.android.com/resources/tutorials/views/index.html*. 20.1.2012.

[VIT] *http://www.vitracom.de*. 17.2.2012.

[VR] *http://de.wikipedia.org/wiki/Virtuelle_Realität*. 20.1.2012.

[VV05] VALERA, M. und S.A. VELASTIN: *Intelligent distributed surveillance systems: a review*. Vision, Image and Signal Processing, IEE Proceedings -, 152(2):192 – 204, april 2005.

[WCRI09] WANG, FENG, XIANG CAO, XIANGSHI REN und POURANG IRANI: *Detecting and leveraging finger orientation for interaction with direct-touch surfaces*. In: *Proceedings of the 22nd annual ACM symposium on User interface software and technology*, UIST '09, Seiten 23–32, New York, NY, USA, 2009. ACM.

[WLA] `http://de.wikipedia.org/wiki/Wireless_Local_Area_Network`. 20.2.2012.

[WMW09] WOBBROCK, JACOB O., MEREDITH RINGEL MORRIS und ANDREW D. WILSON: *User-defined gestures for surface computing*. In: *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, Seiten 1083–1092, New York, NY, USA, 2009. ACM.

[XML] `http://de.selfhtml.org/xml/intro.htm`. 5.2.2012.

[ZFI] `http://wiki.delphigl.com/index.php/ZFighting`. 26.1.2012.

[ZZTM10] ZHONG, CHEN, CHAMSEDDINE ZAKI, VINCENT TOURRE und GUILLAUME MOREAU: *Event-based semantic visualization of trajectory data in urban city with a space-time cube*. In: *Proceedings of the 3rd WSEAS international conference on Visualization, imaging and simulation*, VIS '10, Seiten 99–105, Stevens Point, Wisconsin, USA, 2010. World Scientific and Engineering Academy and Society (WSEAS).

A. Anhang

A.1. Fragebogen des Benutzertests

1. Es wird nun ein Szenario abgespielt.
Untersuchen Sie jeweils die Trajektorie, die den Alarm auslöst.
a) Versuchen Sie, die Trajektorie isoliert von den anderen zu betrachten.
b) Betrachten Sie den dazugehörigen Videostream.
c) Bestätigen Sie die Bearbeitung des Alarms.
d) Kehren Sie danach zurück in die Live-Ansicht.

2. Haben Sie die Aufgabe gelöst?

- ○ **Ja**
- ○ **Nein**

3. Wie gut wurden Sie über die Alarme informiert?

haptisch	○	○	○	○	○
visuell	○	○	○	○	○
audiell	○	○	○	○	○

Was könnte verbessert werden?

4. Sie sehen nun die Trajektoriendaten eines gesamten Tages.
Versuchen Sie, Gefahren mit Hilfe des Programmes zu erkennen und untersuchen Sie die entsprechenden Trajektorien.
Versuchen Sie dabei insbesondere, dominante Bewegungsrichtungen und sich anormal verhaltende Trajektorien zu untersuchen.
a) Versuchen Sie, jeweils eine anormale und eine Trajektorie aus einer dominanten Bewegungsrichtung isoliert von den anderen zu betrachten.
b) Betrachten Sie den dazugehörigen Videostream.
c) Kehren Sie danach zurück in die Live-Ansicht.

5. Haben Sie die Aufgabe gelöst?

- Ja
- Nein

Was könnte verbessert werden?

6. Hat die Visualisierung Sie dabei unterstützt, die Aufgaben effektiv und effizient zu lösen?

- Trifft absolut zu
- Trifft zu
- Weder noch
- Trifft nicht zu
- Trifft absolut nicht zu

Was könnte verbessert werden?

7. War die Interaktion intuitiv, d.h. entsprach die Interaktion Ihren Erfahrungen und Erwartungen?

- Trifft absolut zu
- Trifft zu
- Weder noch
- Trifft nicht zu
- Trifft absolut nicht zu

Was könnte verbessert werden?

Allgemeine Fragen zur Software

8. Nutzten Sie die Gestensteuerung oder das Ausklappmenü?

- Ausschliesslich das Ausklappmenü
- Eher das Ausklappmenü
- Beides gleich
- Eher die Gestensteuerung
- Ausschliesslich die Gestensteuerung

9. Verwenden Sie regelmässig Multitouch-Anwendungen?

- Trifft absolut zu
- Trifft zu
- Weder noch
- Trifft nicht zu
- Trifft absolut nicht zu

10. Verwenden Sie regelmässig kartenbasierte Anwendungen (z.B. Google Maps)?

- Trifft absolut zu
- Trifft zu
- Weder noch
- Trifft nicht zu
- Trifft absolut nicht zu

11. Allgemeine Verbesserungsvorschläge:

A.2. Ergebnisse des Benutzertests

Results for survey: CamInSens Benutzertest

Page: 1/6

Question 1

Es wird nun ein Szenario abgespielt.
Untersuchen Sie jeweils die Trajektorie, die den Alarm auslöst.
a) Versuchen Sie, die Trajektorie isoliert von den anderen zu betrachten.
b) Betrachten Sie den dazugehörigen Videostream.
c) Bestätigen Sie die Bearbeitung des Alarms.
d) Kehren Sie danach zurück in die Live-Ansicht.

No answers required.

Page: 2/6

Question 2

Haben Sie die Aufgabe gelöst?

Ja	12	100.00%
Nein	0	0.00%

Question 3

Wie gut wurden Sie über die Alarme informiert?

	1	2	3	4	5	Responses	Total
haptisch	0%	0%	0%	**33%**	**67%**	**12**	**56**
visuell	0%	**8%**	**17%**	**42%**	**33%**	**12**	**48**
audiell	0%	0%	0%	0%	**100%**	**12**	**60**

ID	View Survey	Was könnte verbessert werden?
10261374	View	Automatisches fokusieren der Karte anhand der Alarmposition, falls diese nicht schon im Sichtfeld liegt.
10265621	View	Nach Alarmoeffnung direkt Anzeige der Trajektorie, Ausblenden aller anderen T.
10393544	View	noch mehr vibration, bitte
10393691	View	Längere Vibration,
10393870	View	stärkere vibration
10394117	View	direkte visualisierung der trajektorie, die den alarm ausgelöst hat wäre besser
10394474	View	größeres alarmfenster
10394651	View	stärkere vibration könnte blinken
10435681	View	auffälligere visualisierung des alarms, mehr bewegung.
10461442	View	mehr vibration, rot leuchtendes display bei alarm

Page: 3/6

Question 4

Sie sehen nun die Trajektoriendaten eines gesamten Tages.
Versuchen Sie, Gefahren mit Hilfe des Programmes zu erkennen und untersuchen Sie die entsprechenden Trajektorien.
Versuchen Sie dabei insbesondere, dominante Bewegungsrichtungen und sich anormal verhaltende Trajektorien zu untersuchen.
a) Versuchen Sie, jeweils eine anormale und eine Trajektorie aus einer dominanten Bewegungsrichtung isoliert von den anderen zu betrachten.
b) Betrachten Sie den dazugehörigen Videostream.
c) Kehren Sie danach zurück in die Live-Ansicht.

This question has no answers

Page: 4/6

Question 5

Haben Sie die Aufgabe gelöst?

Ja	11		100.00%
Nein	0		0.00%

ID	View Survey	Was könnte verbessert werden?
10261374	View	1) Wenn eine Trajektorie isoliert wird, sollten die Kontextmenues geschlossen werden. 2) Ein Zurueckknopf, bzw. Geste um in den vorherigen View bzw. Modus zu gelangen (aehnlich zu einem Webbrowser) 3) Picking der Trajektorien bzw. Cluster/Anaomalien sollte nicht abhaengig des Zoomlevels sein..
10263659	View	Man koennte die Trajektorien in einem Cluster zusaetzlich zu der Listendarstellung visuell anzeigen.

Page: 5/6

Question 6

Hat die Visualisierung Sie dabei unterstützt, die Aufgaben effektiv und effizient zu lösen?

Trifft absolut zu	7	63.64%
Trifft zu	4	36.36%
Weder noch	0	0.00%
Trifft nicht zu	0	0.00%
Trifft absolut nicht zu	0	0.00%

ID	View Survey	Was könnte verbessert werden?
10261374	View	siehe Freitextfeld von Fragen zuvor
10265621	View	Hilighting von ausgewaehlten Pfaden / Gruppen
10393870	View	Button fehlt für direkte Kameraansicht, evtl kurze Ortsangabe
10394117	View	direkte visualisierung der möglichen gesten
10394474	View	die einzelnen trajektorien könnten beim anklicken farblich unterschieden werden, falls mehrere trajektorien an demselben punkt enden.
10435681	View	nach doubletap touchpunkt mittig zentrieren, alarm historie, besseres picking der trajektorien

Question 7

War die Interaktion intuitiv, d.h. entsprach die Interaktion Ihren Erfahrungen und Erwartungen?

Trifft absolut zu	9	81.82%
Trifft zu	1	9.09%
Weder noch	1	9.09%
Trifft nicht zu	0	0.00%
Trifft absolut nicht zu	0	0.00%

ID	View Survey	Was könnte verbessert werden?
10261374	View	siehe Freitextfeld von Fragen zuvor
10265621	View	Auswahlrechteck fuer Trajektorien

Page: 6/6

Allgemeine Fragen zur Software

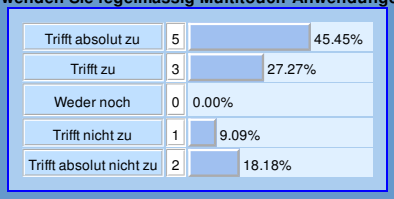
Question 8

Nutzten Sie die Gestensteuerung oder das Ausklappmenü?

Ausschliesslich das Ausklappmenü	1	9.09%
Eher das Ausklappmenü	1	9.09%
Beides gleich	2	18.18%
Eher die Gestensteuerung	4	36.36%
Ausschliesslich die Gestensteuerung	3	27.27%

Question 9

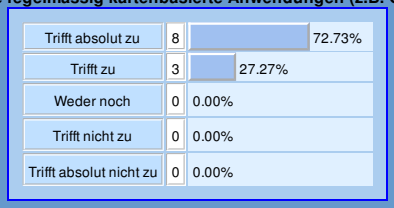
Verwenden Sie regelmässig Multitouch-Anwendungen?



Trifft absolut zu	5	45.45%
Trifft zu	3	27.27%
Weder noch	0	0.00%
Trifft nicht zu	1	9.09%
Trifft absolut nicht zu	2	18.18%

Question 10

Verwenden Sie regelmässig kartenbasierte Anwendungen (z.B. Google Maps)?



Trifft absolut zu	8	72.73%
Trifft zu	3	27.27%
Weder noch	0	0.00%
Trifft nicht zu	0	0.00%
Trifft absolut nicht zu	0	0.00%

Question 11

Allgemeine Verbesserungsvorschläge:

ID	Text Answers (7)	View
10461442	alle verfügbaren gesten anzeigen	View
10435681	dialoge nicht nacheinander schliessen, sondern direkt zum richtigen dialog springen können	View
10394651	pan und pinch nur wenn man die karte anfasst. rechter handle unnoetig wenn keine alarme	View
10394117	direkte anzeige von informationen zu der position an dem die trajektorie betatscht wird -> z.b. popup	View
10393870	Nummerierung der einkommenden Alarme. Möglichkeit alte Alarme in Liste zu betrachten.	View
10393544	noch mehr gesten	View
10261374	1) Der aktuelle View deutlicher hervorheben/visualisieren. 2) Die Heatmap Visualisierung der Trajektorien gibt einen sehr guten Uerblick ueber die dominanten Aufenthaltsbereiche des zu ueberwachenden Bereiches (nicht auf Livedaten).	View