UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Initialisierung im markerlosen Tracking mit dem Analyse-durch-Synthese Ansatz in einem urbanen Kontext

Diplomarbeit

zur Erlangung des Grades eines Diplom-Informatikers
im Studiengang Computervisualistik

vorgelegt von
Sebastian Kowalczyk

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergrafik)
Zweitgutachter:	Dipl.-Inf. Martin Schumann (Institut für Computervisualistik, AG Computergrafik)

Koblenz, im Februar 2012

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....
(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

Teil I
Einleitung

1 Motivation und Zielsetzung

Ein interessantes Forschungsgebiet der Computervisualistik ist das Konzept der *Erweiterten Realität*[1]. Ziel ist es, die Realität mit Einblendungen virtueller Objekte zu ergänzen. Potentielle Anwendungsszenarien reichen dabei von Systemen zur Wartung komplexer Maschinen über touristische Anwendungen, bei denen beispielsweise zerstörte Gebäude virtuell an der korrekten Stelle in ihrem ursprünglichen Zustand eingeblendet werden, bis hin zu Spielesystemen auf mobilen Endgeräten.

Eine besondere Herausforderung bei der Realisierung von AR-Systemen ist dabei das sogenannte *Tracking*[2]. Vereinfacht beschrieben handelt es sich um eine Sammlung von Verfahren, welche die Position und Orientierung eines realen Betrachters schätzen und auf einen virtuellen Betrachter anwenden. Dadurch können virtuelle Objekte unter Berücksichtigung der aktuellen Position und Orientierung[3] eines realen Betrachters zur Laufzeit stets korrekt in die reale Szene eingeblendet werden. Bewegt sich ein Betrachter in der Realität, so soll sich auch automatisch die Pose der virtuellen Kamera anpassen. Das virtuelle Objekt verändert im Gegensatz zur Kamerapose weder seine Position noch seine Orientierung in der realen Szene. Bricht aus bestimmten Gründen die Verfolgung der Betrachterpose in der realen Umgebung ab, so kann auch die Pose des virtuellen Betrachters nicht weiter aktualisiert werden und das Tracking schlägt fehl.

Essentiell für die Qualität des Trackings ist nicht nur die Wahl eines geeigneten Verfahrens, sondern auch die Abschätzung einer ersten Pose des Betrachters zur Initialisierung des Trackings. Je nach Wahl des Tracking-Verfahrens werden diese Daten allerdings nicht automatisch generiert.

Ziel dieser Arbeit ist die Entwicklung und Implementierung eines Verfahrens zur initialen Schätzung einer Betrachterpose in einem urbanen Kontext. Auch eine erneute Initialisierung bei fehlgeschlagenem Tracking soll ermöglicht werden. Das entwickelte System soll weitestgehend automatisch arbeiten. Neben der Konzeption einer intuitiven Benutzerführung sollte die performante Implementierung besonders im Fokus der Entwicklung stehen. Die Berechnung einer neuen initialen Pose bei fehlgeschlagenem Tracking muss zwingend so schnell wie möglich geschehen, damit der bei AR-Systemen gewünschte hohe Grad der *Immersion*[4] erhalten bleibt.

[1] Augmented Reality (AR)
[2] aus dem Englischen übersetzt: „Verfolgen“
[3] zusammen auch Pose genannt
[4] Realität und Virtualität verschmelzen zu einer Einheit

2 Aufbau der Arbeit

Im Folgenden werden zu Beginn einige Grundlagen, welche zum Verständnis dieser Arbeit notwendig sind, erläutert. Es werden einige Tracking-Konzepte vorgestellt, von denen die für diese Arbeit potentiell interessanten Techniken im Anschluss genauer beschrieben werden. Auch die genauere Betrachtung einiger wichtiger Algorithmen der Bildverarbeitung ist dazu notwendig.

Der wesentliche Teil dieser Arbeit beschreibt die Planung und Implementierung des Systems. Dabei wird zuerst auf die benötigte Hard- und Software eingegangen. Nach einer Beschreibung der Systemarchitektur werden die verwendeten Konzepte zur Bestimmung und Verwaltung der benötigten Daten vorgestellt. Im Anschluss wird die initiale Poseschätzung in zwei Stufen erarbeitet. Nachdem die Entwicklung des Systems vollständig erfasst wurde, können in einem nächsten Schritt die Ergebnisse präsentiert werden. Besondere Aufmerksamkeit soll dabei der Präzision der Poseschätzung und dem Grad der Performanz gewidmet werden.

Den Abschluss dieser Arbeit bildet eine Zusammenfassung und ein Ausblick auf mögliche Weiterentwicklungen des beschriebenen Systems.

Teil II
Grundlagen

3 Tracking

Robustes Tracking stellt eine Grundvoraussetzung zur Realisierung von AR-Systemen dar. Bei der Einblendung virtueller Objekte in das Sichtfeld eines Betrachters ist eine lagerichtige Platzierung an der korrekten Position in der realen Szene von entscheidender Bedeutung. Um dieser Forderung nachzukommen, muss die Pose des Betrachters in der realen Szene bekannt sein. Im weiteren Sinne versteht man unter Tracking jedoch nicht nur die Schätzung und Verfolgung einer Betrachterpose. Der Begriff ist allgemeiner zu verstehen. Tracking bedeutet vielmehr das Erkennen und Verfolgen bestimmter Objekte in Bildsequenzen. Eine Betrachterpose kann wiederum aus dem Wissen über die Position und Ausrichtung eines erkannten Objektes in der Welt bestimmt werden.

Im Anwendungsgebiet der Erweiterten Realität werden geeignete Objekte innerhalb einer realen Szene getrackt, sodass eine Interaktion zwischen Realität und Virtualität ermöglicht wird. Bei der Verwendung eines Datenhandschuhs als alternatives Eingabegerät zur Steuerung von Computersystemen ist beispielsweise die Position der Hand in Relation zum Bildschirm von entscheidender Bedeutung. Ein weiteres bekanntes Beispiel ist das sogenannte Eye-Tracking. Dabei trägt der Benutzer eine Brille, welche mit einem Sensor zur Erfassung der Pupillenposition ausgestattet ist. Somit können Rückschlüsse auf die aktuelle Blickrichtung des Betrachters gezogen werden. Im Folgenden sollen einige der wichtigsten Techniken und Konzepte unter dem Begriff des Trackings aufgezeigt werden.

Eine im Alltag häufig genutzte Art des Trackings ist die Verwendung von GPS-Navigationsgeräten. Dabei wird die momentane Position des Gerätes über ein Netzwerk von Satelliten ermittelt. Da diese Technik auch in der Realisierung dieser Arbeit Anwendung findet, wird sie in einem nachfolgenden Abschnitt detaillierter erklärt.

Das wohl bedeutendste Tracking-Verfahren im Forschungsgebiet der Erweiterten Realität ist das optische Tracking. Dabei dienen lediglich Kamerabilder als Eingabe zur Rekonstruktion einer Kamerapose in der realen Szene. Zu unterscheiden sind *markerbasierte* und *markerlose* Verfahren.

Das markerbasierte Tracking verwendet eine Reihe von frei wählbaren Mustern, die im Kamerabild erkannt werden müssen. Eine Grundvoraussetzung ist dabei, dass die Ausrichtung dieses Musters stets eindeutig bestimmt werden kann. Der Benutzer muss nun die Position und Orientierung dieser Marker innerhalb der virtuellen Szene definieren und die besagten Marker dann entsprechend in der realen Umgebung anbringen.

Wird eine Mindestanzahl der Muster von der Kamera erfasst, so kann aus den Informationen über die Lage der Marker eine Position und Ausrichtung des realen Betrachters geschätzt werden. Sind reale und virtuelle Kamera deckungsgleich, wurde die Pose der realen Kamera erfolgreich durch die virtuelle Kamera rekonstruiert. Eine Bibliothek zum Erstellen von markerbasierten AR-Systemen ist das ARToolKit [9].

Markerloses Tracking arbeitet nach demselben Prinzip, verzichtet dabei aber auf die angesprochenen definierten Marker. Lediglich das Kamerabild dient als Anhaltspunkt für die Pose eines realen Betrachters. Im Gegensatz zum markerbasierten Ansatz gibt es noch keine allgemeingültige Lösung für derartige Probleme. Mögliche Konzepte müssen immer speziell an das Anwendungsszenario angepasst werden.

Zuletzt sei noch ein Tracking-Verfahren genannt, welches in einer bekannten Spielekonsole seine Anwendung findet. Konkret ist Nintendos Wii [16] gemeint, welche zum Tracking von Bewegungsabläufen auf die Verwendung von Trägheitssensoren setzt. Innerhalb des Eingabegerätes messen diese Sensoren die Beschleunigung, die ein Benutzer auf das Eingabegerät ausübt. Mit dieser Technik werden unter anderem authentische Simulationen von Sportarten wie Tennis oder Golf ermöglicht.

4 Global Positioning System

Das *Global Positioning System* (GPS) definiert ein Ortungssystem, welches seit Mitte der 90er Jahre durch das U.S. Department of Defense betrieben wird. Anhand eines Netzwerks von Satelliten lässt sich die Position eines GPS-Empfängers an jedem Ort auf der Erdoberfläche bestimmen. Voraussetzung dafür ist nach [25] allerdings ein direkter Kontakt zu mindestens vier Satelliten. Das System funktioniert weder in Gebäuden noch bei zu stark bedecktem Himmel. Des Weiteren muss die Position der Satelliten zum Zeitpunkt der Messung immer bekannt sein. Nach Xu [8] wird die Position eines GPS-Empfängers anhand der Abstände des Empfängers zu den einzelnen Satelliten des Netzwerks berechnet. Eine Positionsbestimmung wird umso genauer, je mehr Satelliten bei der Berechnung berücksichtigt werden können. Durch die kontinuierliche Bestimmung einer Position können auch Geschwindigkeiten über die Positionsänderung innerhalb bestimmter Zeitintervalle errechnet werden.

Xu [8] beschreibt weiterhin, dass das GPS-Netzwerk aus 24 Satelliten besteht, die auf sechs verschiedene Umlaufbahnen verteilt sind. Dadurch soll gewährleistet werden, dass zu jedem Zeitpunkt an jeder Position auf der Erdoberfläche eine GPS-Ortung möglich ist. Zur Kommunikation zwischen den Satelliten und einem GPS-Empfänger dienen nach [8] fünf auf der Erdoberfläche verteilte Basisstationen. Im Gegensatz zu [8] wird in [25] von 27 anstatt 24 aktiven Satelliten gesprochen. Laut [25] bietet GPS bei

der Verwendung herkömmlicher Empfangsgeräte momentan eine auf ca. 20 Meter präzise Positionsbestimmung. Es wird auch beschrieben, dass die Genauigkeit einer Positionsberechnung nicht nur von der Anzahl der verwendeten Satelliten, sondern ebenso vom GPS-Empfänger selbst abhängt. GPS-Geräte sind im freien Handel in verschiedenen Präzisionskategorien erhältlich. Theoretisch wäre eine zentimetergenaue Positionsbestimmung unter idealen Bedingungen mit entsprechenden Geräten möglich.

Da sich diese Arbeit mit der initialen Schätzung einer Betrachterpose in einem urbanen Kontext beschäftigt, eignet sich GPS hervorragend, um die Position eines Betrachters automatisch bestimmen zu lassen. Mit der Verwendung dieser Technik wird eine erste grobe Poseschätzung an jedem Ort der Welt möglich. Dabei muss allerdings bedacht werden, dass weder die Blickrichtung des Betrachters ermittelt noch eine ausreichend genaue Positionsbestimmung garantiert werden kann.

4.1 Grundlagen zur Darstellung von Positionen

Mit GPS lassen sich Positionen auf der Erdoberfläche bestimmen. Die Erde selbst ist jedoch keine perfekte geometrische Kugel. Es gibt eine Reihe von Bezugssystemen, welche die Form der Erdkugel adäquater beschreiben. Zwei häufig verwendete Bezugssysteme sind *WGS'84* und *ETRS'89*.

Das World Geodetic System (WGS'84) [28] wurde durch die U.S. Defense Mapping Agency entwickelt und 1984 veröffentlicht. Anwendung findet WGS'84 vor allem bei der Realisierung von GPS-Systemen. Die Koordinaten eines beliebigen Ortes auf der Erdkugel werden durch ein dreidimensionales, kartesisches Koordinatensystem beschrieben. Dem Bezugssystem liegt weiterhin ein sogenannter Referenzellipsoid zugrunde. Dieser geometrische Körper soll die tatsächliche Form der Erde optimal approximieren.

Das European Terrestrial Reference System (ETRS'89) [23] wurde 1989 durch die Europäische Subkommission für Referenzrahmen (EUREF) eingeführt. Auch in diesem Bezugssystem wird die Erde durch einen eigenen Referenzellipsoiden beschrieben.

Für eine numerische Positionsdarstellung auf der Erdoberfläche gibt es eine Reihe von Möglichkeiten. Die wahrscheinlich bekannteste Darstellungsform ist die Angabe von Längen- und Breitengraden. Weiterhin sind noch sowohl die Gauß-Krüger- als auch die UTM-Darstellung zu nennen.

Der Darstellung mit *Längen- und Breitengraden* [26] liegt eine gleichmäßige Verteilung von die Erde umspannenden, parallel zueinander liegenden Kreisen zugrunde. Breitengrade verlaufen dabei stets zwischen dem Äquator und den Polen. Der Äquator bildet den nullten Breitengrad. Zu den Polen hin steigen die Werte gleichmäßig bis auf 90 Grad an. Jeder Längengrad (auch *Meridian* genannt) verläuft durch beide Pole. Der Wertebereich der Längengrade reicht von 0 bis 180 Grad. Durch diese gleichmäßige Verteilung von Längengraden ist der Abstand der einzelnen Meri-

diane nicht an jeder Position auf der Erde gleich. Zu den Polen hin wird der Abstand zwischen zwei Meridianen immer geringer. Für eine Umrechnung von Längen- und Breitengraden in ein kartesisches Koordinatensystem sind also für jeden Ort der Welt spezielle Umrechnungsfaktoren nötig.

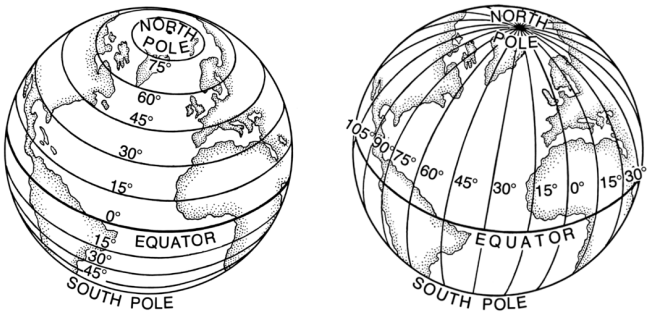

Abbildung 1: Breiten- und Längengrade aus [22] (v.l.n.r.)

Das von der NATO entwickelte *UTM-Koordinatensystem* basiert auf einer *universalen transversalen Mercator-Projektion* [27]. Dabei wird der Erdkörper senkrecht zum Äquator mit einem Zylinder geschnitten, der einen kleineren Umfang als die Erde selbst besitzt. Das Koordinatensystem ist zwischen 80 Grad nördlicher und 80 Grad südlicher Breite als international verwendetes System anerkannt. Deutschland beschloss 1995 einen Umstieg von der bisher verwendeten Gauß-Krüger- zur UTM-Darstellung. Der große Vorteil der UTM-Darstellung ist die Aufteilung der Erdoberfläche in einzelne Zonen, wodurch das Koordinatensystem die Eigenschaft der Orthogonalität erhält. Dabei entspricht eine UTM-Einheit sowohl in x- als auch in y-Richtung innerhalb einer Zone stets einem Meter. Die Genauigkeit dieser Umrechnung ist in der geometrischen Mitte einer Zone am stärksten. Zu den Seiten hin nimmt der Fehler zu. Entlang der Breitengrade existiert eine Aufteilung in 60 Zonen mit einer Breite von je sechs Längengraden. Innerhalb einer der festgelegten Zonen ist der Abstandsunterschied zwischen den einzelnen Längengraden so klein, dass er als konstant angenommen werden kann. Die x-Achse verläuft exakt auf dem Äquator und definiert den Abstand eines Punktes zum Mittelmeridian einer Zone in Metern. Der y-Wert gibt dementsprechend den Abstand zum Äquator wieder. Um negative x-Werte westlich eines Mittelmeridians zu vermeiden, wird der Koordinatenursprung per Definition um 500.000 Meter parallel zur x-Achse verschoben. Dies bezeichnet man als „false easting“. Da in der südlichen Hemisphäre der y-Wert negativ sein müsste, wird dieser um 10.000.000 Meter erhöht („false northing“). Abbildung 3 zeigt die Zonenaufteilung des UTM-Koordinatensystems im europäischen Raum.

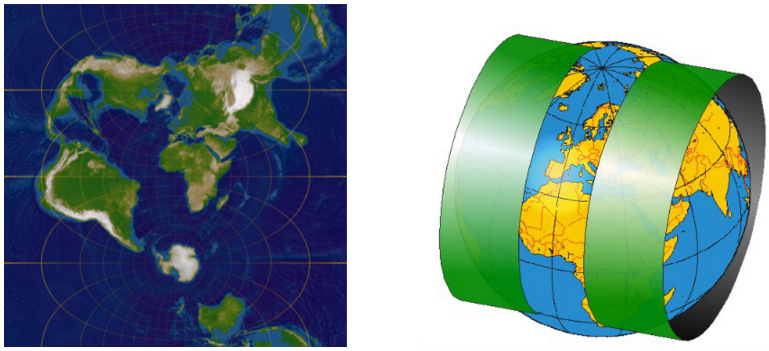

Abbildung 2: Transversale Mercator-Projektion für den Nullmeridian und Zylinderprojektion aus [10] und [35] (v.l.n.r.)

Abbildung 3: Europas UTM-Zonen aus [36]

Das *Gauß-Krüger-System* [24] ist der UTM-Darstellung sehr ähnlich [27]. Zur Erstellung des Koordinatensystems verwendet man ebenso eine transversale Mercator-Projektion. Allerdings basieren beide Systeme auf unterschiedlichen Referenzellipsoiden.

5 Markerloses optisches Tracking

Das Global Positioning System eignet sich zwar für eine grobe Schätzung der Position, allerdings sind Ungenauigkeiten im zweistelligen Meterbereich deutlich zu hoch, bei der Verwendung herkömmlicher GPS-Geräte jedoch kaum vermeidbar. Des Weiteren kann die Orientierung eines realen Betrachters per GPS nicht bestimmt werden. Optisches Tracking bietet die Möglichkeit einer sehr präzisen Rekonstruktion der Kamerapose. Da das zu entwickelnde Verfahren auf urbane Umgebungen angewandt werden soll, ist eine Verwendung von Markern nicht geeignet.

5.1 Analyse durch Synthese

Die *Analyse durch Synthese* wurde erstmals 1961 von Bell et al. [2] im Kontext der Sprachanalyse beschrieben. Demnach verarbeitet das menschliche Gehirn Sprache durch eine Rekonstruktion der wahrgenommenen Laute. Beim Vergleich des empfangenen und selbst rekonstruierten Sprachsignals wird die Abweichung der Signale zueinander berechnet. Mit den Informationen aus dem Vergleich werden iterativ weitere Sprachsignale synthetisch generiert bis die Differenz minimal ist. Zur Erstellung eines Sprachsignals werden bestimmte Parameter vorausgesetzt, wobei die Kombination einzelner Parameter ein eindeutiges Sprachsignal definieren.

Achilles [32] beschreibt eine Anwendung der Analyse durch Synthese im Kontext des optischen Trackings. Dabei stellen die zwei zu vergleichenden Objekte jeweils eine reale und eine virtuelle Kamerapose dar. Die Parameter zur Erstellung einer virtuellen Kamerapose sind eine Translation und Rotation im Raum. Nachdem eine virtuelle Kamerapose erzeugt wurde, kann sie mit der realen Kamerapose verglichen werden. Der Vergleich selbst geschieht über die Bilder, welche jeweils von der realen und virtuellen Kamera aus ihrer Pose heraus erzeugt werden. Je größer das berechnete Maß der Übereinstimmung ist, desto besser wurde die reale Pose rekonstruiert. Um diesen Ansatz zu realisieren, müssen sowohl eine reale als auch eine virtuelle Kamera die Möglichkeit haben, gleiche oder ähnliche Bilder generieren zu können. Dazu muss die Realität möglichst exakt in einem virtuellen Modell abgebildet werden. Da dies aus technischer Sicht sehr aufwendig und in der Darstellung sehr rechenintensiv ist, bieten sich Merkmalsdetektoren an, die die jeweiligen Bilder auf ihre wesentlichen Eigenschaften reduzieren. Achilles [32] verwendet dazu sowohl Punkt- als auch Liniendetektoren.

5.2 Lineare und nichtlineare Verfahren

Zur Rekonstruktion einer realen Kamerapose gibt es eine Vielzahl konkreter Algorithmen, die sich grundsätzlich in zwei Kategorien aufteilen lassen:

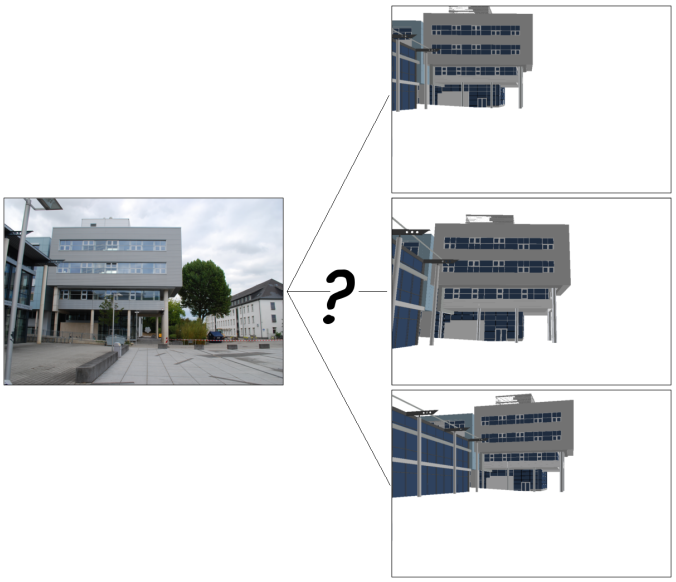

Abbildung 4: Rekonstruktion der Kamerapose mit *Analyse durch Synthese*

Lineare und *nichtlineare* Methoden. Achilles [32] fasst den Unterschied der beiden Ansätze zusammen.

Lineare Verfahren verwenden Merkmalskorrespondenzen zwischen Bildern, um die Kamerapose zu rekonstruieren. Diese Art von Verfahren benötigt keine initiale Poseschätzung, ist dafür aber sehr anfällig gegenüber Bildstörungen. Sobald Merkmalskorrespondenzen aus einer Bildreihe verloren gehen oder falsch zugeordnet werden, schlägt eine weitere Rekonstruktion der Kamerapose fehl. Für eine zuverlässige Rekonstruktion der Kamerapose ist die Detektion und Zuordnung immer gleicher Merkmale essentiell. Konkrete Realisierungen und Weiterentwicklungen solcher Verfahren sind unter anderem die *Direkte lineare Transformation* (DLT) [30] und der in OpenCV [6] implementierte *POSIT-Algorithmus* [4].

Nichtlineare Methoden versuchen das Problem der Empfindlichkeit gegenüber Bildstörungen und den daraus resultierenden Fehler einer Poserekonstruktion zu beheben. Alt [33] beschreibt den Grundsatz solcher Verfahren. Er erkennt, dass nichtlineare Methoden Ausreißer in Messreihen kompensieren können. Mit diesen Methoden sind allerdings nur lokale Minima zu berechnen. Da innerhalb einer Messreihe natürlich mehrere lokale Minima existieren können, muss der Suchbereich eingegrenzt werden. Angewandt auf den Kontext des Trackings bedeutet dies, dass die nichtlinearen Optimierungen zwar nicht so empfindlich gegenüber Bildstörungen sind, dafür aber eine initiale Poseschätzung benötigen.

Die Schätzung einer initialen Pose soll durch das im weiteren Verlauf zu entwickelnde System automatisch berechnet werden. Ist die erste Poseschätzung ausreichend präzise, können nichtlineare Verfahren die weitere

Rekonstruktion der Kamerapose übernehmen, auch wenn die Kamera Veränderung in Bezug auf ihre Position und Orientierung erfährt.

6 Algorithmen der Bildverarbeitung

Ein Bildvergleich zwischen unbearbeiteten Bildern der realen Szene und des virtuellen Modells ist problematisch, da ein geeignetes Bildabstandsmaß nur schwer definiert werden kann. Die Bildverarbeitung bietet eine Vielzahl von Algorithmen, um Merkmale aus Bildern zu extrahieren. Bilder sollen auf ihre für einen Bildvergleich relevanten Eigenschaften reduziert werden. Für die Implementierung eines Systems, dessen Anwendungsgebiet auf einen urbanen Kontext beschränkt ist, bietet sich eine *Analyse auf Kantenbildern* besonders an.

In [34] werden die grundlegenden Überlegungen zur Erstellung von Kantenbildern beschrieben. Angenommen die Grauwerte entlang einer Bildzeile werden durch eine Funktion $f(x)$ repräsentiert, dann enthält die erste Ableitung dieser Funktion $f'(x)$ die Steigung an jeder Stelle x. Positionen, an denen die erste Ableitung ein lokales Maximum aufweist, haben an jener Stelle eine sehr hohe positive Steigung. Analog dazu verhält es sich mit lokalen Minima.

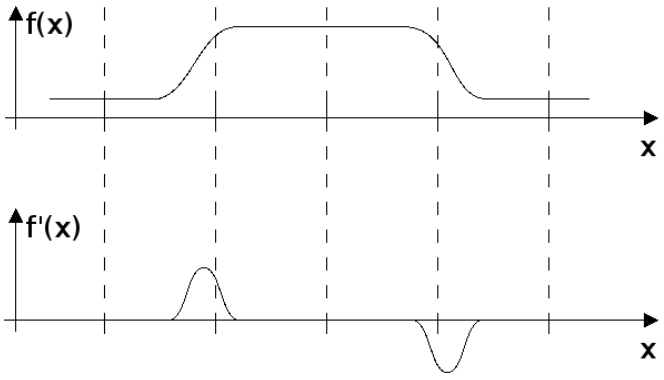

Abbildung 5: Steigung einer Funktion anhand der ersten Ableitung nach [34]

Die Steigung an jeder Stelle x lässt sich ebenso als Tangente durch jeden Punkt x beschreiben. Die Tangente einer diskreten Funktion an einer Position x kann näherungsweise als Parallele der Verbindungsgeraden durch die Funktionswerte $x-1$ und $x+1$ bestimmt werden.

$$\frac{df}{dx}(x) \approx \frac{f(x+1) - f(x-1)}{2} \tag{1}$$

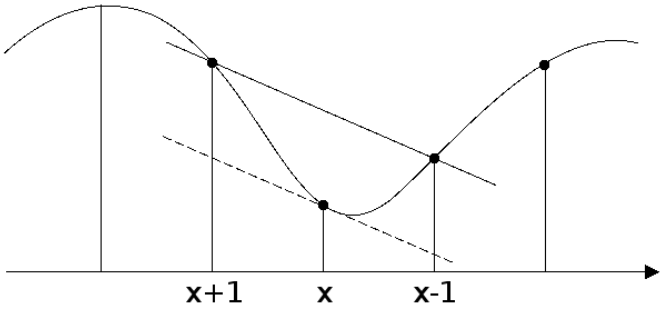

Abbildung 6: Approximation einer Tangente bei diskreten Funktionen nach [34]

Da Bilder im Normalfall natürlich zweidimensional sind, muss die erste Ableitung jeweils auch in zwei Richtungen berechnet werden. Für ein Bild, das durch eine Funktion $I(x, y)$ beschrieben wird, ergeben sich daraus zwei *partielle* Ableitungen.

$$\frac{\partial I}{\partial x}(x, y) \tag{2}$$

Analog zur partiellen Ableitung in x-Richtung ist die Ableitung in y-Richtung definiert.

$$\frac{\partial I}{\partial y}(x, y) \tag{3}$$

Sind die beiden partiellen Ableitungen berechnet, so kann der *Gradient* der Funktion I für jede Position (x, y) berechnet werden.

$$\nabla I(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) \\ \frac{\partial I}{\partial y}(x, y) \end{bmatrix} \tag{4}$$

Der Betrag $|\nabla I|$ des Gradienten ist unabhängig von der Orientierung der Bildstrukturen. Kanten innerhalb eines Bildes können dementsprechend richtungsunabhängig lokalisiert werden.

$$|\nabla I| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{5}$$

In [34] wird weiterhin beschrieben, wie die Approximation der ersten Ableitung als *linearer Filter* zur Anwendung auf ein Bild formuliert werden kann. Man erhält eine Koeffizientenmatrix H_x^D für eine Ableitung in x-Richtung und analog dazu H_y^D.

$$H_x^D = \begin{bmatrix} -0.5 & \underline{0} & +0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & \underline{0} & +1 \end{bmatrix} \tag{6}$$

$$H_y^D = \begin{bmatrix} -0.5 \\ 0 \\ +0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ +1 \end{bmatrix} \tag{7}$$

Für den Fall der partiellen Ableitung in x-Richtung und eines konkreten Bildelements $I(x_0, y_0)$ bezieht sich der Koeffizient -0.5 auf das Bildelement $I(x_0 - 1, y_0)$. Der Koeffizient $+0.5$ betrifft folglich Bildelement $I(x_0 + 1, y_0)$. Analog dazu ist die Ableitung in y-Richtung definiert.

Die Reaktion der beiden *Gradientenfilter* ist sehr unterschiedlich. Während H_x^D besonders stark auf Werteänderungen in horizontaler Richtung reagiert, so ist H_y^D auf die Detektion von Werteänderungen in vertikaler Richtung ausgelegt. Durch die Kombination beider Filter können nun beliebige Kanten innerhalb eines Eingabebildes gefunden werden. Diese Überlegungen stellen die Grundlage des *Sobel-Operators* und anderen hier nicht näher betrachteten Kantendetektoren, wie dem *Prewitt-Operator* beispielsweise, dar.

6.1 Sobel-Operator

Ähnlich den vorangegangenen Überlegungen wird der *Sobel-Operator* nach [34] separat in die horizontale und vertikale Richtung angewandt. Im Unterschied zu den zwei bisher gezeigten Filterkernen werden allerdings nicht nur die zwei direkten Nachbarn berücksichtigt, sondern immer mindestens sechs direkte Nachbarn. Daraus ergibt sich für einen Sobel-Filterkern eine Matrixdimension von 3x3. Es können auch noch weiter außen liegende Nachbarpixel einbezogen werden. Die Größe des Filterkerns kann dann einfach auf 5x5 oder 7x7 erhöht werden. Entscheidend dabei ist, dass die Matrix quadratisch bleibt und die Dimensionen ungerade Werte besitzen. Die Anwendung eines Filterkerns auf jedes Pixel eines Bildes wird auch als *Faltung* bezeichnet. Die 3x3-Filterkerne des Sobel-Operators werden nun wie folgt definiert.

$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ zur Kantendetektion in x-Richtung} \tag{8}$$

$$H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \text{ zur Kantendetektion in y-Richtung} \tag{9}$$

Die Anwendung des Sobel-Operators hat automatisch eine *Glättung* des Bildes zur Folge, was zur Verringerung von Bildstörungen[5] durch künst-

[5] insbesondere Rauschen

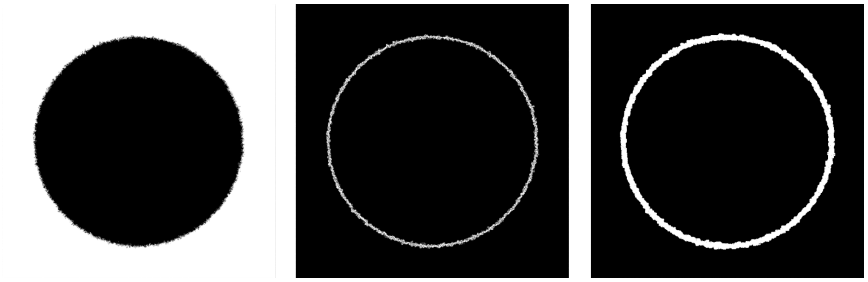

Abbildung 7: verrauschte Kantenübergänge eines Kreises, Kantenstärkebild nach der in OpenCV implementierten Sobel-Filterung mit einem 3x3- bzw. 7x7-Filterkern (v.l.n.r.)

lich erzeugte Unschärfe beiträgt. Je größer die Dimension der Filterkerne gewählt wird, desto stärker wird das Bild geglättet und mehr Pixel werden bei der Kantendetektion berücksichtigt. Die Glättung eines Bildes kann sowohl vor als auch nach der Kantendetektion geschehen, da Faltungen kommutativ sind. In Abbildung 7 erkennt man die unterschiedlichen Ergebnisse bei verschiedenen Filtergrößen.

Für die vorgestellten 3x3-Filterkerne erfolgt die Schätzung des Gradienten für jedes Bildelement nach folgender Formel.

$$\nabla I(x,y) \approx \frac{1}{8} \begin{bmatrix} H_x^S \cdot I \\ H_y^S \cdot I \end{bmatrix} \tag{10}$$

Durch Anwendung des Sobel-Operators lassen sich nach [34] jedoch nicht nur die *Kantenstärken* berechnen, sondern gleichzeitig auch die *Kantenrichtungen*. Seien $S_x(x,y)$ und $S_y(x,y)$ die Ergebnisse der Anwendung des jeweiligen Filterkerns auf eine Bildfunktion $I(x,y)$.

$$S_x(x,y) = H_x^S \cdot I \tag{11}$$

$$S_y(x,y) = H_y^S \cdot I \tag{12}$$

Die Kantenstärke $E(x,y)$ jedes Bildelements lässt sich durch den Betrag des Gradienten bestimmen.

$$E(x,y) = \sqrt{(S_x(x,y))^2 + (S_y(x,y))^2} \tag{13}$$

Die Kantenrichtung $\Phi(x,y)$ jedes Bildelements hingegen berechnet sich wie folgt.

$$\Phi(x,y) = \tan^{-1}\left(\frac{S_y(x,y)}{S_x(x,y)}\right) \tag{14}$$

Abbildung 8 stellt den Ablauf einer Kantendetektion mit dem Sobel-Operator noch einmal exemplarisch dar.

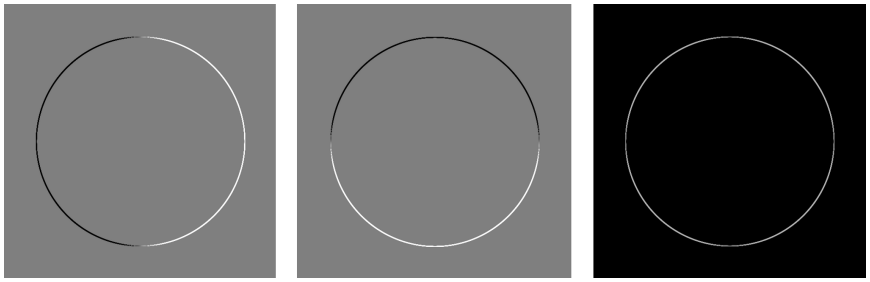

Abbildung 8: Sobel-Faltung in x-Richtung, Sobel-Faltung in y-Richtung, Kantenstärkebild nach einer Sobel-Kantendetektion (v.l.n.r.)

6.2 Canny-Operator

Der *Canny-Operator* ist ein weiteres Verfahren der Bildverarbeitung zur Detektion von Kanten. Zusätzlich zu Sobel haben die gefundenen Kanten in den Ergebnisbildern immer nur eine Breite von exakt einem Pixel. [15] beschreibt die Problematik bei Kantendetektoren als ständigen Kompromiss zwischen einer guten Reduktion von Bildstörungen oder besserer Lokalisierung von Kanten. Der Canny-Operator soll dabei einen geeigneten Mittelweg finden.

[15] fasst die Anwendung des Canny-Operators weiter zusammen. In einem ersten Schritt werden Bildstörungen durch eine Glättung des Bildes mittels *Gauß-Filterung*[6] eliminiert. Dabei ist durch den Benutzer eine *Standardabweichung* σ festzulegen, welche die Stärke der Glättung definiert. Je größer σ gewählt wird, desto stärker wird das Bild geglättet. Im Gegenzug können jedoch schwache Kantenzüge bei der Wahl eines zu großen σ verloren gehen.

Im Anschluss werden die Kanten des geglätteten Bildes hervorgehoben. Dieser Schritt zerlegt sich in zwei Teilaufgaben. Zu Beginn werden die Gradienten eines jeden Pixels, beispielsweise durch Anwendung des Sobel-Operators, berechnet. Die Kantenrichtung $\Phi(x, y)$ eines jeden Pixels wird auf die Werte 0°, 45°, 90° und 135° gerundet. Diese Winkel entsprechen vier möglichen Kantenrichtungen, in die eine Kante durch die rasterförmige Anordnung der Pixel verlaufen kann. Aus den vier Kantenrichtungen ergeben sich für ein konkretes Pixel $I(x_0, y_0)$ acht benachbarte Pixel, die mit $I(x_0, y_0)$ nun in Bezug auf ihre Kantenstärke verglichen werden. Nur wenn die Kantenstärke $E(x_0, y_0)$ eines Pixels $I(x_0, y_0)$ größer ist als bei allen acht Nachbarpixeln, wird $I(x_0, y_0)$ nicht auf den Wert 0 gesetzt. In Kantenrichtung eines konkreten Pixels $I(x_0, y_0)$ liegende Pixel müssen bei diesem Vergleich der Kantenstärken ignoriert werden. Nach der Detektion lokaler Maxima mit dem besagten Verfahren[7] sind alle Kantenzüge auf eine

[6] Anwendung einer Gauß'schen Glockenkurve als Glättungsfilter [15]

[7] *Non-Maximum Suppression*

Breite von exakt einem Pixel reduziert. In einem nächsten Schritt wird ein *Hysterese-Schwellwertverfahren* angewandt. Dabei werden durch den Benutzer manuell zwei Schwellwerte C_L und C_H definiert. Ziel des Verfahrens ist es, die für eine Kante nicht relevanten Pixel zu eliminieren. Wird ein konkretes Pixel $I(x_0, y_0)$, dessen Kantenstärke höher als C_H ist, im Kantenbild gefunden, so wird die Kante in beide Richtungen weiterverfolgt. Ein Pixel bleibt Teil der Kante, sofern die Kantenstärke größer als C_L ist.

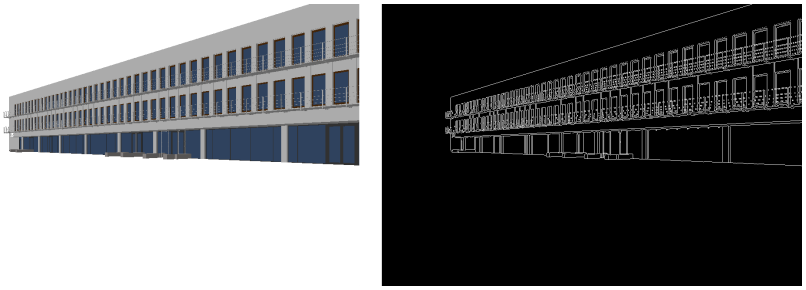

Abbildung 9: Eingabe- und Ausgabebild nach Anwendung des Canny-Algorithmus mit OpenCV ($C_L = 30$, $C_H = 50$) (v.l.n.r.)

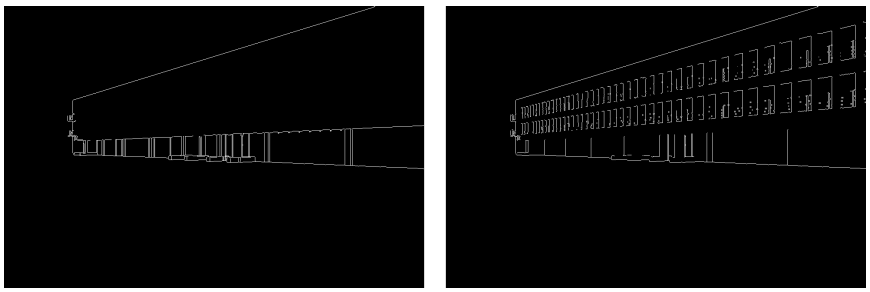

Abbildung 10: Beispiele für Canny-Bilder mit verschiedenen Schwellwerten ($C_L = 30$, $C_H = 800$) und ($C_L = 300$, $C_H = 500$) (v.l.n.r.)

Abbildung 9 zeigt dabei exemplarisch das Ergebnis einer Kantendetektion durch Verwendung des Canny-Operators. In Abbildung 10 wird deutlich, dass die beiden Schwellwerte C_L und C_H stets mit Bedacht gewählt werden sollten.

6.3 Hough-Transformation

Die *Hough-Transformation*[8] ist ein Detektor für einfache geometrische Formen wie Geraden oder Ellipsen. In [34] wird die Funktionsweise einer HT beschrieben. Entscheidend dabei ist, dass die gesuchte Form *parametrisierbar* sein muss. Da sich das Auffinden von Geraden in einem urbanen Kontext besonders anbietet, soll dieser Fall nun genauer betrachtet werden.

[8] im Folgenden als HT bezeichnet

Eine Gerade lässt sich in der zweidimensionalen Ebene in ihrer Parameterform folgendermaßen beschreiben.

$$y = f(x) = s \cdot x + t \tag{15}$$

Die beiden Parameter s und t bestimmen die Lage und Ausrichtung der Geraden in der Ebene. Parameter s gibt die Steigung der Geraden an und t steht für den Schnittpunkt der Geraden mit der y-Achse. Seien beispielsweise zwei beliebige Punkte a und b gegeben, wobei sowohl a als auch b auf einer gemeinsamen Geraden liegen. Um die Parameter der besagten Geraden durch a und b zu bestimmen, müssen für die Parameter s und t die folgenden zwei Gleichungen gültig sein.

$$y_a = f(x_a) = s \cdot x_a + t \tag{16}$$

$$y_b = f(x_b) = s \cdot x_b + t \tag{17}$$

Die Suche nach den passenden Werten für s und t bilden die Grundlage einer HT. Sucht man alle Geraden, die durch einen konkreten Punkt $p_0 = (x_{p_0}, y_{p_0})$ verlaufen, so lässt sich jede Gerade L_j beschreiben durch.

$$L_j : f(x_{p_0}) = s_j \cdot x_{p_0} + t_j \tag{18}$$

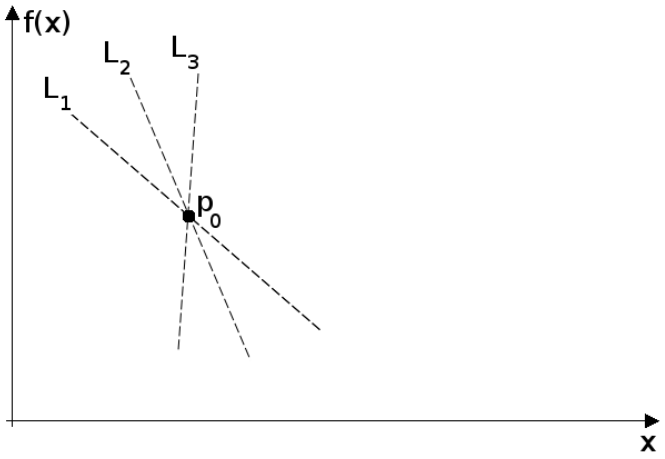

Abbildung 11: Geradenbüschel durch einen Punkt p_0 nach [34]

Da jede Gerade L_j durch p_0 die obige Gleichung erfüllen muss, lässt sich p_0 auch als die Menge aller entsprechenden Parameter (s_j, t_j) jeder Geraden L_j beschreiben. Für einen konkreten Parameter s_j lässt sich der zweite Parameter t_j wie folgt berechnen.

$$t_j = -x_{p_0} \cdot s_j + f(x_{p_0}) \tag{19}$$

Ein beliebiger Punkt q kann anhand der vorangegangenen Überlegungen in zwei verschiedenen Darstellungsformen beschrieben werden. Im sogenannten *Bildraum* wird ein Punkt q durch seine Koordinaten (x_q, y_q) dargestellt. Ein Punkt q kann weiterhin mit allen Geraden L_i beschrieben werden, die durch den Punkt q verlaufen, wobei jede Gerade durch ihre Parameter s_i und t_i, die den sogenannten *Parameterraum* oder auch *Hough-Raum* aufspannen, definiert ist. Ein Punkt im Bildraum ist also als Gerade im Parameterraum zu verstehen. Analog dazu werden Geraden im Bildraum als Punkte im Parameterraum beschrieben. Abbildung 12 verdeutlicht den Zusammenhang zwischen den Darstellungsformen. Die Punkte p_1 und p_2 im Bildraum werden im Parameterraum durch die Geraden M_1 und M_2 dargestellt. Die Gerade L_{12} im Bildraum wird zum Punkt q_{12} im Parameterraum.

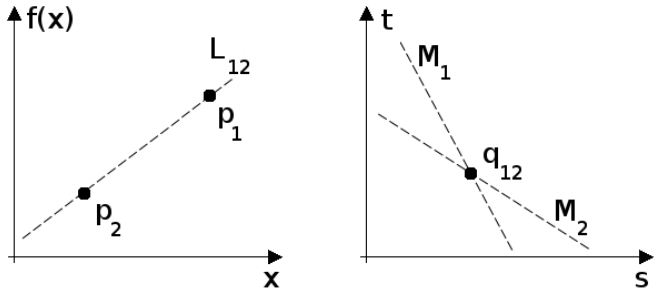

Abbildung 12: Repräsentation von Punkten und Geraden im Bildraum und Parameterraum (v.l.n.r.) nach [34]

Aus dem Zusammenhang zwischen Bild- und Parameterraum lässt sich schließen, dass wenn Z Geraden an einer konkreten Position (x_{p_0}, y_{p_0}) im Parameterraum einen gemeinsamen Schnittpunkt besitzen, auch genau Z Punkte im Bildraum auf einer gemeinsamen Geraden liegen. Eine HT findet Geraden in einer Punktewolke, indem alle Punkte durch ihre Geraden im Parameterraum repräsentiert werden. In der Darstellung des Parameterraums können Schnittpunkte der Geraden gesucht werden. Die Punkte im Parameterraum, die von besonders vielen Geraden geschnitten werden, entsprechen den Geraden im Bildraum, die besonders viele gemeinsame Punkte haben und somit eine dominante Kante bilden.

Nach [34] sind Geradengleichungen mit Angabe der bisher verwendeten Parameter s und t in der Praxis unbrauchbar, da für Geraden, die parallel zur y-Achse verlaufen, die Bedingung $(s = \infty)$ gelten muss. Die Parametrisierung einer Geraden durch die *Hessesche Normalenform* hat sich als unproblematischer erwiesen. In ihrer Hesseschen Normalenform wird eine Gerade durch den Winkel θ und den Abstand zum Koordinatenursprung ρ beschrieben. Der Winkel θ wird zwischen der Normalen einer Geraden durch den Koordinatenursprung und der x-Achse gemessen. Der Abstand zwischen dem Schnittpunkt der Normalen mit ihrer Geraden und dem Koordinatenursprung definieren den Abstand ρ. Abbildung 13 verdeutlicht die Darstellung einer Geraden durch ihre Hessesche Normalenform.

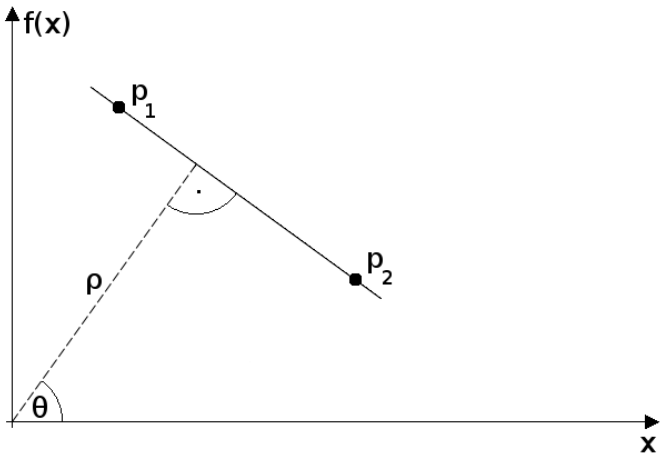

Abbildung 13: Hessesche Normalenform einer Geraden nach [34]

Für jeden beliebigen Punkt (x, y) auf einer konkreten Geraden gilt nach der Hesseschen Normalenform die folgende Gleichung.

$$\rho = x \cdot \cos(\theta) + y \cdot \sin(\theta) \tag{20}$$

Teil III
Planung und Implementierung

7 Hardware

Bevor mit der Planung und Implementierung des Systems begonnen werden kann, müssen zuerst einige Überlegungen über die benötigte Hardware erörtert werden. Das in dieser Arbeit realisierte Verfahren kombiniert zwei verschiedene Ansätze: markerloses optisches Tracking und Tracking via GPS. Die grobe Poseschätzung soll durch eine GPS-Ortung berechnet werden. Die Verfeinerung dieser Schätzung wird durch Vergleiche zwischen einem realen und einer Reihe von synthetisch erzeugten Bildern ermöglicht. Zur Realisierung und Evaluation des Systems werden also die folgenden Geräte benötigt.

Hardware	Verwendung
GPS-Empfänger	Messung der Position zur Laufzeit
Kompass	Messung der Blickrichtung zur Laufzeit
Kamera	Erstellung der Fotos zum Bildvergleich

Da für eine initiale Poseschätzung lediglich ein Einzelbild benötigt wird, arbeitet das System ausschließlich auf Fotos und nicht auf Videoaufnahmen. Theoretisch könnte man als Eingabe ein Videosignal, von welchem zu festgelegten Zeiten ein Einzelbild entnommen wird, nutzen. Zum Testen des erarbeiteten Verfahrens ist dies aber nicht zwingend erforderlich. Zur Erstellung der Testbilder wurde auf eine Spiegelreflexkamera Nikon D60 zurückgegriffen.

Um im späteren Verlauf wie geplant reale und synthetische Bilder miteinander vergleichen zu können, müssen die Einstellungen der realen Kamera mit denen der virtuellen Kamera übereinstimmen[9]. Der wichtigste Parameter ist dabei die verwendete Brennweite zum Zeitpunkt der Aufnahme. Ein Bildvergleich zwischen Kameras mit unterschiedlicher Brennweite ist nicht möglich, weil die Bilder trotz gleicher Position und Orientierung der Kameras dennoch völlig unterschiedlich wären. Da die eingestellte Brennweite der verwendeten Kamera einfach abgelesen werden konnte, war es problemlos möglich, diesen Parameter der virtuellen Kamera zu übergeben. Die restlichen Parameter, wie beispielsweise die radiale Verzerrung, konnten für dieses Anwendungsszenario vernachlässigt werden.

Zur Bestimmung der Position zum Zeitpunkt der Bildaufnahme diente das von der AG Computergrafik ausgeliehene Smartphone HTC Desire. Moderne Smartphones stellen gewöhnlich standardmäßig GPS-Navigation zur Verfügung. Zum Auslesen der gemessenen Positionsdaten wurde die

[9] *intrinsische Kamerakalibrierung*

im Android-Market kostenlos erhältliche Software GPSTest [3] gewählt, da eine Ausgabe der Positionsdaten in vielen gängigen Formaten unterstützt wird.

Für eine Messung der Blickrichtung zum Aufnahmezeitpunkt stellte sich der im HTC Desire integrierte Kompass als nicht sonderlich präzise heraus. Aus diesem Grund wurde ein handelsüblicher analoger Kompass verwendet.

8 Software

Da ein grober Ansatz des zu implementierenden Verfahrens festgelegt ist, müssen einige Entscheidungen für die zu verwendende Software getroffen werden. Das System sollte intuitiv über eine *GUI*[10] bedienbar sein. Für die Implementierung bieten sich die folgenden Bibliotheken an.

Software	Verwendung
Qt	Erstellung der grafischen Benutzeroberfläche
OpenCV	Umsetzung einiger Bildverarbeitungsalgorithmen
OGRE	Bereitstellung einer 3D-Engine

Nokias Qt-SDK [17] bietet eine Reihe von Tools zur Konzeption und Realisierung grafischer Benutzeroberflächen. In dieser Arbeit wurde Qt in der Version 4.7.3 verwendet. Die Erstellung der GUI mit einem strukturierten Layout wurde mit dem Qt-Designer durchgeführt. In einem nächsten Schritt mussten lediglich die Funktionen der einzelnen Bedienelemente implementiert werden. Eine Einarbeitung in die Konzepte von Qt erfolgte anhand der guten durch den Hersteller auf seiner Website [17] bereitgestellten Dokumentation.

Zum Rendern der Modelle dient die OGRE3D-Engine[11] [18]. Das Konzept dieser Grafik-Engine erlaubt es, mit verhältnismäßig geringem Aufwand das Rendern der Modelle vorzubereiten und durchzuführen. Eine Vielzahl von durch die OGRE-Community bereitgestellten Tutorials und [7] trug zu einem raschen Verständnis der in OGRE verwendeten Konzepte bei. Für die Implementierung dieser Arbeit wurde OGRE in der Version 1.7.3 verwendet.

Da sowohl OGRE als auch Qt die Programmiersprache C++ unterstützen, wurde zur Umsetzung der Bildverarbeitungsalgorithmen auf OpenCV [6] zurückgegriffen. Die gute Dokumentation der in dieser Arbeit verwendeten Version 2.3 sollte an dieser Stelle nicht unerwähnt bleiben. Einen guten Einstieg in OpenCV bieten [5] und [31].

[10] *Graphical User Interface*

[11] Object-Oriented Graphics Rendering Engine

Für die Bearbeitung möglicher Modelle wurde die Modellierungssoftware Blender [1] in der Version 2.49b gewählt, da für diese Version bereits Plugins zum Exportieren in das von OGRE benötigte Datenformat[12] vorhanden waren.

9 Systemarchitektur

Der folgende Abschnitt soll den Aufbau und einen groben Einblick in die Funktionsweise des Systems wiedergeben. Auf den detaillierten Programmablauf wird an dieser Stelle noch nicht eingegangen. Das implementierte System wird im weiteren Verlauf der Arbeit als *PoseInitializer* bezeichnet.

Abbildung 14 zeigt ein vereinfachtes UML-Klassendiagramm des PoseInitializers. Das erste zentrale Element ist die QOgreGLWidget-Klasse. In dieser Klasse werden alle Aufgaben erledigt, die im Kontext des Renderns stehen. Ein zentrales Konzept von OGRE ist die Verwendung des sogenannten *Frame Listeners*, wodurch es dem Programmierer ermöglicht wird, Einzelaufgaben automatisch vor bzw. nach dem Rendern eines Einzelbildes ausführen zu lassen. Dieses Konzept wurde stark in die Planung des Programmablaufs eingebunden.

Listing 1: Grober Ablauf der Render-Methode

```
1  void QOgreGLWidget::paintGL()
2  {
3          // do something before rendering one frame
4          ...
5
6          // start the rendering
7          ogreRoot->_fireFrameStarted();
8          ogreRenderWindow->update(true);
9          ...
10
11         // rendering of one frame is finished
12         ogreRoot->_fireFrameEnded();
13
14         // do something after rendering one frame
15         ...
16 }
```

[12] *.mesh, *.material

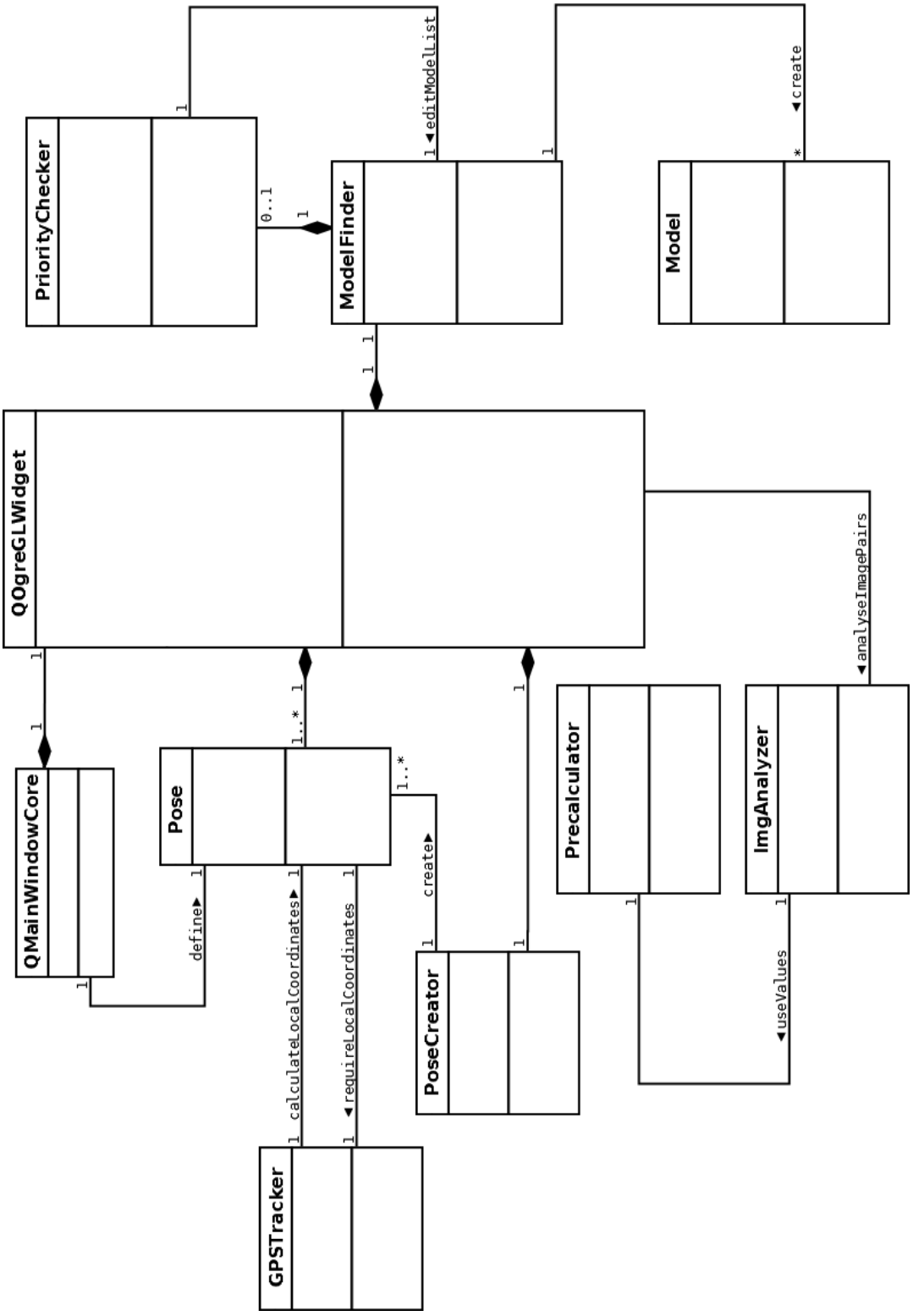

Abbildung 14: UML-Klassendiagramm des PoseInitializers

Die Aufgaben eines QOgreGLWidget-Objekts sind im Groben folgendermaßen festgelegt.

- Definition und Konfiguration des Rendering-Systems (inkl. Optionen wie AA[13], V-Sync[14], etc.)
- Definition des Szenegraphen
- Definition der künstlichen Lichtquellen
- Laden aller Ressourcen (Modelle, Materialien, Texturen)
- Definition und Konfiguration der Kamera (inkl. Änderung der Position und Orientierung zur Laufzeit)
- Laden des Bildschirmhintergrundes (Foto der realen Szene)
- dynamisches Erstellen und Löschen weiterer benötigter Instanzen anderer Klassen
- Aufrufen aller zur Laufzeit nötigen Methoden unter Berücksichtigung der Rendergeschwindigkeit
- dynamisches Laden und Löschen der Modelle innerhalb des Szenegraphen
- Erstellen von Screenshots der virtuellen Szene (ohne den Hintergrund)
- Erfassung und Ausgabe von Statistiken (Rendergeschwindigkeit, etc.)

Das zweite essentielle Element ist die QMainWindowCore-Klasse. Innerhalb dieser Klasse wird die Funktionalität der grafischen Benutzeroberfläche mittels Qt definiert. Alle Eingaben, die durch einen Benutzer gemacht werden können, werden innerhalb des QMainWindowCore-Objekts zentral verwaltet und an das QOgreGLWidget-Objekt übergeben. Des Weiteren wird über die Benutzereingabe ein Pose-Objekt definiert, welches die Position und Orientierung der virtuellen Kamera festlegt. Der Benutzer kann dabei zwei Eingaben für die aktuelle GPS-Position und eine Eingabe für seine Blickrichtung machen.

Die Pose-Klasse bietet Methoden und Attribute sowohl zur Erstellung einer Pose anhand einer GPS-Position und Kompassrichtung als auch zur Erstellung synthetischer Posen, die zusätzliche Kameraparameter, wie Kippung und Neigung, unterstützen. Aus den Eingaben des Benutzers müssen die Position und Blickrichtung in das von OGRE verwendete Koordinatensystem transformiert werden. Diese Aufgabe wird einem für diesen Zweck zur Laufzeit erstellten GPSTracker-Objekts zuteil.

[13] Anti-Aliasing
[14] vertikale Synchronisation

Ein PoseCreator-Objekt wird erstellt, sobald eine neue Poseschätzung stattfinden soll. Anhand der durch den PoseCreator erstellten Pose-Objekte kann eine Reihe von synthetischen Bildern erzeugt werden, die für den späteren Bildvergleich notwendig sind.

Der ImgAnalyzer implementiert alle Funktionen, die zum Vorverarbeiten und Vergleichen von Bildern notwendig sind. Zwar werden die zu vergleichenden Bilder immer unmittelbar nach dem Rendern eines Einzelbildes im QOgreGLWidget erstellt, vorverarbeitet und analysiert werden sie allerdings durch die im ImgAnalyzer implementierten Methoden. Manche Berechnungen der Bildvorverarbeitung und des Bildvergleichs können bereits beim Programmstart durchgeführt werden, sodass eine signifikante Steigerung der Performanz zu erwarten ist. Diese Ergebnisse werden innerhalb eines Precalculator-Objekts gespeichert und werden zur Laufzeit vom ImgAnalyzer ausgelesen.

Je nach Position und Ausrichtung der virtuellen Kamera müssen entsprechende 3D-Modelle dynamisch geladen werden. Diese Aufgabe wird innerhalb eines ModelFinder-Objekts bearbeitet, welches ebenfalls bei Bedarf zur Laufzeit erstellt wird. Je nachdem, ob immer alle Modelle oder nur eine bestimmte Teilmenge der Modelle geladen werden soll, wird ein PriorityChecker-Objekt erzeugt, welches nur die zur Laufzeit interessanten Modelle herausfiltert.

Jedes Modell wird in der Instanz einer Model-Klasse definiert, welche eine Reihe von Methoden zur Positionierung und Ausrichtung der Modelle und deren Eigenschaften bereithält.

10 Daten

10.1 Modell des Campus

Eine Evaluation des PoseInitializers setzt die Bereitstellung eines realen Testszenarios voraus. Zum einen müssen Fotos dieser realen Umgebung erstellt und gleichzeitig dazu korrespondierende GPS- und Kompassdaten erfasst werden. Des Weiteren muss die reale Szene in einem Modell abgebildet werden, damit ein Bildvergleich zwischen realen und synthetisch erstellten Bildern ermöglicht werden kann. In der Vergangenheit wurde bereits ein Modell des Koblenzer Campus erstellt [29]. Abbildung 15 zeigt einen Screenshot der selbstgesteuerten echtzeitfähigen Version des Campusmodells.

Für die Generierung einer Serie synthetischer Bilder des Campus Koblenz anhand des Modells ist der Detailgrad jedoch zu hoch. Teile, wie Büsche oder Bäume, des modellierten Campus eignen sich kaum für eine strukturierte Analyse, da ihr Erscheinungsbild in der realen Umgebung stark von der aktuellen Jahreszeit abhängig ist. Transparenzen der Glas-

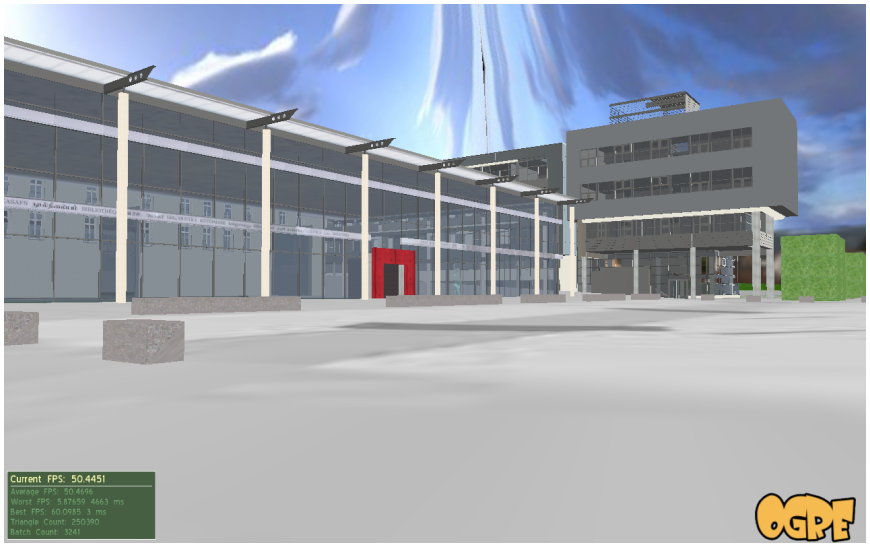

Abbildung 15: Screenshot aus der selbstgesteuerten Echtzeit-Version des Campusmodells von [29]

fronten und Fenster können den Bildvergleich ebenfalls negativ beeinflussen, da für einen virtuellen Betrachter eventuell Objekte sichtbar werden, die in der Realität selbst nicht sichtbar sind. In der Realität reflektieren Fenster normalerweise eher die Umgebung, anstatt das Innere eines Gebäudes zu zeigen. Da ein kantenbasierter Bildvergleich sinnvoll erscheint, wurden aus dem Modell alle grafischen Effekte, alle unnötigen Teile des Modells, wie beispielsweise Bäume und nahezu alle Texturen entfernt. Für eine spätere Detektion der Kanten sind lediglich das Polygonnetz und die Farbe der Materialien von entscheidender Wichtigkeit. Diese Reduzierung beseitigt nicht nur potentielle Fehlerquellen, sondern beschleunigt ein späteres Laden und Rendern des Modells deutlich. Die Durchführung besagter Reduzierung auf das Wesentliche des Modells wurde mit Blender [1] durchgeführt. Das Campusmodell ist in einem geeigneten Dateiformat auf der Website von [29] verfügbar. Die Bearbeitung des Modells offenbarte einige Unstetigkeiten. Beim Entfernen einiger nicht benötigter Objekte wurden teilweise auch für das Gebäude relevante Teile, wie Fenster oder Türen, gelöscht. Grund dafür ist ein fehlerhafter Aufbau des in Blender verwendeten Szenegraphen. Eine detaillierte Nachbearbeitung kam unter Berücksichtigung der begrenzten Zeit nicht infrage. Nur in seltenen Fällen wurde der Bildvergleich durch besagte Unstetigkeiten negativ beeinflusst.

10.2 Bestimmung und Umrechnung relevanter Daten

Position und Rotation der realen Gebäude und deren Modelle müssen in geeigneter Art und Weise zusammengetragen werden. Die Lage realer Ge-

bäude kann beispielsweise anhand der GPS-Koordinaten seiner Eckpunkte und der Ausrichtung entlang der weitesten Ausdehnung als Himmelsrichtung beschrieben werden. Bisher sind die folgenden Daten in zwei verschiedenen Koordinatensystemen vorhanden.

1. Reales Koordinatensystem
    - Position des Betrachters in GPS-Koordinaten (z.B. UTM)
    - Blickrichtung des Betrachters als Himmelsrichtung
2. Virtuelles Koordinatensystem
    - drei Parameter für die Position eines Gebäudes im dreidimensionalen kartesischen Koordinatensystem von Blender
    - drei Parameter für die Rotation des Gebäudes
    - drei Parameter für die Skalierung des Gebäudes

Zu Beginn soll die Erfassung der Modelldaten im virtuellen Koordinatensystem betrachtet werden. Wie im vorigen Abschnitt erläutert, steht eine Version des Campusmodells im nativen Blender-Format[15] zur Verfügung.

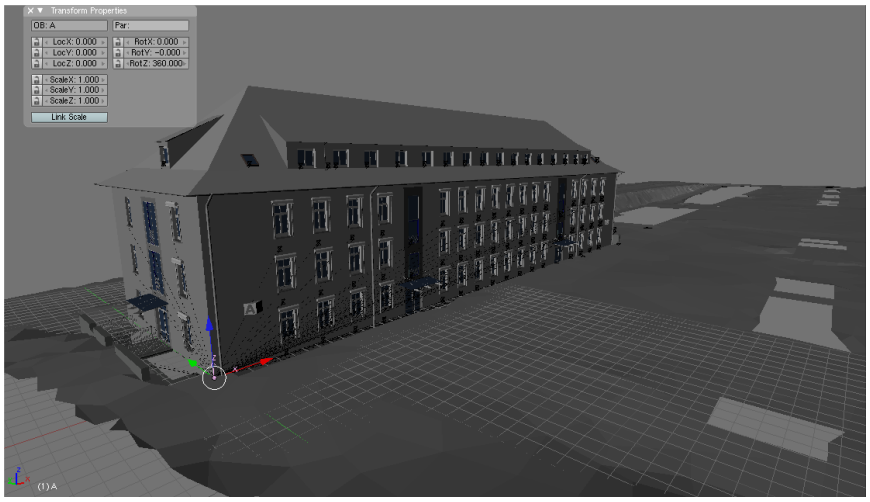

Abbildung 16: Screenshot des reduzierten Campusmodells in Blender (rot: x, grün: y, blau: z)

Abbildung 16 zeigt das A-Gebäude des Campus Koblenz. Man erkennt unter anderem, dass selbst das Terrain als separates Objekt modelliert wurde. Diese Tatsache ist für den späteren Verlauf dieser Arbeit noch von entscheidender Bedeutung. An den nicht modellierten Stellen des Terrains befanden sich für eine Analyse nicht relevante Objekte, die manuell entfernt

[15] *.blend

wurden. Das dreifarbige Koordinatenkreuz stellt das kartesische virtuelle lokale Koordinatensystem des A-Gebäudes dar. Alle Elemente des Gebäudes sind relativ zu diesem lokalen virtuellen Koordinatensystem transformiert[16]. Jedes Gebäude des Campus Koblenz ist als separate Entität im Szenegraphen von Blender vorhanden. Die Transformationsparameter des lokalen virtuellen Koordinatensystems einer Entität in Bezug auf das globale virtuelle Koordinatensystem des Campus lassen sich in Blender auslesen. Abbildung 17 zeigt die in Blender ausgegebenen Transformationsparameter des A-Gebäudes.

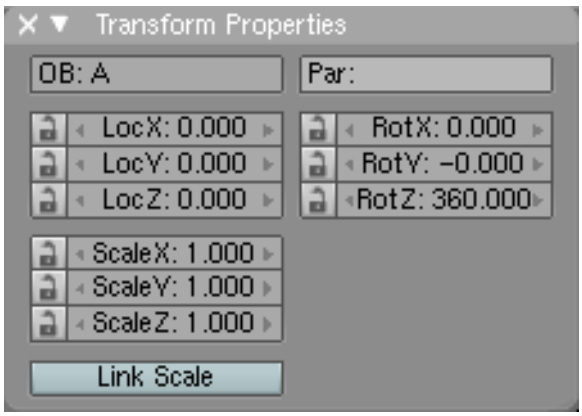

Abbildung 17: Transformationsparameter zwischen dem Koordinatensystem des Campus und dem Koordinatensystem des A-Gebäudes in Blender

Es fällt auf, dass das lokale virtuelle Koordinatensystem des A-Gebäudes genau mit dem globalen virtuellen Koordinatensystem des Campus übereinstimmt. Alle auf dem Campus vorhandenen Entitäten wurden so transformiert, dass eine manuell festgelegte Ecke des A-Gebäudes den Ursprung des globalen Campus-Koordinatensystems bildet. Die zueinander relative Platzierung der Gebäude wurde dadurch nicht beeinflusst. Die Ausrichtung des globalen virtuellen Koordinatensystems orientiert sich an einer Fassade des A-Gebäudes. Um virtuelle und reale Kameraposen miteinander vergleichen zu können, muss bekannt sein, wie die einzelnen Entitäten des Modells in Bezug auf die Realität positioniert und rotiert sind. Dementsprechend ist es sinnvoll, das virtuelle Koordinatensystem an einem Gebäude selbst auszurichten. Die Ausrichtung und Lage des virtuellen Koordinatensystems kann in diesem Fall einfach per GPS und Kompass bestimmt werden.

Liegt der Ursprung des virtuellen Koordinatensystems des Campus an einer bekannten Ecke eines bestimmten Gebäudes, so lässt sich die Translation vom realen ins virtuelle Koordinatensystem leicht berechnen. Eine Rotation zwischen den beiden Koordinatensystemen ist über die bekann-

[16] Transformation bestehend aus Translation, Rotation und Skalierung

te Ausrichtung des virtuellen Koordinatensystems ebenfalls möglich. Im Kontext des PoseInitializers wurde der Ursprung des virtuellen Koordinatensystems an der linken unteren Ecke der Front des A-Gebäudes definiert. Die virtuelle x-Achse verläuft entlang der Gebäudefront. Der Grundriss des Gebäudes kann als rechtwinklig angenommen werden. Die virtuelle z-Achse zeigt genau senkrecht nach oben. Wird nun die GPS-Koordinate der besagten Gebäudeecke gemessen und die Ausrichtung des Grundrisses als Himmelsrichtung beschrieben, so erhält man die nötigen Transformationsparameter zwischen GPS- und dem virtuellen Koordinatensystem. Das in Abbildung 16 gezeigte virtuelle Koordinatensystem entspricht jedoch nicht dem von OGRE verwendeten Koordinatensystem. Im Gegensatz zu Blender verwendet OGRE ein Koordinatensystem in einer xz-Ebene. Die y-Achse wird zur Hochachse. Sowohl Blender als auch OGRE setzen dabei auf ein rechtshändiges Koordinatensystem.

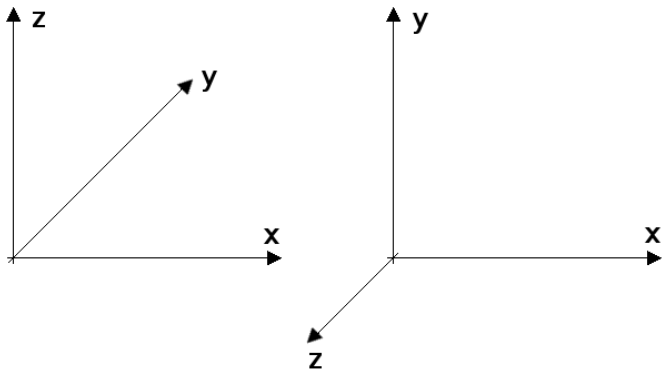

Abbildung 18: Koordinatensysteme in Blender und OGRE (v.l.n.r.)

Mit Blender lassen sich die Transformationsparameter für jedes einzelne Gebäude wie beschrieben auslesen. Damit ist zwar bekannt, wie ein virtuelles lokales Koordinatensystem einer jeden Entität im virtuellen globalen Koordinatensystem des Campus definiert ist, gesucht ist allerdings eine Transformation von GPS- und Kompassdaten in das virtuelle definierte Koordinatensystem des Campus. Bisher fehlen allerdings die GPS-Koordinaten und Ausrichtung des A-Gebäudes.

Das Landesamt für Vermessung und Geobasisinformation Rheinland-Pfalz bietet zur Bestimmung dieser Daten den online frei zugänglichen Service GeoPortal.rlp [12] an. Durch Verwendung des GeoPortals lassen sich auf einfachste Art und Weise GPS-Koordinaten in verschiedenen Formaten an der mit dem Maus-Cursor beschriebenen Position auslesen. Neben der Angabe der GPS-Position in Längen- und Breitengraden werden auch die

Formate Gauß-Krüger und UTM unterstützt. Weiterhin hat der Benutzer die Möglichkeit, sich entweder die Luftaufnahmen der betrachteten Region anzeigen zu lassen oder auch nur eine Katasterkarte mit den Grundrissen einzelner Gebäude.

Abbildungen 19 und 20 zeigen eine Katasterkarte des Campus Koblenz. Auffällig ist, dass Karte 19 verzerrt zu sein scheint. Der Grund dafür ist, dass eine Darstellung basierend auf Längen- und Breitengraden kein kartesisches Koordinatensystem bilden kann. Karte 20 hingegen erfüllt dieses Kriterium. Die rechten Winkel der Grundrisse bleiben erhalten.

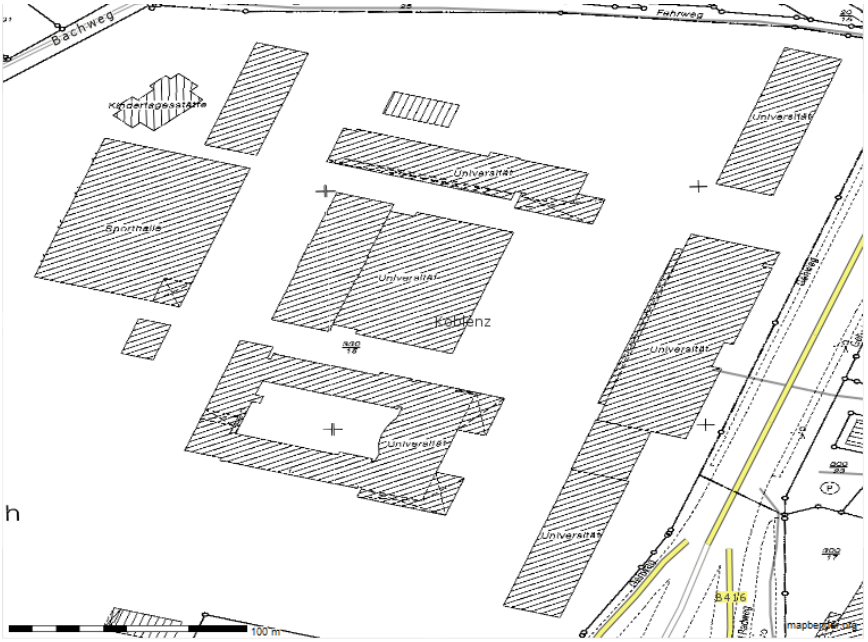

Abbildung 19: Katasterkarte des Campus Koblenz in geografischen Koordinaten (Längen- und Breitengrade) aus [13]

Im vorigen Verlauf dieser Arbeit wurde beschrieben, dass eine UTM-Projektion bereits ein kartesisches Koordinatensystem mit einer metrischen Skalierung der Werte definiert. Somit eignet sich diese Repräsentation der GPS-Positionen am besten. Für eine Umrechnung der UTM-Koordinaten in das von OGRE genutzte Koordinatensystem ist lediglich eine Translation und eine Rotation nötig. Skalierungen oder gar Scherungen müssen nicht berechnet werden.

Der Translationsparameter lässt sich unter Verwendung des GeoPortals durch ein Auslesen der GPS-Koordinaten der besagten Ecke des A-Gebäudes messen. Zur vollständigen Transformation zwischen dem UTM- und OGRE-Koordinatensystem fehlt bisher noch ein Rotationsparameter, dessen Bestimmung sich in der Praxis als problematisch herausstellt. Theoretisch wäre eine Vermessung der Ausrichtung des A-Gebäudes ausrei-

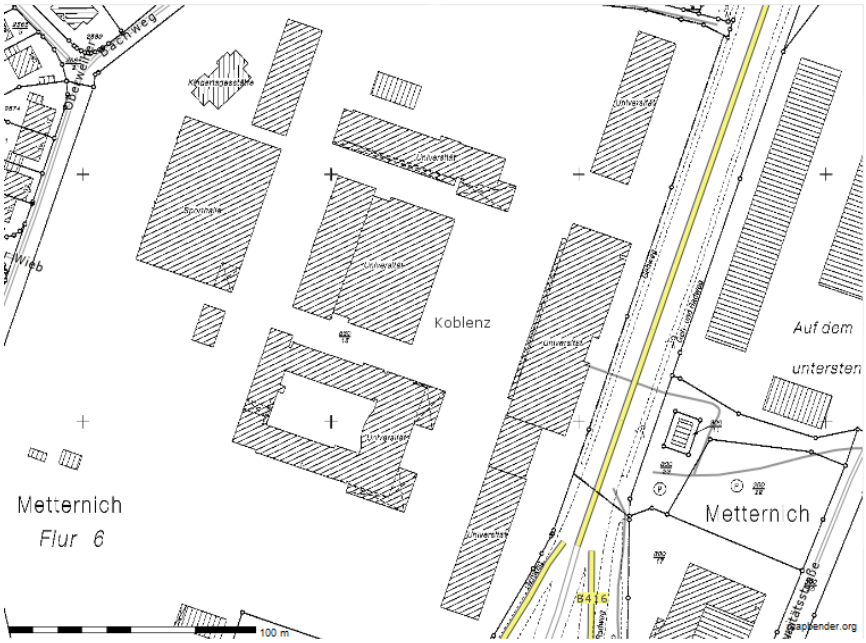

Abbildung 20: Katasterkarte des Campus Koblenz in UTM-Koordinaten aus [13]

chend. Die Platzierung der einzelnen Gebäude des Campusmodells in Relation zueinander weicht von der Realität allerdings teilweise um mehrere Meter ab. Die Berechnung des Rotationsparameters muss einen Kompromiss darstellen, sodass der Fehler für alle Gebäude minimal ist.

Der folgende Pseudo-Code zeigt eine Möglichkeit zur systematischen Bestimmung des optimalen Rotationsparameters.

Listing 2: Pseudo-Code zum automatischen Finden des idealen Rotationswinkels

```
1  // define min and max rotation
2  minRot = ...;
3  maxRot = ...;
4
5  // define the campus' origin in gps coordinates
6  // = the special corner of building A
7  campusorigin = getUTMCoordinates(building_A);
8
9  // get the gps coordinates of each models' origin
10 // (use the GeoPortal)
11 list origingps = getAllModelUTMOrigins();
12
13 // get the local transformations of each model
14 // (usage of Blender)
15 list originVirt = getAllModelBlenderOrigins();
16
17 for (all angles between minRot and maxRot)
18 {
19
20  for (int i = 0; i < number of models; i++)
21  {
22
23   // translate the origin of a model to the
24   // campus' origin
25   translateP = translateToCampusOrigin
26               (origingps[i],campusorigin);
27
28   //rotate the translated point
29   transformP = rotateToCampusSystem(translateP);
30
31   // check difference between calculated
32   // and by blender given point
33   diff = abs(originVirt[i] - transformP);
34   }
35
36  // search the max difference
37  searchMaxDiff();
38 }
39
40 // find the angle with the least differences
41 bestAngle = searchAngleWithLeastDiffs();
```

Ein grober Rotationsfaktor kann über die Ausrichtung des A-Gebäudes geschätzt werden. Getestet werden alle Rotationswinkel in bestimmten Abständen innerhalb eines definierten Bereiches um den geschätzten Rotationsfaktor. Die GPS-Koordinaten des Campusursprungs sind bekannt. Jedes Gebäude hat ein lokales Koordinatensystem, dessen Transformationsparameter aus Blender in Bezug auf das definierte virtuelle Koordinatensystem des Campus ebenfalls bekannt sind. Der Ursprung des lokalen Koordinatensystems eines jeden Gebäudes liegt ähnlich wie bei Gebäude A in einer konkreten Ecke, die in Blender bestimmt werden kann. Die zu den Ecken korrespondierenden GPS-Koordinaten können im Anschluss über das GeoPortal ausgelesen werden. Nun kann für jeden zu testenden Winkel eine Transformation der vermessenen Eckpunkte von GPS-Koordinaten in das von Blender verwendete Koordinatensystem durchgeführt werden. Im Idealfall erhält man für jede Ecke die in Blender ausgelesenen Translationsparameter. Der beste Rotationsfaktor zeichnet sich dadurch aus, dass für alle Gebäudeecken die Differenz zwischen der eben errechneten und der aus Blender vorgegebenen Translation minimal ist.

10.3 Verwaltung der Daten

Um die Gebäude des Campusmodells für die Verwendung mit OGRE vorzubereiten, müssen die Gebäude jeweils separat aus dem in Blender vorliegenden Modell exportiert werden. Dies ist nötig, damit in einem späteren Schritt die im vorigen Verlauf ausgelesenen Transformationsparameter jedem einzelnen Gebäude zugewiesen werden können. Zum Exportieren der Modelle aus Blender in ein für OGRE unterstütztes Dateiformat wurde ein entsprechendes Python-Skript [14] mit Blender verwendet.

Abbildung 21 zeigt die Optionen des Export-Skripts. Der wichtigste zu aktivierende Punkt ist dabei das Ausführen des OgreXMLConverters, der zum Ende des Exports die benötigten Dateien nach folgendem Muster erstellt.

- gebA.mesh
- gebA.material
- Texturen in üblichen Bildformaten (*.jpg, *.png, etc.)

Die Datei „gebA.mesh“ enthält lediglich das Polygonnetz des exportierten Gebäudes, wogegen in „gebA.material“ alle Materialeigenschaften und eventuelle Texturzuweisungen definiert sind. Um die Materialeigenschaften eines Modells nach dem Export manuell zu ändern, kann einfach die entsprechende Datei editiert werden.

Es werden nicht nur alle für den Bildvergleich relevanten Gebäude exportiert, sondern ebenfalls das Terrain selbst. Der Grund dafür findet sich

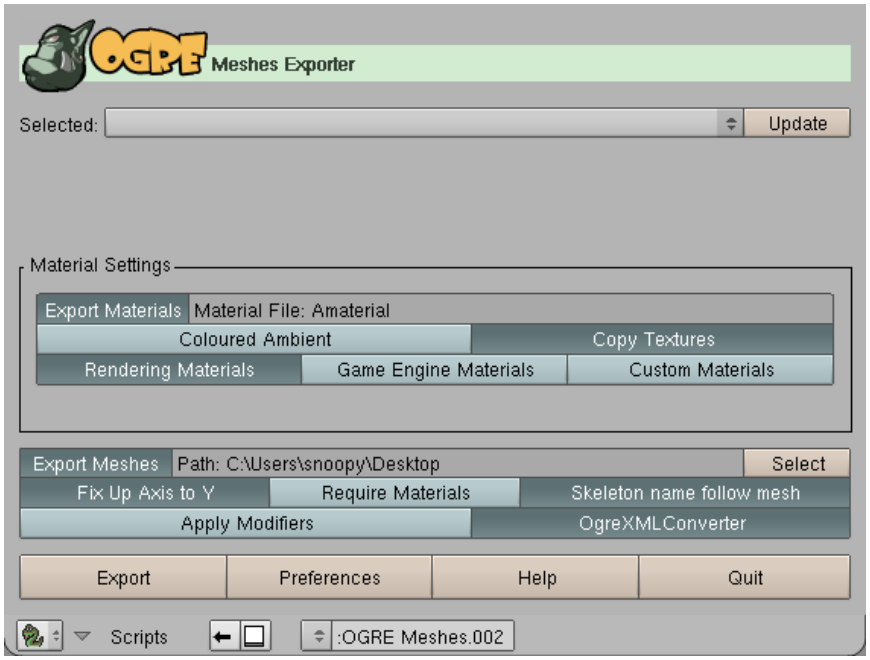

Abbildung 21: Blender Export-Skript für OGRE [14]

in den Modellen der einzelnen Gebäude. Die Modelle zeigen nicht nur die üblichen sichtbaren Teile des Gebäudes, sondern auch Teile des Fundaments, die für den Betrachter in der Realität üblicherweise nicht sichtbar sind. Um diese nicht sichtbaren Teile der Gebäude für den virtuellen Betrachter auszublenden, soll auch das Terrain selbst gerendert werden. Um nun zu erreichen, dass das Terrain zwar nicht sichtbare Gebäudeteile verdeckt, aber dennoch für den Betrachter unsichtbar bleibt, soll das Terrain ausschließlich in den Tiefenpuffer gerendert werden. Dies kann über die Modifikation der Material-Datei eines Modells erreicht werden. Zuerst können alle Farb- und Texturzuweisungen entfernt werden. Im Anschluss muss ein Schreiben in den Colour-Buffer deaktiviert werden. Der folgende Ausschnitt zeigt die veränderte Material-Datei des Terrains.

Listing 3: modifizierte Material-Datei des Terrains

```
material Terrain
{
 technique
 {
  pass
  {
   colour_write off
  }
 }
}
```

Das Terrain ist in diesem Fall das einzige Modell, dessen Material-Datei manuell verändert werden musste. Beim Starten des PoseInitializers soll das Terrain stets automatisch geladen werden. Durch die beschriebene Definition des Materials bleibt es für den Betrachter dennoch unsichtbar. Lediglich unter dem Terrain liegende Teile der Gebäude werden durch das unsichtbare Terrain verdeckt. Im weiteren Verlauf der Arbeit wird sich herausstellen, dass eine Verwendung des Terrains auch noch weitere Vorteile mit sich bringt.

Damit das Laden der Modelle und ihrer Materialen in OGRE überhaupt möglich wird, müssen die Pfade aller Dateien in OGRE bekannt sein. Dazu sind alle Dateien je nach Typ in separate Ordner abzulegen. Ein konkreter Ordner beinhaltet also beispielsweise nur Material-Dateien, ein anderer Ordner enthält alle Mesh-Dateien. Die Pfade zu den einzelnen Speicherplätzen müssen in der für OGRE relevanten Datei „ressource.cfg“ eingetragen werden, die beim Start eines OGRE-Systems automatisch geladen wird.

Listing 4: Beispiel einer „ressource.cfg“

```
[General]
FileSystem=media
Zip=media/packs/OgreCore.zip
FileSystem=media/materials/programs
FileSystem=media/materials/scripts
FileSystem=media/materials/textures
FileSystem=media/fonts
FileSystem=media/models

[BackgroundLocation]
FileSystem=media/background
```

Prinzipiell können nun die Gebäude in OGRE geladen werden, allerdings sind bisher keine Transformationsparameter mit den einzelnen Modellen verknüpft worden. Die relative Lage der Gebäude auf dem Campus findet also keine Berücksichtigung. Für ein automatisches Einlesen einer Datei, die Objekte mit bestimmten zusätzlichen Parametern verknüpft, eignet sich die XML-Syntax. Innerhalb des PoseInitializers bietet die Model-Klasse alle Methoden und Attribute, um ein Objekt in OGRE zu laden und zu bearbeiten. Für das spätere Laden der Objekte soll eine XML-Datei „campus.xml“ traversiert werden, in der alle nötigen Informationen erfasst sind. Diese Datei ist manuell zu erstellen und mit den nötigen Parametern zu füllen. Die Lage des lokalen Koordinatenursprungs und alle Transformationsparameter können aus Blender gelesen werden. Die GPS-Koordinaten des lokalen Ursprungs sind über das GeoPortal zu ermitteln.

Listing 5: Ausschnitt aus „campus.xml“

```
<campus>
 ...
 <model id ="2">
  <meshName>CBib.mesh</meshName>
  <originsGPSCoordX>397450.36</originsGPSCoordX>
  <originsGPSCoordY>5580054.24</originsGPSCoordY>
  <localTransX>154.620</localTransX>
  <localTransY>0.287</localTransY>
  <localTransZ>91.893</localTransZ>
  <localRotation>91.982</localRotation>
  <localScaleX>1.0</localScaleX>
  <localScaleY>1.0</localScaleY>
  <localScaleZ>1.0</localScaleZ>
  <controlPoints>
   <cpoint>
    <cpX>397468.9</cpX>
    <cpY>5580107.2</cpY>
   </cpoint>
   <cpoint>
    <cpX>397464.2</cpX>
    <cpY>5580093.9</cpY>
   </cpoint>
   ...
  </controlPoints>
 </model>
 ...
</campus>
```

Jedes Gebäude, welches über seine ID angesprochen werden kann, wird mit den folgenden Informationen verknüpft.

- Name der Mesh-Datei
- GPS-Koordinaten des Modell-Ursprungs
- Transformationsparameter
- GPS-Kontrollpunkte, die zum späteren Laden der Gebäude nötig sind

Eine genauere Erläuterung über den Nutzen der zu setzenden Kontrollpunkte erfolgt im weiteren Verlauf dieser Arbeit. Essentiell für eine korrekte Anwendung der Transformation ist, dass die Transformationsparameter durch die Verschiedenartigkeit der Koordinatensysteme von OGRE und Blender nicht einfach in die XML-Datei übertragen werden können. Damit die Transformationswerte aus Blender in OGRE korrekt verwendet werden, muss der y- und z-Wert der Translation jeweils vertauscht werden. Zusätzlich muss das Vorzeichen des neuen z-Werts geändert werden.

11 Grobe Poseschätzung

Eine erste grobe Poseschätzung ist anhand von GPS-Koordinaten und einer Kompass-Richtung möglich. Prinzipiell muss lediglich eine Translation und eine Rotation durchgeführt werden, da es sich sowohl bei dem durch die UTM-Projektion definierten Koordinatensystem als auch dem in OGRE verwendeten Koordinatensystem um ein dreidimensionales kartesisches Koordinatensystem handelt. Eine Höhenmessung per GPS ist zwar möglich, allerdings sehr unpräzise und daher kaum aussagekräftig. Die Berechnung der Höhenangabe erfordert andere Methoden.

11.1 Koordinatentransformation

Für die Transformation vom UTM- in das OGRE-Koordinatensystem liegen die folgenden Daten vor.

- gpsPosX, gpsPosY: GPS-Koordinaten des Betrachters
- viewDirection: Blickrichtung des Betrachters (Kompass)
- localOriginX, localOriginY: GPS-Position des für OGRE festgelegten Koordinatenursprungs (Ecke des A-Gebäudes)
- coordinateRotation: Rotationsfaktor für die Transformation zwischen UTM- und OGRE-Koordinaten

Eine Umrechnung von UTM- in OGRE-Koordinaten wird innerhalb des PoseInitializers stets von einem GPSTracker-Objekt durchgeführt. Die GPS-Koordinaten und Blickrichtung des Betrachters werden zur Laufzeit über die grafische Benutzeroberfläche des PoseInitializers eingegeben. Die GPS-Koordinaten des OGRE-Koordinatenursprungs sowie der Rotationsparameter wurden mit den im Vorfeld beschriebenen Methoden bestimmt.

Eine Transformation zwischen zwei identisch skalierten, kartesischen Koordinatensystemen setzt sich lediglich aus einer Translation und Rotation zusammen. Der folgende Code des GPSTracker-Konstruktors zeigt, wie durch die Angabe der GPS-Koordinaten des OGRE-Koordinatenursprungs und des Rotationsparameters die Transformation mit einer Translation und Rotation durchgeführt wird. Als Eingabeparameter für den Konstruktor dienen die GPS-Position und Blickrichtung eines Betrachters zur Laufzeit. Die Blickrichtung des Betrachters muss ebenfalls in einen lokalen Rotationsparameter umgerechnet werden.

Listing 6: Konstruktor der GPSTracker-Klasse

```
1  // GPSTracker-constructor
2  GPSTracker::GPSTracker(float gpsPosX, float gpsPosY,
3                         float viewDirection)
4  {
5  // gps-coordinates of OGRE's defined coordinate-origin
6  this->localOriginX = 397590.31;
7  this->localOriginY = 5580165.54;
8
9  // rotation between UTM- and OGRE-coordinates
10 this->coordinateRotation = -200.758;
11
12 // position transformation
13 std::pair <float, float> localPos =
14    computeLocalPosition(gpsPosX, gpsPosY);
15
16 // view-direction transformation
17 this->localOrientation =
18    computeLocalRotation(viewDirection);
19
20 this->localPositionX = localPos.first;
21 this->localPositionZ = localPos.second;
22 }
```

Zu Beginn ist die Translation zwischen den beiden Koordinatensystemen zu berechnen. Dazu wird ein Vektor vom selbst definierten Ursprung des OGRE-Koordinatensystems zur GPS-Position des Betrachters aufgespannt.

$$\begin{pmatrix} T_x \\ T_y \end{pmatrix} = \begin{pmatrix} P_x \\ P_y \end{pmatrix} - \begin{pmatrix} O_x \\ O_y \end{pmatrix} \tag{21}$$

Um die Koordinatentransformation abzuschließen, wird noch eine einfache Rotation angewandt. $R_{GPS \to OGRE}$ ist dabei der Rotationsparameter (hier: -200.758).

$$T_{TR} = \begin{pmatrix} \cos(R_{GPS \to OGRE}) \cdot T_y - \sin(R_{GPS \to OGRE}) \cdot T_x \\ \sin(R_{GPS \to OGRE}) \cdot T_y + \cos(R_{GPS \to OGRE}) \cdot T_x \end{pmatrix} \tag{22}$$

Um die Blickrichtung der virtuellen Kamera mit der Blickrichtung des realen Betrachters gleichzusetzen, wird auch dieser Wert transformiert.

$$R_{OGRE} = (-1) \cdot (R_{GPS \to OGRE} + 90) - R_{GPS} \tag{23}$$

Abbildung 22 zeigt das für OGRE definierte Koordinatensystem, welches am Grundriss des A-Gebäudes ausgerichtet ist.

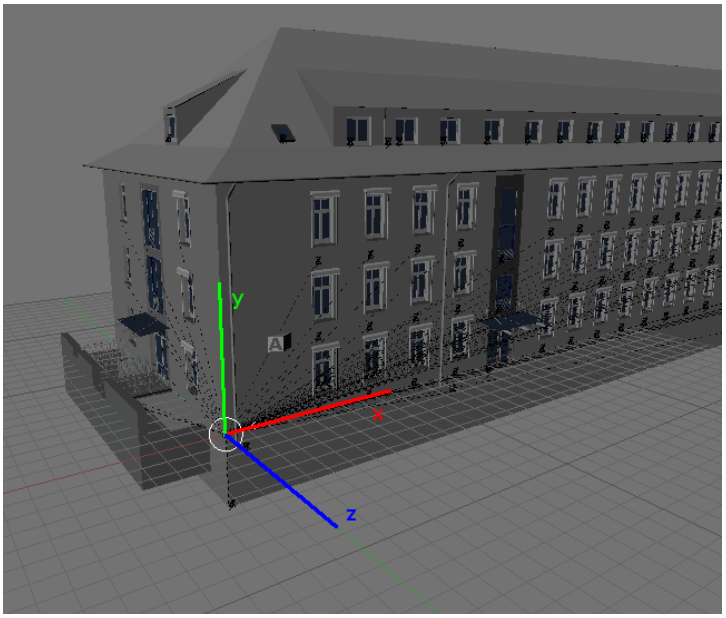

Abbildung 22: In OGRE definiertes Koordinatensystem

Standardmäßig blickt eine Kamera in OGRE stets entlang der negativen z-Achse. Dies würde einer Rotation von 0 entsprechen. Schaut ein Betrachter nach Norden, so zeigt der Kompass ebenfalls eine Rotation von 0 an. Eine Rotation der negativen z-Achse von 90° gegen den Uhrzeigersinn um die y-Achse entspricht der negativen x-Achse. Die Ausrichtung der x-Achse in Bezug auf die Himmelsrichtung wurde bereits bestimmt. Nun kann eine Rotation der negativen x-Achse im Uhrzeigersinn um die y-Achse durchgeführt werden. Nach Anwendung dieser Transformation

der Blickrichtung entspricht Norden der negativen z-Achse in OGRE. Der auf dem Kompass abgelesene Wert kann mit dieser Transformation als Rotation im Uhrzeigersinn in OGRE angewandt werden.

Bisher kann zwar eine virtuelle Kamera in OGRE zumindest in der xz-Ebene platziert werden, allerdings ist das Testszenario keine planare Ebene. Dementsprechend muss auch ein y-Wert als Höhenangabe berechnet werden. Für die Berechnung des besagten Wertes wird das im Modell des Campus enthaltene Terrain herangezogen, da eine Höhenberechnung per GPS, wie bereits erwähnt, zu unpräzise wäre. Im folgenden Code wird das Terrain dem Szenegraphen von OGRE hinzugefügt. Dazu ist das Erstellen eines Szene-Knotens nötig, der die einzelnen Transformationen und weitere Eigenschaften der dazugehörigen Entität, also dem Modell selbst, beinhaltet. Wichtig dabei ist auch, die Reihenfolge der zu rendernden Objekte zu beeinflussen. Das Terrain muss vor allen Gebäuden gezeichnet werden, damit Artefakte bei der Verdeckung nicht sichtbarer Teile vermieden werden.

Listing 7: Einfügen des Terrains in OGREs Szenegraphen

```
1  // create scene node
2  modelTerrainNode = windowSceneManager
3                     ->getRootSceneNode()
4                     ->createChildSceneNode("mTNode");
5
6  // create entity and attach it to the node
7  modelTerrainObject = windowSceneManager
8                       ->createEntity("Terrain",
9                                      "Terrain.mesh");
10
11 modelTerrainNode->attachObject(modelTerrainObject);
12
13 // apply transformations
14 modelTerrainNode->translate(345.369, 3.086, 153.599);
15 modelTerrainNode->yaw(Ogre::Degree(110.747));
16
17 // render the terrain at the beginning
18 modelTerrainObject->setRenderQueueGroup(10);
```

Die Transformationsparameter werden wie bei den restlichen Modellen aus Blender ausgelesen und können dann zur Laufzeit angewandt werden. Um eine Höhenangabe für eine bestimmte Position in der xz-Ebene zu berechnen, soll das Terrain mit einem synthetisch generierten Strahl geschnitten werden, der von einer bestimmten Höhe senkrecht auf das Terrain geschossen wird. Der Schnittpunkt mit dem Terrain an der Position des Betrachters lässt einen Rückschluss auf den gesuchten y-Wert zu. OGRE un-

terstützt Schnitttests mit Strahlen zwar, jedoch werden nicht die Objekte selbst getestet, sondern nur deren Bounding-Boxes[17]. Abbildung 23 verdeutlicht das Problem. Um korrekte Höhenangaben des Terrains an einer bestimmten Position zu berechnen, muss das Modell selbst und somit letztlich jedes Polygon des Modells mit einem Strahl geschnitten werden. Zwar ist diese Vorgehensweise nicht sonderlich performant, dafür ist der Grad der Genauigkeit sehr gut. Eine existierende OGRE-Implementierung dieses Ansatzes findet man unter [19]. Der dort beschriebene Code wird in der Implementierung des PoseInitializers in leicht modifizierter Form verwendet. Der Strahl wird von einer bestimmten Höhe y senkrecht auf das Gelände geschossen. Die Differenz zwischen der gesamten Länge des Strahls bis zur xz-Ebene in OGRE und der Länge des Strahls bis zum Schnittpunkt mit dem Terrain ergibt die Höhe des Terrains an der gewünschten Stelle. Zu diesem Ergebnis wird noch die Höhe eines Betrachters (z.B. 1,80 Meter) addiert.

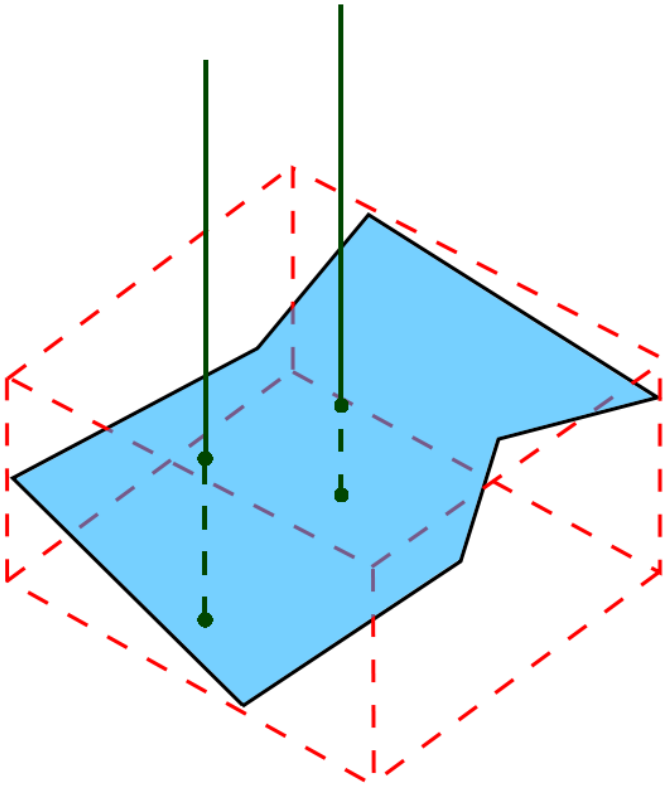

Abbildung 23: Schnitttest des Terrains per Raycasting mit Bounding-Boxes bzw. dem Polygonnetz

[17] OGRE unterstützt nur Axis-Aligned-Bounding-Boxes

Mit den bisherigen Ansätzen lässt sich eine grobe Poseschätzung bereits durchführen. Sofern eine GPS-Position und eine Kompassrichtung vorhanden sind, kann der virtuelle Betrachter im Campusmodell platziert werden. Für eine ausreichend präzise Schätzung der Pose reicht dies allerdings noch nicht aus. Die Gründe dafür sind wie folgt.

- Eine Ortung per GPS ist sehr ungenau.
- Die Präzision von GPS ist von vielen verschiedenen Faktoren abhängig.
- Lediglich die Drehung um OGREs y-Achse wird berücksichtigt. Der Betrachter schaut allerdings nie genau parallel zur xz-Ebene in OGRE. Bei einer Betrachtung von Gebäuden wird der Kopf bzw. die Kamera meist unbewusst leicht in Richtung des Himmels geneigt.

Um die angesprochenen Probleme zu lösen, wird im späteren Verlauf der Arbeit auf optisches markerloses Tracking zurückgegriffen. Zuvor soll jedoch ein anderes Thema behandelt werden. Für einen urbanen Kontext kann die Anzahl der verwendeten Modelle leicht sehr groß werden. Um zur Laufzeit weiterhin performant arbeiten zu können, ist ein kontextsensitives Laden der Modelle notwendig. Es sollen nur diejenigen Modelle geladen werden, die in Bezug auf die aktuelle Position und Blickrichtung eines Betrachters relevant sind.

11.2 Dynamisches Laden der Modelle

Im bisherigen Verlauf dieser Arbeit wurde beschrieben, wie Modelle mit zusätzlichen Informationen unter Verwendung einer XML-Datei verknüpft werden können. Die Erstellung dieser besagten XML-Datei ist für das dynamische Laden der Modelle zur Laufzeit essentiell, sofern die Position und Blickrichtung des Betrachters berücksichtigt werden soll. Der Ablauf einer Filterung der Modelle anhand ihrer Wichtigkeit zu einem konkreten Zeitpunkt soll in diesem Abschnitt etwas detaillierter betrachtet werden.

Die Anzahl der Gebäudemodelle einer realen Szene lässt sich durch das Traversieren der XML-Struktur leicht berechnen. Unabhängig davon, wie viele Modelle benutzt werden sollen, wird beim Start des PoseInitializers einmal die besagte XML-Datei durchlaufen, um die Anzahl der Modelle zu bestimmen. Dieser Schritt ist nötig, um den Szenegraphen in OGRE beim Programmstart vorzubereiten. Vorteilhaft an dieser Vorgehensweise ist, dass der OGRE-Szenegraph stets anhand der XML-Datei konfiguriert wird und kein weiterer Eingriff in den Quellcode notwendig ist, auch wenn das System um weitere Modelle ergänzt werden sollte.

Das Traversieren der XML-Struktur zum Erstellen der Instanzen einer Model-Klasse ist Aufgabe eines zur Laufzeit erstellten ModelFinder-Objekts.

Der folgende Code zeigt den Anfang der parseXMLFile()-Methode, welche innerhalb der ModelFinder-Klasse implementiert wurde. Als Bibliothek zum Parsen von XML-Strukturen wird in diesem Projekt TinyXML [11] verwendet.

Listing 8: Parsen der XML-Datei mit TinyXML

```
1  ...
2  // load the xml
3  TiXmlDocument doc(this->xmlFileName);
4  bool loadOK = doc.LoadFile();
5
6  if (loadOK)
7  {
8   // go to the root and get the first element
9   TiXmlElement *root = doc.RootElement();
10  for(TiXmlElement* xmlM = root->FirstChildElement();
11        xmlM;
12        xmlM = xml->NextSiblingElement())
13  {
14    // the root's children are always new models
15    // create the model
16    Model *myModel= new Model(xmlM->Attribute("id"));
17
18    // set the name of the mesh-file
19    myModel->setMeshName(xmlM->FirstChild("meshName")
20            ->FirstChild()->Value());
21    ...
22    }
23   ...
24  }
```

Wird beim Parsen jeweils ein direkter Kindknoten des Wurzelelements gefunden, so wird ein neues Model-Objekt erzeugt. Im Anschluss werden die einzelnen in der XML-Datei hinterlegten Eigenschaften dem Model-Objekt hinzugefügt. So ist jedes mögliche Modell als Instanz der Model-Klasse verfügbar und kann dem OGRE-Szenegraphen hinzugefügt werden.

Um nur die in Bezug auf die aktuelle Position und Blickrichtung eines Betrachters wichtigsten Gebäude zur Laufzeit zu laden, muss jedes Gebäude mit einer Art künstlich erstelltem Frustum[18] vom Betrachter aus getestet werden. Dazu muss zu Beginn eine Reihe von Kontrollpunkten entlang des Grundrisses eines Gebäudes definiert werden.

[18] Kegelstumpf

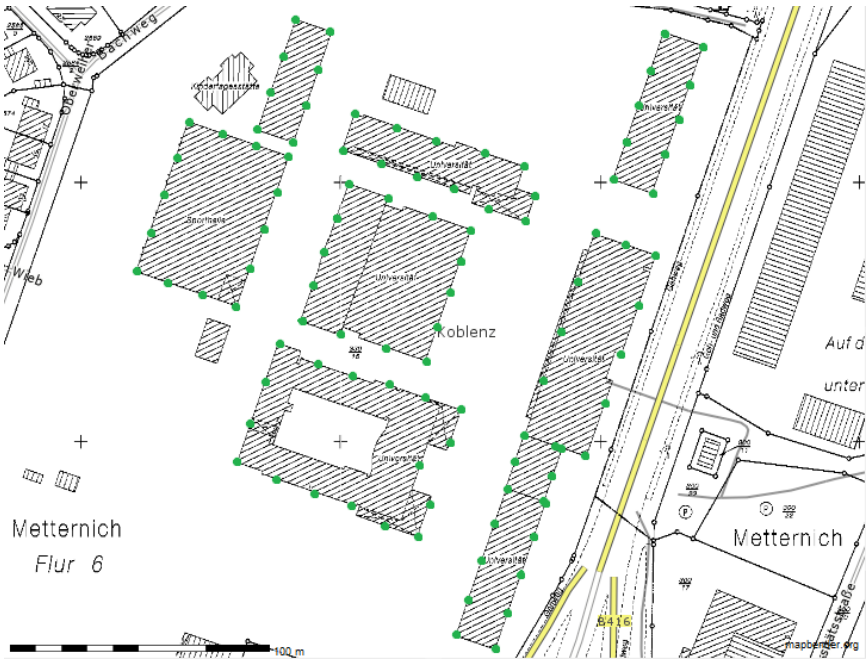

Abbildung 24: Definierte Kontrollpunkte der Gebäude auf dem Campus [13]

Abbildung 24 stellt die manuell ausgewählten Kontrollpunkte der Gebäude des Campus Koblenz dar. Die GPS-Positionen der einzelnen Kontrollpunkte werden wie beschrieben der XML-Datei hinzugefügt. Für die Analyse, ob ein Gebäude zur Laufzeit geladen werden muss, werden im Konstruktor des ModelFinders folgende Parameter benötigt.

- Name der XML-Datei
- aktuelle GPS-Position P_{GPS} des Betrachters
- aktuelle Blickrichtung R_{GPS} des Betrachters
- Winkel ω des Suchfeldes
- minimale erlaubte Entfernung d_{min}
- maximale erlaubte Entfernung d_{max}

Abbildung 25 visualisiert, wie anhand der Position und Blickrichtung eines Betrachters, Kontrollpunkte der einzelnen Gebäude getestet werden. Ein Kontrollpunkt gilt dabei als interessant, sobald er innerhalb des festgelegten Suchbereiches liegt. Um den Betrachter wird ein großer und ein kleiner Radius definiert. Ist ein Kontrollpunkt weiter entfernt vom Betrachter bzw. näher liegend am Betrachter, so wird der Kontrollpunkt nicht berücksichtigt. Die Blickrichtung des Betrachters halbiert den festgelegten Winkel

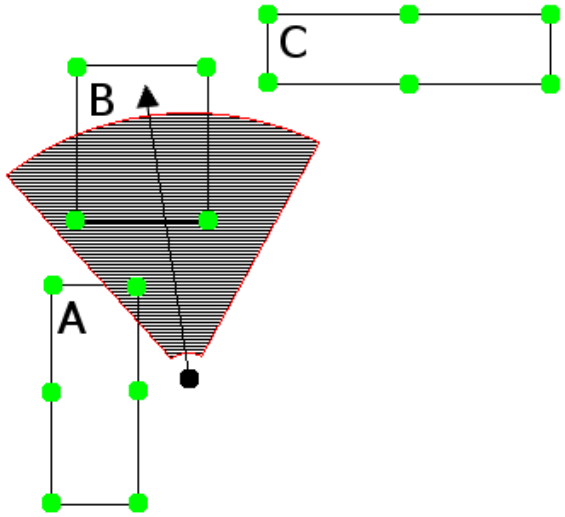

Abbildung 25: Definierter Suchbereich für alle Kontrollpunkte der Gebäude

ω des Suchfeldes. In dem hier gezeigten Beispiel finden sich insgesamt drei gültige Kontrollpunkte von zwei verschiedenen Gebäuden.

Die nötigen Berechnungen für die Klassifikation eines Gebäudes werden in der PriorityChecker-Klasse zur Verfügung gestellt. Bei der Erstellung einer Instanz dieser Klasse wird sowohl eine Liste aller Model-Objekte als auch die Parameter zur Definition des Suchfeldes übergeben. Damit ein Kontrollpunkt K als gültig bezeichnet werden darf, müssen zwei Kriterien erfüllt sein, die im Folgenden als *Distanz-* und *Winkel-Kriterium* bezeichnet werden.

Das erste Kriterium ist die Entfernung des Kontrollpunktes zur Position des Betrachters. Mathematisch berechnet sich die Entfernung zwischen Punkten einer Ebene wie folgt.

$$d(a,b) = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \tag{24}$$

Der für die Entfernung der Punkte berechnete Wert kann nun einfach mit der minimalen und maximalen Entfernung verglichen werden. Dabei ist das Distanz-Kriterium erfüllt, wenn die Distanz zwischen der Position des Kontrollpunktes und des Betrachters in den geforderten Grenzen liegt.

$$d(a,b) > d_{min} \text{ und } d(a,b) < d_{max} \tag{25}$$

Das zweite Kriterium überprüft den Winkel zwischen zwei Vektoren. Ein Vektor zeigt dabei vom Betrachter aus genau in Richtung Norden. Der zweite Vektor verläuft von der Position des Betrachters zu einem Kontrollpunkt. Zur Berechnung des Winkels zwischen zwei Vektoren eignet sich die Definition des Skalarprodukts.

$$\vec{a} \cdot \vec{b} = |\vec{a}||\vec{b}| \cos \sphericalangle(\vec{a}, \vec{b}) \tag{26}$$

Der Winkel zwischen zwei Vektoren lässt sich dementsprechend folgendermaßen berechnen.

$$\sphericalangle(\vec{a}, \vec{b}) = \arccos \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} \tag{27}$$

Um zu prüfen, ob ein Kontrollpunkt das Winkel-Kriterium erfüllt, muss die Richtung des Vektors vom Betrachter zum Kontrollpunkt mit seiner tatsächlichen Blickrichtung verglichen werden. Dabei ist es wichtig, dass die Winkelangabe des besagten Vektors zwischen Kontrollpunkt und Betrachter genau wie die Blickrichtung eines Betrachters selbst als Himmelsrichtung angegeben werden muss. Dies kann dadurch erreicht werden, indem einer der Punkte der vorigen Formel als nach Norden zeigender Vektor definiert wird. Die beschriebene Berechnung liefert stets den kleinsten Winkel zwischen zwei Vektoren. Eine Interpretation dieser Winkel als Himmelsrichtung ist somit noch nicht möglich. Angenommen der Endpunkt eines Vektors $\vec{z}$ liegt weiter westlich in der realen Szene als der Startpunkt, so muss der errechnete Winkel von 360 subtrahiert werden. Abbildung 26 verdeutlicht das Problem. Auch wenn die beiden Vektoren unterschiedliche Endpunkte besitzen, so hätten die errechneten Winkel dennoch denselben Wert.

Da Winkel zwischen Vektoren nun als Himmelsrichtungen mit einem Wertebereich $[0..360]$ dargestellt werden können, ist der Vergleich einer Blickrichtung mit den definierten Vektoren des Suchfeldes möglich. Das Winkel-Kriterium ist erfüllt, wenn eine der drei Bedingungen gültig ist.

$$\left| \sphericalangle(\overrightarrow{P_{GPS}N}, \overrightarrow{P_{GPS}K}) - R_{GPS} \right| \leq \frac{\omega}{2} \tag{28}$$

$$R_{GPS} < \frac{\omega}{2} \text{ und } \sphericalangle(\overrightarrow{P_{GPS}N}, \overrightarrow{P_{GPS}K}) > \left(360 - \frac{\omega}{2} + R_{GPS}\right) \tag{29}$$

$$R_{GPS} > \left(360 - \frac{\omega}{2}\right) \text{ und } \sphericalangle(\overrightarrow{P_{GPS}N}, \overrightarrow{P_{GPS}K}) < \left(360 - R_{GPS} + \frac{\omega}{2}\right) \tag{30}$$

Sollte sowohl das Winkel- als auch das Distanz-Kriterium erfüllt sein, so handelt es sich bei dem untersuchten Kontrollpunkt K um einen in Bezug auf das definierte Suchfeld gültigen Kontrollpunkt. Abbildung 27

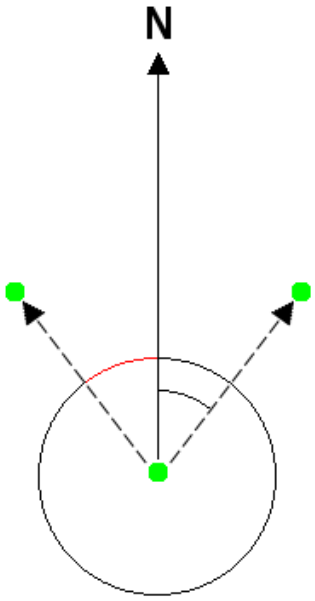

Abbildung 26: Darstellung der Winkel als Himmelsrichtung

fasst die Analyse der Kontrollpunkte in einem UML-Aktivitätsdiagramm zusammen.

Um nun die für eine bestimmte Position und Blickrichtung eines Betrachters wichtigsten Gebäude zu finden, müssen alle Kontrollpunkte eines jeden Gebäudes unter Berücksichtigung der eben genannten Kriterien analysiert werden. Das PriorityChecker-Objekt testet alle Kontrollpunkte eines jeden Model-Objekts. Hält ein Kontrollpunkt den Kriterien nicht stand, so wird die Kopie des Kontrollpunktes innerhalb des PriorityChecker-Objekts gelöscht. Im Anschluss werden diejenigen Model-Objekte gesucht, bei denen die Anzahl der verbliebenen Kontrollpunkte maximal ist.

Lediglich die Modelle mit einer maximalen Anzahl von Kontrollpunkten zu laden kann jedoch in vielen Fällen zu Problemen führen. Abbildung 25 zeigte bereits die Suche nach gültigen Kontrollpunkten innerhalb eines selbst definierten Bereiches. Man erkennt, dass Modell A und B gültige Kontrollpunkte besitzen. Obwohl Modell B zwei gültige und damit auch die maximale Anzahl der gültigen Kontrollpunkte besitzt, so muss Modell A dennoch geladen werden, da es Teile von Modell B verdeckt. Abbildung 28 verdeutlicht noch einmal die Auswirkungen, falls verdeckende Modelle nicht geladen werden.

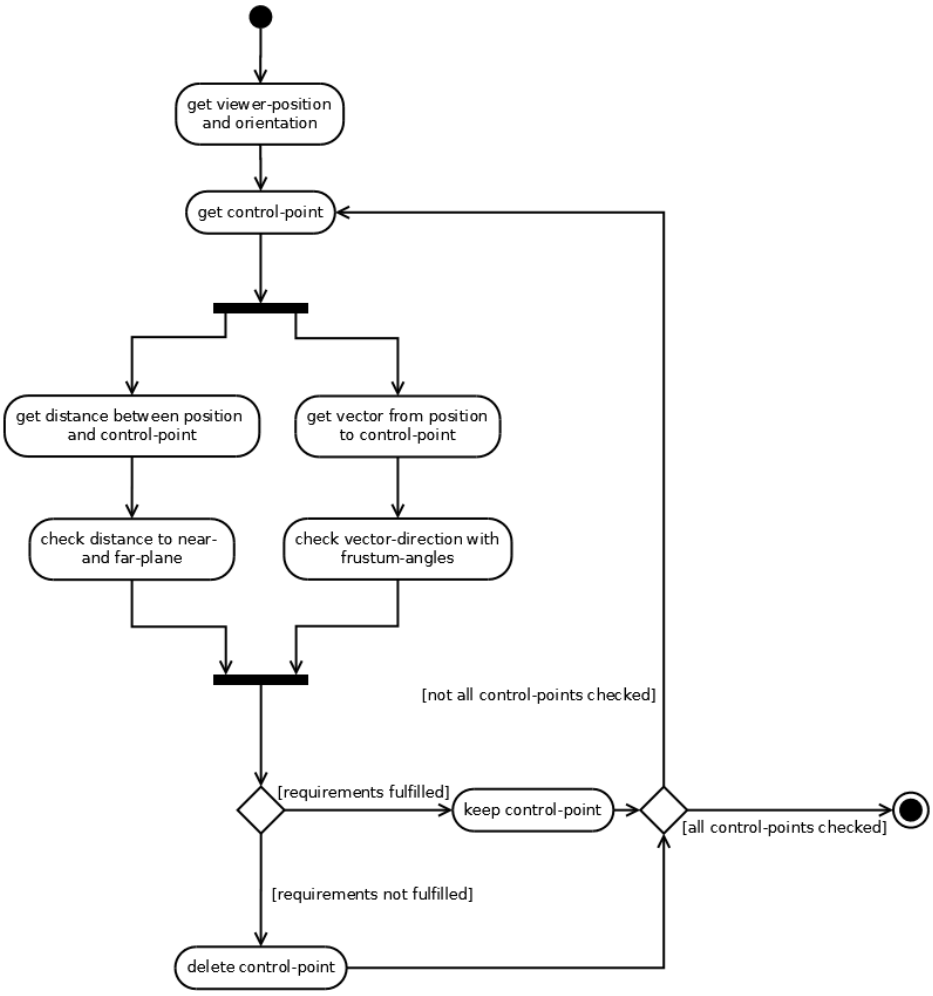

Abbildung 27: Überprüfung der Gültigkeit der Kontrollpunkte unter Berücksichtigung der Position und Blickrichtung des Betrachters

Abbildung 28: Fehlende Verdeckung

Abbildung 29 fasst den gesamten Ladevorgang der Modelle in einem Aktivitätsdiagramm zusammen.

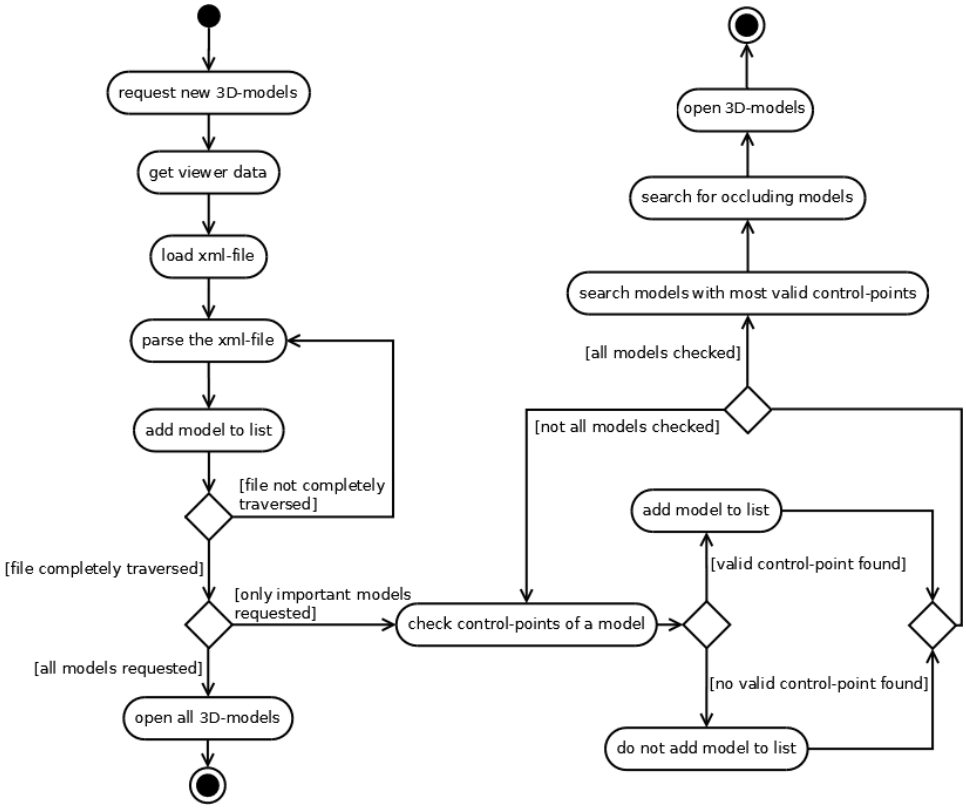

Abbildung 29: Laden der Modelle

Abschließend ist noch die Überlegung nach einer alternativen Möglichkeit zur manuellen Bestimmung der Kontrollpunkte interessant, da dies je nach Anzahl der Modelle einen nicht unerheblichen Aufwand darstellt. Es wäre beispielsweise möglich, für jedes Gebäude lediglich den Mittelpunkt des Grundrisses mit GPS-Koordinaten zu beschreiben. Zusätzlich könnte die Länge der Grundrisskante in Richtung der größten Ausdehnung und die Himmelsrichtung dieser Kante vermessen werden. Eine Klassifizierung der Gebäude zur Laufzeit unter Berücksichtigung der Position und Blickrichtung des Betrachters wäre mit diesen Daten durchaus denkbar. In Abbildung 30 wird allerdings ein Problem dieses Ansatzes klar. Obwohl der Betrachter aus seiner derzeitigen Position und Blickrichtung keine Teile des Gebäudes sehen kann, wird das Modell geladen, da der Betrachter sich sehr nah am geometrischen Mittelpunkt des Grundrisses befindet. Für größere Gebäude, wie beispielsweise historische Bauten und Schlösser, ist diese Form des Grundrisses nicht unüblich. Bedenkt man die Komplexität des zu ladenden Modells, sollte ein Laden und Anzeigen dieses Objekts auch nur

dann geschehen, wenn der Betrachter es auch wirklich sehen kann. Dieser alternative Lösungsansatz kann also kein fehlerfreies Verhalten bei konkaven Grundrissen gewährleisten. Hinzu kommt, dass eine korrekte Analyse von der Genauigkeit der Messung eines einzigen Punktes und einer Richtung abhängig gemacht wird. Eine manuelle Vermessung mehrerer Kontrollpunkte schafft hingegen zusätzliche Sicherheit.

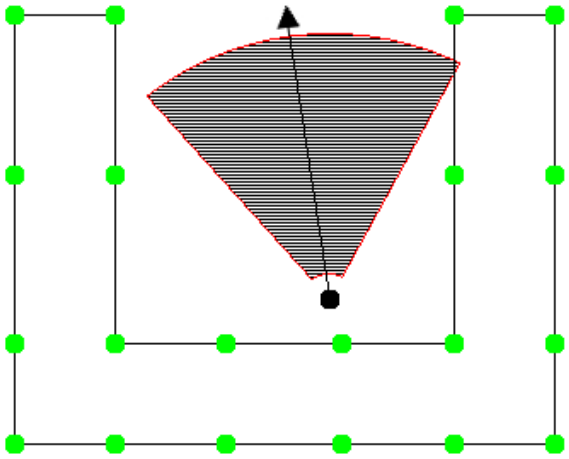

Abbildung 30: Definierter Suchbereich für alle Kontrollpunkte des konkaven Gebäudegrundrisses

12 Verfeinerung der Poseschätzung

12.1 Vorgehensweise

Mit dem bisher beschriebenen Verfahren ist eine erste grobe Schätzung der initialen Kamerapose anhand von GPS-Koordinaten und einer Kompassrichtung realisiert worden. Wie im Vorfeld bereits beschrieben, ist eine GPS-Messung meist recht unpräzise. Aus diesem Grund soll der Ansatz *Analyse durch Synthese* eine Anwendung in dieser Arbeit finden. Zur Erzeugung einer Reihe von Kamerabildern des virtuellen Modells muss auf eine virtuelle Kamera eine Reihe von Translationen und Rotationen angewandt werden. Diese Reihe von Transformationen wird im Folgenden als *Posestreuen* bezeichnet.

Posen können verschiedenartig gestreut werden. Eine Möglichkeit ist dabei ein iterativer stochastischer Ansatz. In einem ersten Schritt werden einige Posen sehr grob um die erste Pose, die durch GPS-Position und Blickrichtung gegeben ist, verstreut. Nun kann aus jeder Pose heraus ein Bild der virtuellen Szene erstellt und mit dem Foto der realen Szene verglichen werden. Diejenigen Posen, deren Bilder die geringste Differenz zum Foto der realen Szene aufweisen, dienen als Eingabepose für den nächsten Iterationsschritt. Dieses Verfahren wird beendet, wenn eine festgelegte Differenz zwischen den Bildern der realen und virtuellen Szene eine definierte Grenze unterschreitet. Abbildung 31 zeigt ein Beispiel. Die neue Position einer Kamera wird zufällig innerhalb der definierten Grenzen gewählt. Nach jeder Iteration verkleinert sich der mögliche Bereich für die Streuung weiterer Posen. Im Beispiel wird nach der dritten Iteration eine Pose gefunden, bei denen die zu vergleichenden Bilder einen hinreichend kleinen Abstand haben.

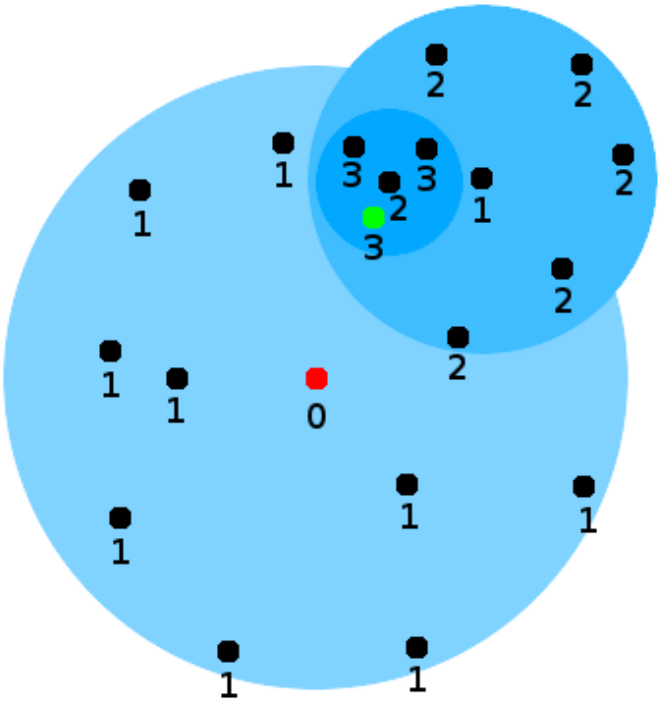

Abbildung 31: Beispielhafter Ablauf für iteratives Posestreuen[19]

Ein weitaus problemloserer Ansatz zum Streuen der Posen ist eine regelmäßige, strukturierte Verteilung der Posen mit festen Parametern. Um eine Eingabepose wird ein festes Raster definiert, welches die weiteren Po-

[19] rot: erste Pose, grün: gesuchte Pose

sitionen für die zu testenden Kameraposen festlegt. Abbildung 32 stellt ein Beispiel für regelmäßiges Posestreuen dar. Es fällt direkt auf, dass in jedem Fall eine Vielzahl von Posen generiert wird, die fernab der gesuchten Pose liegen.

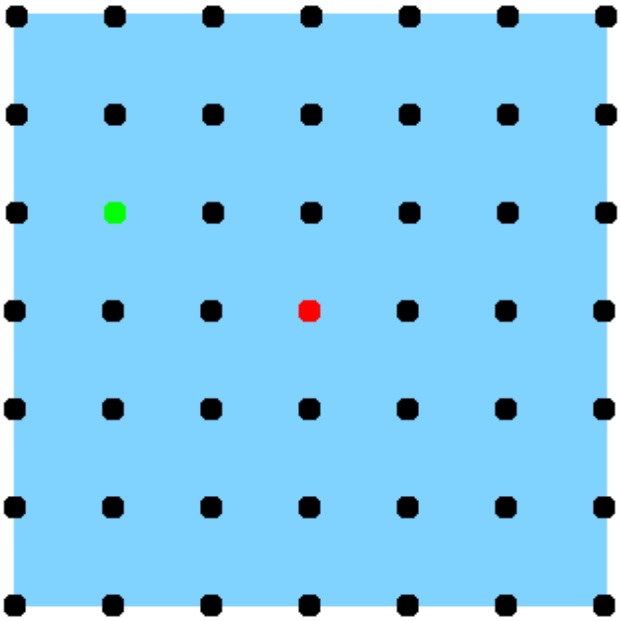

Abbildung 32: Beispielhafter Ablauf für regelmäßiges Posestreuen[20]

Auch wenn das Konzept des regelmäßigen Posestreuens auf den ersten Blick nicht ideal zu sein scheint, wurde dieser Ansatz für eine Implementierung gewählt. Relevant waren dafür die folgenden Gründe.

- Die Betrachterhöhe wird bekanntlich über einen Schnitttest mit dem Terrain ermittelt. Für das Posestreuen sind alleinige Veränderungen der Position nicht ausreichend. Auch die Rotation der Kamera muss in den einzelnen Posen verändert werden. Sollte für das Posestreuen der stochastische Ansatz beibehalten werden, so hätte dies einen enormen Anstieg der Menge von nötigen Schnitttests zur Folge. Beim regelmäßigen Posestreuen können nach einer Änderung der Position erst einige Posen erstellt werden, die lediglich eine veränderte Kameraorientierung aufweisen. Somit ist ein Schnitttest mit dem Terrain nicht bei jeder neuen Pose notwendig.

[20] rot: erste Pose, grün: gesuchte Pose

- Unter Umständen kann der iterative Ansatz schneller konvergieren als das Konzept des regelmäßigen Posestreuens. Im ungünstigen Fall kann es jedoch auch mehr Zeit in Anspruch nehmen, da die gesuchte Pose nicht direkt gefunden wird. Der regelmäßige Ansatz benötigt immer dieselbe Zeit zur Bestimmung der besten Pose.

- Im weiteren Verlauf dieser Arbeit werden zwei Ansätze zum Vergleichen von Bildern beschrieben. Mit beiden Ansätzen hat sich ein regelmäßiges Streuen der Posen als effizienter und sicherer erwiesen. Der Grund dafür ist, dass bei den vorgestellten Bildvergleichen nur eine hohe Übereinstimmung gefunden wird, wenn die Bilder auch sehr ähnlich sind. Sobald eine zu große Abweichung der Kamerapose besteht, werden die Differenzen zwischen den Bildern schlagartig sehr groß.

Jede Pose wird innerhalb des PoseInitializers durch ein Pose-Objekt repräsentiert. Pose-Objekte enthalten dabei unter anderem Parameter für Translationen und Rotationen. Eine erste grobe Pose wird durch die Umrechnung von GPS- und Kompassdaten bestimmt. Anhand dieser Pose können nun weitere Posen durch Erstellung eines PoseCreator-Objekts generiert werden. Der folgende Code zeigt den Konstruktor der PoseCreator-Klasse.

Listing 9: Konstruktor der PoseCreator-Klasse

```
1  // constructor for a PoseCreator
2  PoseCreator::PoseCreator(Pose *inputPose,
3                           int poseQuantity)
4  {
5          // set some attributes
6          this->inputPose = inputPose;
7          this->poseQuantity = poseQuantity;
8
9          // create more poses
10         generatePoseVector();
11 }
```

Ein weiterer Auszug des Codes erklärt, wie neue Posen generiert und regelmäßig verstreut werden. Die Angabe der maximalen Translation bzw. Rotation hängt dabei von der Anzahl der zu erstellenden Posen ab. In diesem Fall werden insgesamt 676 Kameraposen erstellt. Die erstellten Posen werden in einer Liste gespeichert, die innerhalb des QOgreGLWidget-Objekts durchlaufen wird. Die Eigenschaften eines Pose-Objekts bestimmen die Translation und Rotation der virtuellen Kamera. Neue Kameraposen werden in regelmäßigen Abständen von zwei Metern in einem Raster

mit einer maximalen Kantenlänge von 24 Metern gestreut. An jeder neuen Position werden vier verschiedene Winkel für die Neigung der Kamera gesetzt. Andere Parameter für die Rotation der Kamera können durch die verwendeten Bildvergleiche vernachlässigt werden. Die Anzahl der zu erstellenden Posen kann dadurch auf einem niedrigen Niveau gehalten werden. Im weiteren Verlauf der Arbeit wird sich zeigen, dass dies zu keinem Nachteil bei der Verfeinerung der Poseschätzung führt.

Listing 10: Regelmäßiges Streuen neuer Posen

```
1  ...
2  // start values for translation and rotation
3  // measurement units is in meters
4  int xPosShift = -12;
5  int yPosShift = -12;
6  int pitchShift = 0;
7
8  ...
9  for (int i = 0; i < this->poseQuantity; i++)
10  {
11  // create a new pose and write it to the pose-vector
12  this->poseVector.push_back(new Pose(
13  this->inputPose->getGpsX()+xPosShift,
14  this->inputPose->getGpsY()+yPosShift,
15  this->inputPose->getGpsDirection(),
16  pitchShift,
17  0));
18
19   // pitch
20   // possible values: 0, 5, 10, 15
21   if ((i+1) % 4 == 0) pitchShift = 0;
22    else pitchShift += 5;
23
24   // gps-x-coordinate
25   // possible values: -12,-10,..,0,..,10,12
26   if ((i+1) % 52 == 0) xPosShift = -12;
27    else if ((i+1) % 4 == 0) xPosShift += 2;
28
29   // gps-y-coordinate
30   // possible values: -12,-10,..,0,..,10,12
31   if ((i+1) % 676 == 0) yPosShift = -12;
32    else if ((i+1) % 52 == 0) yPosShift += 2;
33  }
```

Für eine Schätzung der initialen Pose wird nun jede der Posen einmal gesetzt und die Ergebnisbilder miteinander verglichen. Danach werden alle Posen nach der Ähnlichkeit ihrer Bilder zum Foto der realen Szene sortiert.

Unklar ist bisher immer noch, wie die Bilder einer realen und virtuellen Szene miteinander verglichen werden sollen. Im weiteren Verlauf werden nun zwei Lösungsansätze vorgestellt. Alle Methoden zur Analyse zweier Bilder sind innerhalb der ImgAnalyzer-Klasse implementiert.

12.2 Bildvergleich basierend auf einer Hough-Transformation

12.2.1 Idee

Da der PoseInitializer auf die Anwendung in einem urbanen Kontext beschränkt ist, eignen sich kantenbasierte Bildvergleiche besonders gut. Eine Möglichkeit zur Kantenextraktion in Bildern ist die bereits beschriebene Hough-Transformation[21]. Nachdem jeweils eine HT auf das Bild der realen und der virtuellen Szene angewandt wurde, sollen die folgenden Daten der Kanten zweier Bilder miteinander verglichen werden.

- Abstand ρ der Kante zum Ursprung des Bildkoordinatensystems
- Orientierung der Kante als Winkel θ
- Länge l einer Kante
- Start- und Endpunkt der Kante

Die Anwendung einer gewöhnlichen HT mit OpenCV liefert lediglich den Abstand ρ und den Winkel θ. Da die Start- und Endpunkte der Kanten aus diesen Daten nicht berechnet werden können, ist ein Vergleich basierend auf den vorgeschlagenen Werten nicht durchführbar. Die sogenannte *probabilistische HT* [20] liefert als Ergebnis die Start- und Endpunkte einer gefundenen Kante. Die Lage, Ausrichtung und Länge einer Kante können aus diesen Informationen leicht berechnet werden.

12.2.2 Realisierung

Bevor die probabilistische HT angewandt werden kann, wird auf dem Foto der realen Szene zuerst ein *Histogrammausgleich* durchgeführt, der zur Verbesserung des Kontrastes und somit zu einer zuverlässigeren Detektion der Kanten führen soll. Danach wird auf beiden Bildern der Canny-Algorithmus angewandt. Dadurch werden die Fotos der realen und virtuellen Szene auf binäre Kantenbilder reduziert. Auf das Ergebnis des Canny-Algorithmus wird im Anschluss die probabilistische HT angewandt.

[21] im Folgenden als HT bezeichnet

Die Durchführung der hier beschriebenen Schritte setzt die Festlegung einer Reihe von Parametern voraus, die der Benutzer zur Laufzeit über die GUI ändern kann. Abbildung 33 zeigt die konfigurierbaren Parameter des PoseInitializers mit ihren vorbelegten Werten. Je nach Art des zu transformierenden Bildes müssen diese Parameter eventuell angepasst werden.

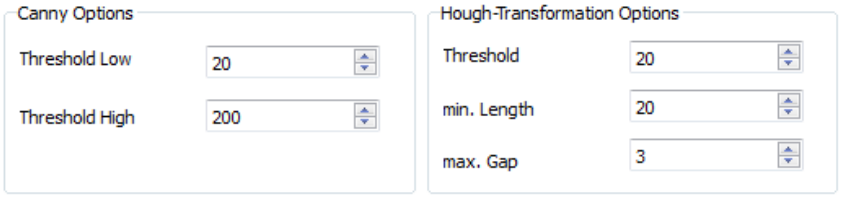

Abbildung 33: Einstellungen der Parameter für Canny und Hough über die GUI des PoseInitializers

Die Manipulation der einzelnen Parameter ist für beide Eingabebilder separat einstellbar. Die Bedeutung der Parameter ist folgendermaßen definiert.

- Canny-Algorithmus
  - Threshold Low: unterer Schwellwert C_L
  - Threshold High: oberer Schwellwert C_H
- Hough-Transformation
  - Threshold: minimale Anzahl von nötigen Punkten, die auf einer Geraden liegen müssen
  - min. Length: erlaubte minimale Länge einer Kante
  - max. Gap: erlaubte maximale Lücke zwischen zwei zu einer Kante gehörigen Segmenten

Je nach Wahl der Parameter ergeben sich sehr unterschiedliche Ergebnisbilder der HT. Die Anwendung der probabilistischen HT liefert eine Liste von zugehörigen Start- und Endpunkten, die jeweils eine gefundene Kante repräsentieren. Um die beiden Ergebnisse miteinander vergleichen zu können, wird jede Kante des ersten Bildes mit jeder Kante des zweiten Bildes verglichen. Je mehr Kanten der beiden Bilder als zueinander passend erkannt werden, desto größer ist das Maß der Ähnlichkeit zwischen beiden Bildern.

Für den Vergleich zweier Hough-Kanten sind zwei Punktepaare mit jeweils (x_a, y_a) als Startpunkt und (x_b, y_b) als Endpunkt gegeben. Abbildung 34 zeigt eine Hough-Linie im zweidimensionalen kartesischen Bildkoordinatensystem, welches seinen Ursprung in der linken oberen Ecke hat.

Ausgehend von den gegebenen Informationen lässt sich der Winkel θ der entsprechenden Hough-Linie wie folgt berechnen.

$$\Delta_x = x_b - x_a \tag{31}$$

$$\Delta_y = -(y_b - y_a) \tag{32}$$

$$\theta = \arctan \frac{\Delta_x}{\Delta_y} \tag{33}$$

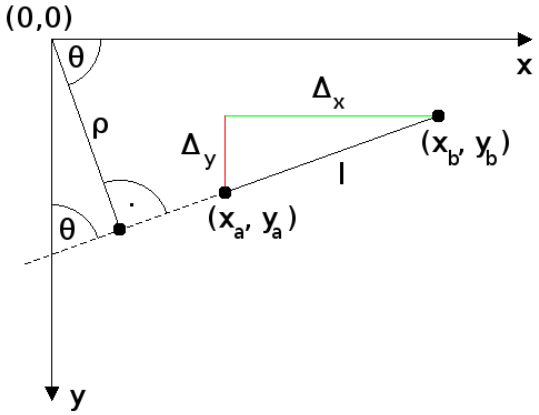

Abbildung 34: Hough-Linie mit Start- und Endpunkt

Als weitere Eigenschaft einer Hough-Linie lässt sich der Abstand zum Bildursprung bestimmen. Gemessen wird der Abstand ρ anhand der Normalen einer Hough-Linie geschnitten mit dem Ursprung des Bildkoordinatensystems. Über den Winkel lässt sich mit einem beliebigen bekannten Punkt der Geraden der Abstand ρ berechnen. Ein bekannter Punkt der Geraden ist beispielsweise durch (x_a, y_a) gegeben.

$$\rho = x_a \cdot \cos(\theta) + y_a \cdot \sin(\theta) \tag{34}$$

Eine andere Möglichkeit zum Vergleich zwischen den Hough-Linien bietet die Länge der über die Start- und Endpunkte definierten Kanten. Die Länge l einer Geraden lässt sich bekanntermaßen mit folgender Formel berechnen.

$$l = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \tag{35}$$

Die aus einer probabilistischen HT ermittelten Kanten können nun anhand der vier beschriebenen Eigenschaften miteinander verglichen werden. Zwei Hough-Kanten zweier unterschiedlicher Bilder sind also ähnlich, sofern die Eigenschaften beider Hough-Kanten ähnlich sind. Beim Vergleich dieser Eigenschaften müssen natürlich gewisse Toleranzen eingeräumt werden. Dies hat zur Folge, dass Störungen im Bild und leichte Unterschiede zwischen den zu vergleichenden Bildern und letztendlich den entsprechenden Kameraposen geduldet werden.

In der Implementierung des PoseInitializers werden zwei Hough-Kanten als ähnlich definiert, wenn die folgenden Merkmale erfüllt sind.

- Für die Winkel θ_1 und θ_2 gilt $|(\theta_1 - \theta_2)| \leq 5°$.
- Für die Abstände ρ_1 und ρ_2 gilt $|(\rho_1 - \rho_2)| \leq 10$ Pixel.[22]
- Für die Länge l_1 und l_2 gilt $|(l_1 - l_2)| \leq 20$.
- Für die Start- und Endpunkte gilt, dass sie nicht weiter als 10 Pixel voneinander entfernt sind.

12.2.3 Ergebnisse

Bei der auf einer HT basierenden Bildanalyse ergeben sich einige Probleme.

- OpenCV findet nicht alle prägnanten Kanten im Bild. Ob eine Kante gefunden wird, hängt stark von der aktuellen Pose ab. Geringe Veränderung im Blickwinkel oder der Position einer Kamera können sich schnell negativ auf die Kantendetektion auswirken, obwohl das menschliche Auge kaum einen Unterschied zwischen den Bildern erkennt. Ob eine eigene Implementierung der HT das Problem beheben würde, darf wohl bezweifelt werden.
- Zwei Hough-Kanten werden fälschlicherweise als korrespondierende Linien erkannt, obwohl sie zu unterschiedlichen Teilen eines Modells gehören.
- Der Benutzer muss zur Laufzeit stets selbst die idealen Parameter für Hough und Canny finden, damit die Ergebnisse optimal sind. Eine automatische Bestimmung der idealen Parameter wäre wünschenswert. Für eine Verwendung des PoseInitializers auf mobilen Endgeräten ist es sogar essentiell, dass der Benutzer nicht selbst nach den besten Parametern suchen muss.

Die folgenden zwei Abbildungen 35 und 36 zeigen einmal eine ermittelte beste und viertbeste Pose. Sowohl die Parameter für Canny und Hough

[22] bei einer Bildgröße von jeweils 242x162

als auch die Eingabedaten für die Position und Blickrichtung waren bei beiden Bildern identisch. Jede Abbildung zeigt die HT des Renderings und des Fotos (v.l.n.r.). In Abbildung 35 fällt direkt auf, dass falsch ermittelte Korrespondenzen zu einer nicht korrekten Poseschätzung geführt haben. Die ermittelte viertbeste Pose scheint die reale Kamerapose deutlich besser zu treffen.

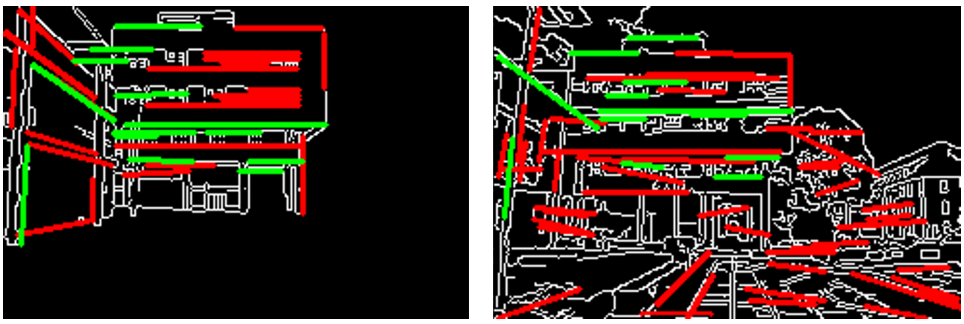

Abbildung 35: Ermittelte beste Pose (Rendering links, Foto rechts)[23]

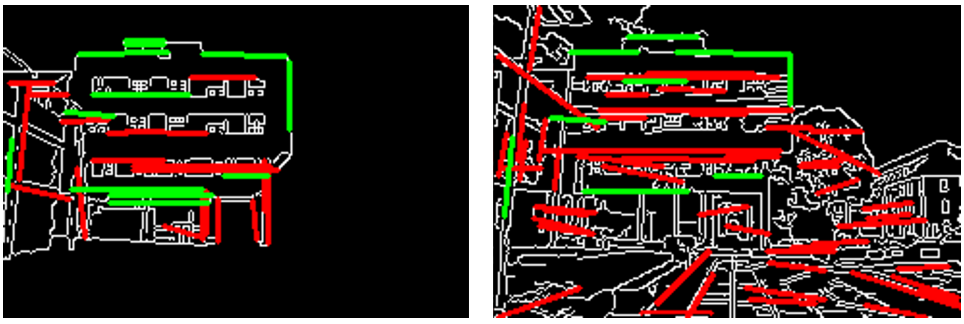

Abbildung 36: Ermittelte viertbeste Pose

Abbildung 37 zeigt die beste ermittelte Pose bei einer leicht veränderten Position des Betrachters. Im Vergleich zu den Abbildungen 35 und 36 wurde der Betrachter lediglich um 30 cm nach Norden verschoben. Die Blickrichtung sowie die Canny- und Hough-Parameter wurden nicht verändert. Auch wenn diese Verschiebung in Bezug auf die Entfernung zum beobachteten Gebäude kaum ins Gewicht fällt, so wird die Kantendetektion und somit auch das Ergebnis des Bildvergleichs stark beeinflusst.

In Abbildung 37 ist auch gut zu erkennen, dass nicht alle markanten Linien eines Gebäudes gefunden werden. Obwohl die rechte äußere Kante für das menschliche Auge eigentlich sehr gut zu sehen ist, wird sie bei einer HT als solche nicht erkannt und spielt dementsprechend beim Vergleich der Posen keine Rolle. Abbildung 38 zeigt eine Poseschätzung mit dersel-

[23] Rot: Hough-Kante, Grün: korrespondierende Hough-Kante

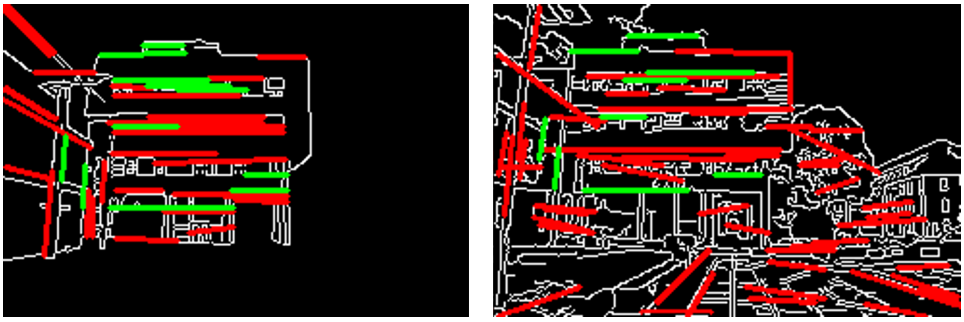

Abbildung 37: Ermittelte beste Pose bei leichter Positionsänderung

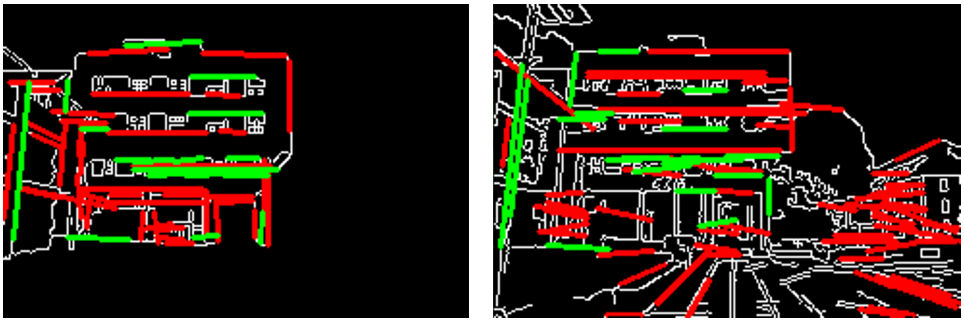

Abbildung 38: Ermittelte beste Pose bei erheblicher Veränderung der Canny- und Hough-Parameter

ben Position und Blickrichtung des Betrachters wie in Abbildung 35 und 36, allerdings mit erheblich veränderten Canny- und Hough-Parametern.

Nach einer Betrachtung dieser Probleme wird klar, dass dieses Verfahren von zu vielen Faktoren abhängig und dementsprechend für den Benutzer nicht sicher zu bedienen ist. Ein anderes Konzept zur Analyse zweier Bilder in einem urbanen Kontext wird im nächsten Abschnitt vorgestellt.

12.3 Pixelbasierter Bildvergleich

12.3.1 Idee

Hough-Transformationen und mit dem Canny-Algorithmus erstellte Kantenbilder benötigen je nach Eingabebild unter Umständen eine Anpassung der Parameter, welche die Ergebnisse der Merkmalsextraktion beeinflussen. Eine Anpassung der Parameter durch den Benutzer zur Laufzeit ist jedoch problembehaftet, besonders wenn man bedenkt, dass die Anwendung idealerweise von mobilen Geräten aus bedient werden soll.

Ein weiteres in dieser Arbeit realisiertes Konzept zum kantenbasierten Bildvergleich bedient sich der Anwendung des Sobel-Operators. Bei den Ergebnisbildern des Sobel-Algorithmus handelt es sich um 8-Bit Grau-

wertbilder, wobei ein konkreter Pixelwert die Kantenstärke an dieser Position beschreibt. OpenCV stellt eine Implementierung des Sobel-Operators in einer eigenen Methode [21] bereits zur Verfügung. Theoretisch ließen sich zwei Bilder anhand ihrer Grauwertkantenbilder miteinander pixelweise vergleichen. Um falsch kalkulierte Korrespondenzen zu reduzieren, soll zusätzlich auch die Kantenrichtung der Pixel in die Analyse einfließen. Es müssen also folgende Berechnung auf beiden Bildern durchgeführt werden.

- Anwendung des Sobel-Operators in x- und y-Richtung
- Berechnung der Kantenstärke $E(x, y)$ für jedes Pixel
- Berechnung der Kantenrichtung $\Phi(x, y)$ für jedes Pixel

Nachdem die beschriebenen Werte berechnet wurden, kann jedes Pixel des ersten Bildes mit dem entsprechenden Pixel des zweiten Bildes verglichen werden. Unklar ist noch, inwieweit dieses Verfahren performant implementiert werden kann.

12.3.2 Realisierung

Genau wie das im vorigen Teil beschriebene Konzept des Vergleichs durch eine probabilistische Hough-Transformation wurde die pixelbasierte Bildanalyse in der ImgAnalyzer-Klasse ausgelagert. Um eine bestimmte Toleranz für die Poseschätzung zu gewährleisten, wird auf das Foto vor einer Anwendung des Sobel-Operators ein *Gauß'scher Unschärfefilter* verwendet. Die durch den Sobel-Operator detektierten Kanten werden dadurch künstlich aufgebläht. Ob beim späteren Bildvergleich die eventuell korrespondierenden Pixel minimal verschoben sind, ist dadurch irrelevant.

Zuerst wird auf jedes der Bilder der Sobel-Operator in x- und y-Richtung angewandt und die Ergebnisse in zwei separaten Bildern gespeichert. Der folgende Code zeigt, wie der von OpenCV bereitgestellte Sobel-Operator auf ein Eingabebild „input“ angewandt wird. Die Filtergröße in diesem Beispiel beträgt 3x3. Zu beachten ist weiterhin, dass auf jeden Pixelwert 127 addiert wird, da der Wertebereich dieser Funktion $[-128..127]$ beträgt. Durch die Addition wird der Wertebereich der Ergebnisbilder auf einen 8-Bit Grauwertbereich von $[0..255]$ verschoben.

Listing 11: Aufruf des in OpenCV implementierten Sobel-Operators

```
1 // x-direction
2 Sobel( input, gradx, CV_8U, 1, 0, 3, 1, 127 );
3
4 // y-direction
5 Sobel( input, grady, CV_8U, 0, 1, 3, 1, 127 );
```

Die Pixelwerte der temporären Ergebnisbilder liegen also im Bereich $[0..255]$. Um nun die Kantenstärke und Kantenrichtung eines Pixels zu bestimmen, muss der Pixelwert wieder um 127 reduziert werden. Dies ist durch die Implementierung und Zwischenspeicherung der Bilder notwendig, da somit ein Verlust von Informationen verhindert wird. Die Kantenstärke $E(x, y)$ eines jeden Pixels lässt sich anhand der mit dem Sobel-Operator gefilterten Bilder „gradx“ und „grady“ bzw. $S_x(x, y)$ und $S_y(x, y)$ mit der folgenden Formel berechnen.

$$E(x, y) = \sqrt{(S_x(x, y))^2 + (S_y(x, y))^2} \tag{36}$$

Die Kantenrichtung $\Phi(x, y)$ lässt sich wie folgt bestimmen.

$$\Phi(x, y) = \arctan\left(\frac{S_y(x, y)}{S_x(x, y)}\right) \cdot \frac{255}{\pi} \tag{37}$$

Die Multiplikation mit $\frac{255}{\pi}$ ist notwendig, damit die Kantenrichtungswerte beim späteren Vergleich nicht den Wertebereich $[0..1]$ aufweisen, sondern den Wertebereich eines 8-Bit Grauwertbildes voll ausschöpfen. Abbildung 39 zeigt das Kantenrichtungsbild eines Kreises. Pixelwerte im Bereich $[0..255]$ bilden also Winkel im Bereich $[0..179]$ ab.

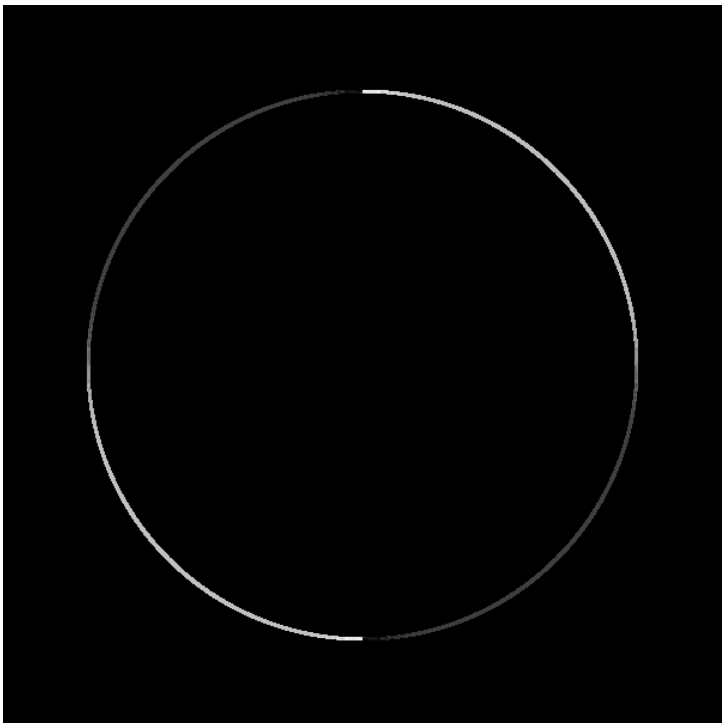

Abbildung 39: Kantenrichtungsbild eines Kreises

Die Werte für die Kantenstärke und Kantenrichtung werden in temporären Bildern zwischengespeichert. Pixel, die in zwei verschiedenen Eingabebildern an der selben Position liegen, können also als passend zueinander bezeichnet werden, sofern sie sowohl ähnliche Kantenstärken als auch Kantenrichtungen besitzen. Die Implementierung dieses Verfahrens wird im PoseInitializer zu einem einzigen Ähnlichkeitswert pro Pixel zusammengefasst. Der folgende Code konkretisiert diesen Ansatz.

Listing 12: Bildvergleich über die Kantenstärke und Kantenrichtung pro Pixel

```
1  ...
2
3  // both pixels have to be bright
4  edgeMatch = (float)edgeIntensityR/255.0 *
5              (float)edgeIntensityP/255.0;
6
7  // calculate the difference between the two directions
8  directionDifference = edgeDirectionR - edgeDirectionP;
9
10 // some special-cases
11 if (directionDifference < -127)
12  directionDifference += 255;
13 if (directionDifference > 127)
14  directionDifference -= 255;
15
16 directionDifference = abs(directionDifference);
17
18 // check the direction difference
19 if (directionDifference > 8)
20  {
21   // difference too big -> no match
22   directionMatch = 0;
23  }
24  else
25  {
26  // difference small enough
27   directionMatch = 1.0 - (directionDifference / 8.0);
28  }
29
30 // add the match values of all pixels
31 matchValue += edgeMatch * directionMatch;
32 ...
```

Die Ähnlichkeit zweier Pixel hängt also sowohl von deren Kantenstärke als auch Kantenrichtung ab. Für zwei zu vergleichende Pixel werden zuerst die Kantenstärken an der entsprechenden Position zu einem einzigen Wert verrechnet. Je größer beide Kantenstärken sind, desto größer wird dieser Wert, dessen möglicher Wertebereich $[0..1]$ ist. Im Anschluss wird die Differenz der Kantenrichtungen an besagter Position berechnet. Ist die Differenz der Kantenrichtungen zu groß, so fließt der kombinierte Kantenstärkewert in das Ähnlichkeitsmaß der beiden Eingabebilder nicht ein. Unterschreitet die Differenz der Kantenrichtungen eine definierte Schwelle, so fließt die Ähnlichkeit der Kantenstärke an dieser Position umso stärker in die gesamte Ähnlichkeitsbestimmung ein, je kleiner die Differenz der Kantenrichtungen ist. Die Ähnlichkeit zweier Eingabebilder wird folglich umso größer, je mehr zueinander passende Pixel gefunden werden.

Unter Berücksichtigung einer möglichst performanten Implementierung fällt auf, dass die Werte der beiden mit dem Sobel-Operator in x- und y-Richtung erstellten Bilder den begrenzten Wertebereich von $[0..255]$ haben. Dadurch bietet sich die Möglichkeit, alle Berechnung zur Bestimmung der Kantenstärke und Kantenrichtung eines Pixels im Vorfeld durchzuführen, sodass später nur auf die in einer Lookup-Tabelle gespeicherten Werte zugegriffen werden muss. Für die Berechnung dieser Werte sind zwei Parameter nötig, wodurch 256x256 mögliche Kombinationen für alle möglichen Eingaben existieren. Alle Ergebniswerte werden jeweils in einem eigenen 8-Bit Grauwertbild gespeichert. Die Position in diesem vorberechneten Ergebnisbild entspricht dabei den Eingabeparametern für die Berechnung. Somit enthält beispielsweise das vorberechnete Kantenstärkebild an der Position (x_0, y_0) das Ergebnis der Berechnung $\sqrt{(x_0)^2 + (y_0)^2}$. Analog dazu verhält es sich mit dem Zugriff auf die vorberechneten Ergebnisse für die Kantenrichtung. Beide Vorberechnungen werden unmittelbar nach dem Programmstart durch ein Precalculator-Objekt durchgeführt und in entsprechenden Grauwertbildern gespeichert. Abbildung 40 visualisiert die durch den Precalculator erstellten Ergebnismatrizen.

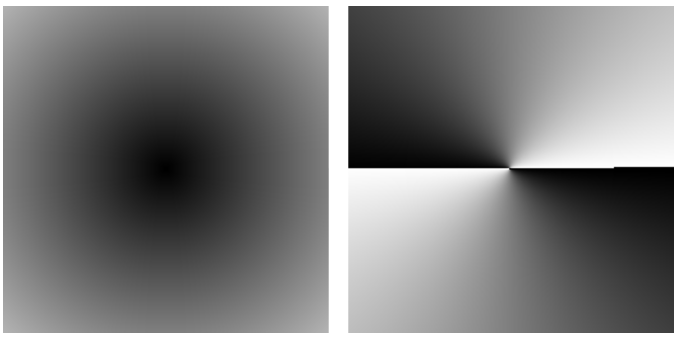

Abbildung 40: vorberechnete Kantenstärken und Kantenrichtungen für 8-Bit Grauwertbilder (v.l.n.r)

12.3.3 Ergebnisse

Für einen pixelbasierten Bildvergleich auf Kantenstärke- und Kantenrichtungsbildern ist es nicht notwendig, dass der Benutzer zur Laufzeit eine Manipulation der Parameter vornimmt, was einen erheblichen Vorteil gegenüber der auf einer Hough-Transformation basierenden Bildanalyse darstellt. Auch die Ergebnisse der Poseschätzung scheinen deutlich robuster zu sein. Die folgende Bildserie (Abbildungen 41, 42, 43, 44) wurde mit denselben Angaben einer Position und Blickrichtung wie bei den Abbildungen 35, 36 und 38 erstellt. Es fällt auf, dass die ersten drei Posen nahezu identisch sind. Erst die viertbeste ermittelte Pose stellt eine unpräzise Rekonstruktion der realen Kamerapose dar.

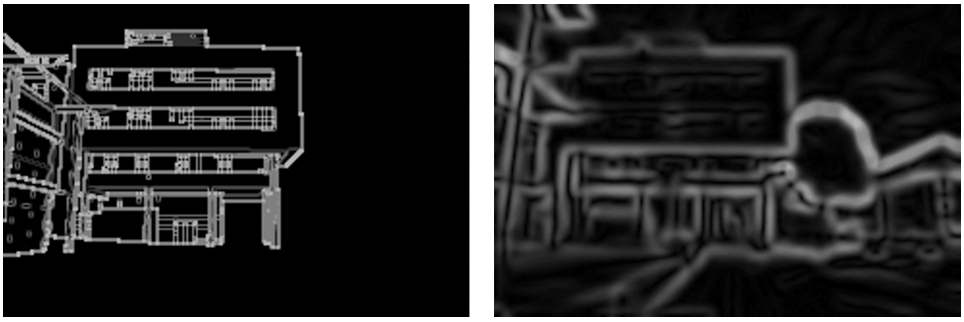

Abbildung 41: Ermittelte beste Pose (Rendering links, Foto rechts)

Abbildung 42: Ermittelte zweitbeste Pose

Abbildung 43: Ermittelte drittbeste Pose

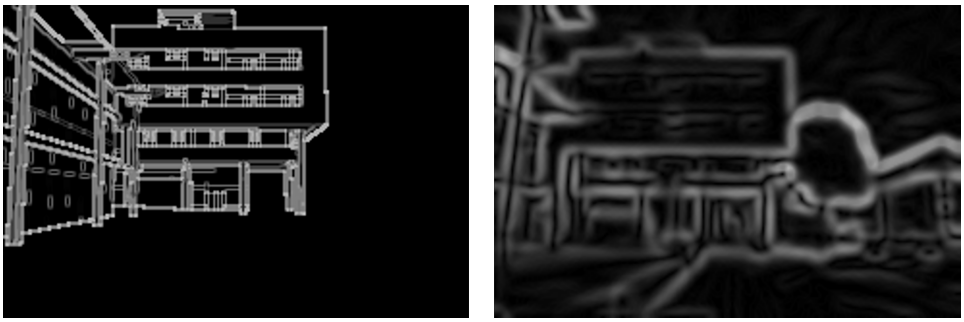

Abbildung 44: Ermittelte viertbeste Pose

12.4 Fazit

Schon nach wenigen Tests mit verschiedenen Fotos, die zur Evaluation des Bildvergleichs erstellt wurden, hat sich gezeigt, dass der pixelbasierte Bildvergleich eine Reihe von Vorteilen gegenüber der Analyse, die auf dem Vergleich von Hough-Kanten basiert, mit sich bringt. Die Vorteile können dabei folgendermaßen zusammengefasst werden.

- Eine Poseschätzung basierend auf durch den Sobel-Algorithmus erstellten Kantenstärke- und Kantenrichtungsbildern stellt sich als äußerst robust gegenüber leichten Änderungen der realen Kamerapose heraus. Sowohl Kantenrichtungen als auch Kantenstärken werden durch geringe Translationen oder Rotationen der realen Kamera kaum beeinflusst. Eine probabilistische Hough-Transformation hingegen verpasst schon bei kleinsten Veränderungen der realen Kamerapose eindeutige Kanten. Womöglich könnte dieses Problem durch eine nicht so starke Verkleinerung[24] der Vergleichsbilder behoben werden. Dies hätte jedoch starke Einbrüche der Performanz des Bildvergleichs zur Folge.

[24]Die Auflösung des Renderings und des Fotos ist standardmäßig 968x648. Die Vergleichsbilder werden auf eine Auflösung von 242x162 herunterskaliert.

- Zur Erstellung der Kantenstärke- und Kantenrichtungsbilder sind zur Laufzeit im Gegensatz zur Hough-Transformation keine Anpassungen der Parameter nötig. Lediglich die Glättung des Fotos der realen Szene und die Größe des Sobel-Filterkerns können beeinflusst werden. Es hat sich herausgestellt, dass die vorab konfigurierten Parameter für nahezu alle Bilder des Testszenarios gut funktionieren.
- Auch wenn ein pixelbasierter Vergleich anfangs den Anschein eines nicht performanten Verfahrens erweckt, so zeigt sich, dass dieses Verfahren im Gegensatz zu einer auf der Hough-Transformation basierenden Analyse bis zu zweimal so schnell durchgeführt werden kann. Die Spezifikationen des verwendeten Testsystems werden im weiteren Verlauf dieser Arbeit beschrieben.
- Die Erstellung der Kantenstärke- und Kantenrichtungsbilder lässt sich durch eine Reihe von Vorberechnungen sehr effizient beschleunigen, ohne dass es dabei zu einer Verringerung der Präzision einer Poseschätzung kommt.

13 Systematischer Ablauf einer Poseschätzung

Bevor die Ergebnisse der Implementierung erörtert werden, soll an dieser Stelle zusammenfassend der vollständige Ablauf einer Poseschätzung beschrieben werden. Abbildung 45 stellt den systematischen Ablauf in einem UML-Aktivitätsdiagramm dar. Zu Beginn werden durch den Benutzer über die GUI eine GPS-Position und eine Kompassrichtung vorgegeben. Dazu passend wird ein Foto geladen, welches zu den eingegebenen Daten korrespondiert. Anhand dieser Daten wird eine erste grobe Poseschätzung der realen Kamera durchgeführt. Diese Pose dient als Referenzpose zur Erstellung weiterer Posen. Nachdem die für den späteren Bildvergleich essentiellen Posen erstellt wurden, müssen die entsprechenden Modelle anhand der groben Pose gesucht und geladen werden. Dazu werden pro Modell einige Kontrollpunkte benötigt. Im Anschluss wird jede der generierten Posen einmal auf die virtuelle Kamera angewandt und die Szene von dieser Perspektive aus in eine Textur gerendert. Das eben erstellte Bild der virtuellen Szene wird durch Anwendung des Sobel-Operators oder einer Hough-Transformation auf einige wichtige Merkmale reduziert. Derselbe Algorithmus wird auf das Referenzfoto angewandt. Beide Bilder werden nun miteinander verglichen. Zu jeder verwendeten Pose wird der ermittelte Ähnlichkeitswert gespeichert. Danach werden die Pose-Objekte nach ihrer Ähnlichkeit zur realen Kamerapose sortiert und die beste ermittelte Pose wird angezeigt. Der Benutzer kann nun manuell auch eine andere Pose aus der sortierten Liste aller Pose-Objekte laden und betrachten.

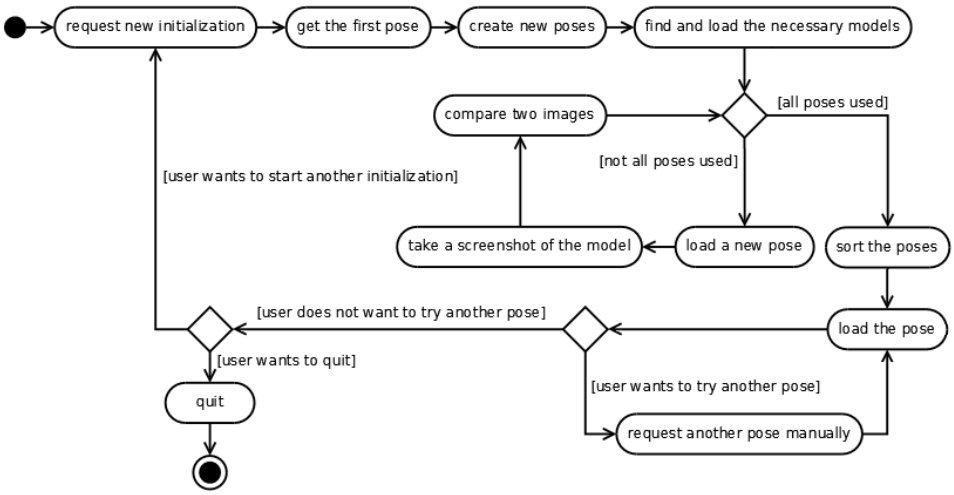

Abbildung 45: Vollständiger Ablauf einer Poseschätzung als UML-Aktivitätsdiagramm

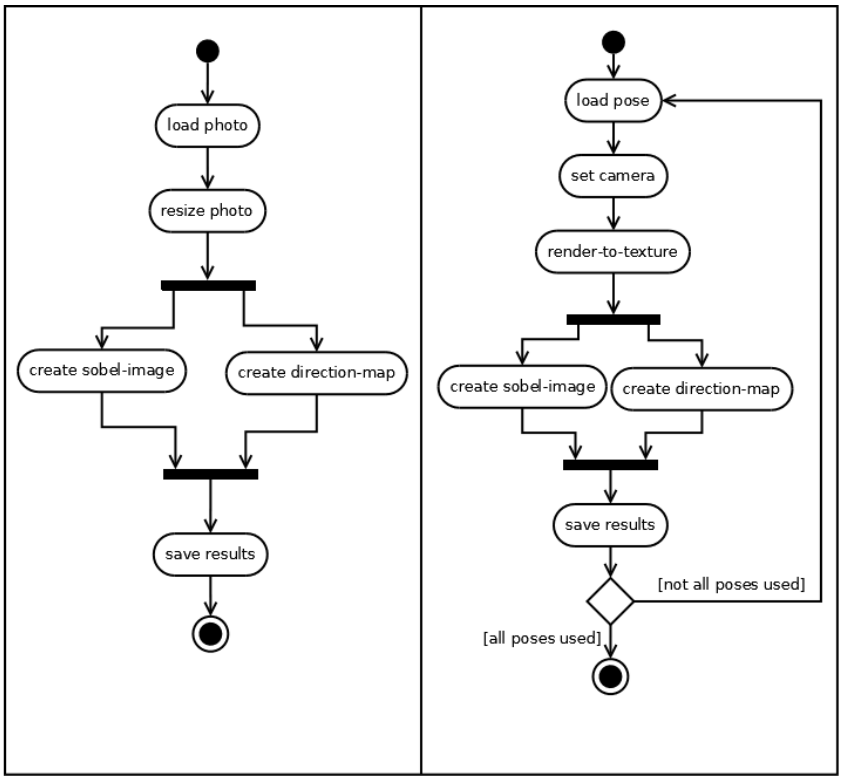

Abbildung 46: Laden und Erstellen der Vergleichsbilder mit dem pixelbasierten Ansatz

Abbildung 46 verdeutlicht noch einmal das Laden und automatische Erstellen der Vergleichsbilder mit dem pixelbasierten Ansatz. Die mit einer realen Kamera erstellten Aufnahmen werden dabei über die GUI geladen. Unmittelbar danach wird das Foto auf eine definierte Größe heruntersklaliert und die Bildvorverarbeitung durchgeführt. Die Ergebnisse dieser Vorverarbeitung bleiben so lange erhalten und können zur Analyse herangezogen werden, bis ein neues Foto manuell geladen wird. Screenshots bzw. Kantenbilder der virtuellen Szene werden hingegen unmittelbar nach dem Rendern aus einer neu gesetzten Kamerapose heraus automatisch erzeugt. Nachdem zwei Bilder miteinander verglichen wurden, wird eine neue Pose gesetzt. Dieser Vorgang wird so lange wiederholt, bis alle Posen einmal getestet wurden.

Teil IV
Ergebnisse

14 Funktionen

Nachdem die Implementierung der beschriebenen Konzepte ausführlich dargelegt wurde, können im folgenden Verlauf die Ergebnisse betrachtet werden. Abbildung 47 zeigt die grafische Benutzeroberfläche des PoseInitializers. Ein Benutzer kann die folgenden Aktionen über die GUI des PoseInitializers durchführen.

- Laden eines Fotos der realen Szene
- Anzeige oder Ausblendung der manuell vergebenen GPS-Kontrollpunkte
- De- und Aktivierung des automatischen Posestreuens zur Verfeinerung der Poseschätzung[25]
- Anzeige oder Ausblendung der Debug-Fenster, welche die vorverarbeiteten Vergleichsbilder visualisieren
- Eingabe der GPS-Koordinaten im UTM-Format
- Eingabe der Kompassrichtung
- Starten der Poseschätzung unter Berücksichtigung der zuvor eingestellten Parameter
- Beenden des PoseInitializers

Des Weiteren stehen dem Benutzer diese Tastenkürzel zur Verfügung.

- Esc: Beenden
- O: Anzeige des Statistikfensters (Rendergeschwindigkeit, etc.)
- Enter, Return: Starten einer Poseschätzung
- L: Laden eines neuen Fotos
- +: manuelles Laden der nächsten Pose
- -: manuelles Laden der vorigen Pose

[25]Sollte das automatische Posestreuen deaktiviert werden, so wird lediglich eine sehr grobe Poseschätzung anhand der GPS- und Kompassdaten durchgeführt.

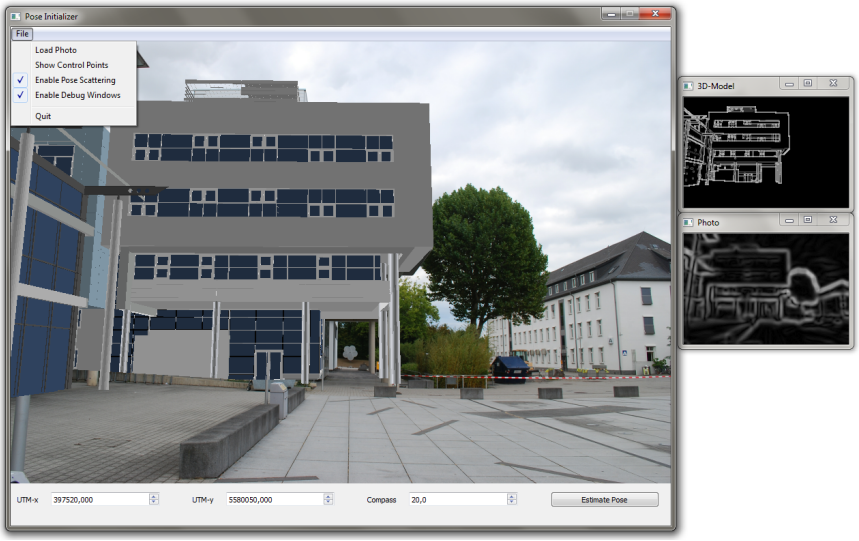

Abbildung 47: Grafische Benutzeroberfläche des PoseInitializers

15 Performanz

Zur Evaluation des PoseInitializers wurde das folgende System verwendet.

Kategorie	verbaute Hardware
Prozessor	AMD Phenom II X4 965 (4 x 3,40 GHz)
Arbeitsspeicher	2 x 2 GB 1333 MHz Corsair XMS3
Grafikkarte	ATI Radeon HD 5850 1 GB
Speichermedium	S-ATA Festplatte 7200 U/min

In der aktuellen Version des PoseInitializers werden mit einer durch GPS und Kompass grob geschätzten Pose 676 neue Posen erstellt, aus denen jeweils ein Bildvergleich zwischen realer und virtueller Szene resultiert. Die Dauer zur Schätzung einer Pose hängt nicht nur von der Anzahl der verstreuten Posen ab, sondern auch von der Anzahl und Komplexität der verwendeten Modelle.

Optimierungen bei der Implementierung des pixelbasierten Verfahrens, welches sich als robuster und präziser herausstellte, konnten ohne Präzisionsverlust eine signifikante Steigerung der Performanz bewirken. Je nach aktuell angezeigtem 3D-Modell sind mit dem verwendeten Testsystem bis zu 200 Bildvergleiche pro Sekunde möglich[26]. Wenn zusätzliche Gebäude in Abhängigkeit der GPS-Position und Blickrichtung geladen werden, so sinkt die Geschwindigkeit bei vier angezeigten Gebäuden auf ca. 150 Bildvergleiche pro Sekunde[27]. Selbst unter diesen Umständen ist die initiale

[26] hier: Gebäude G des Campus Koblenz
[27] hier: Gebäude EF, C (inkl. Bibliothek), G und D

Poseschätzung in einem Zeitraum zwischen vier und fünf Sekunden vollständig durchgeführt. Der nächste Schritt zur Optimierung mit der hier verwendeten Testszene ist die Bearbeitung des Campusmodells selbst, was jedoch den zeitlichen Rahmen dieser Arbeit überschritten hätte. Eine Reduzierung des Detailgrades der einzelnen Gebäude wäre problemlos möglich, da die Analyse der Bilder nur auf markanten Kantenzügen arbeitet. Die Modellierung perfekter Rundungen von Säulen beispielsweise, wie sie im Campusmodell realisiert wurden, bedeuten lediglich eine unnötig hohe Zahl von Polygonen, ohne dass die hier verwendete Bildanalyse davon profitieren könnte.

16 Präzision

Im Folgenden sollen einige, zu Beginn dieser Arbeit erstellte Fotografien Aufschluss über die Präzision einer Poseschätzung liefern. Diese Referenzfotos wurden zusammen mit ihrer aktuellen GPS-Position und Kompassrichtung am 20.09.2011 auf dem Campus Koblenz erstellt. Anhand des Fotos und dazu korrespondierender GPS- und Kompassdaten soll die reale Kamerapose geschätzt werden.

Abbildung 48: EF-Gebäude (GPS-X: 397420; GPS-Y: 5579969; Kompass: 63°)

Abbildung 49: G-Gebäude links, Bibliothek rechts (GPS-X: 397495; GPS-Y: 5580105; Kompass: 92°)

Abbildung 50: Sporthalle links (GPS-X: 397440; GPS-Y: 5580090; Kompass: 20°)

Abbildung 51: B-Gebäude links, EF-Gebäude rechts (GPS-X: 397514; GPS-Y: 5580014; Kompass: 200°)

Abbildung 52: EF-Gebäude links, Bibliothek rechts (GPS-X: 397524; GPS-Y: 5580043; Kompass: 245°)

Abbildung 53: neue Mensa links, B-Gebäude rechts (GPS-X: 397525; GPS-Y: 5580013; Kompass: 160°)

Abbildung 54: EF-Gebäude (GPS-X: 397512; GPS-Y: 5579990; Kompass: 258°)

Anhand dieser Beispiele lässt sich eine recht präzise Funktionsweise des PoseInitializers nachweisen, sofern der pixelbasierte Bildvergleich gewählt wird. Teilweise ist die tatsächliche Blickrichtung leicht abweichend von dem über den Kompass ausgelesenen Wert. Bewegt sich diese Abweichung in einem moderaten Bereich, so kann sie vernachlässigt werden, da eine Glättung des Referenzfotos vor einer Anwendung des Sobel-Operators eine gewisse Toleranz beim Bildvergleich garantiert. Abweichungen von der tatsächlichen Kompassrichtung lassen sich ebenfalls umgehen, indem beim Posestreuen auch Rotationen um die Hochachse berücksichtigt werden. Eine Poseschätzung wäre unter Umständen zwar präziser, durch die zusätzlichen Bildvergleiche würde die Performanz jedoch negativ beeinflusst werden.

Es muss auch erwähnt werden, dass eine fehlerfreie Funktionsweise nur möglich ist, wenn eine per GPS gemessene Position von ihrer tatsächlichen Position nicht zu stark abweicht. Ist die GPS-Ortung zu unpräzise, so kann der Bildvergleich kein passendes Ergebnis mehr finden. Dieses Problem könnte man wieder auf Kosten der Performanz durch einen größeren Bereich des Posestreuens ausgleichen.

Das nicht vollständig korrekte Modell der realen Szene beeinflusst das Verfahren ebenfalls negativ. Abbildung 55 zeigt das EF-Gebäude. Es fällt auf, dass einige Fenster im Modell fehlen. Eine zuverlässige Poseschätzung wird dadurch erschwert.

Abbildung 55: EF-Gebäude mit fehlenden Fenstern (GPS-X: 397414; GPS-Y: 5579949; Kompass: 22°)

Teil V
Zusammenfassung und Ausblick

17 Zusammenfassung

Diese Arbeit hat gezeigt, dass eine initiale Poseschätzung mit *Analyse durch Synthese* durchaus umsetzbar ist. Für urbane Umgebungen kann eine Schätzung der realen Kamerapose durch die Kombination aus GPS- und Kompassdaten sowie einem Einzelbild der Szene zu einem bestimmten Zeitpunkt berechnet werden.

Durch eine Verwendung der GPS-Koordinaten im UTM-Format und einer Interpretation der Kompassrichtung als Rotation um die Hochachse lassen sich die zu einem konkreten Zeitpunkt gemessene Position und Blickrichtung durch eine einfache Transformation in die lokalen Koordinaten des virtuellen Modells der entsprechenden realen Szene umrechnen. Alle in der realen Szene relevanten Objekte müssen in der virtuellen Szene maßstabsgetreu und korrekt positioniert bzw. ausgerichtet sein. Des Weiteren müssen die Geländeeigenschaften der realen Szene im virtuellen Modell enthalten sein, sodass eine zuverlässige Berechnung der Betrachterhöhe ermöglicht wird. Ein Konzept zur Verwaltung von Metadaten, welche mit einzelnen virtuellen Objekten verknüpft werden, wurde durch die Erstellung und dem automatischen Auslesen einer XML-Datei vorgestellt.

Für jedes virtuelle Objekt wurden manuell einige Kontrollpunkte definiert. So lassen sich zur Laufzeit stets diejenigen Objekte herausfiltern, die unter Berücksichtigung der aktuellen Position und Blickrichtung relevant sind. Die GPS-Positionen der besagten Kontrollpunkte werden ebenfalls als Metadaten eines entsprechenden Modells gespeichert. Durch die Filterung der zu ladenden Objekte können beliebig große Umgebungen mit beliebig vielen Objekten virtuell modelliert und verwendet werden. Eine effiziente Verwaltung der benötigten Ressourcen ermöglicht das dynamische Laden beliebig vieler Modelle, wobei diese Modelle nur unter den besagten Voraussetzungen in den Speicher des Systems geladen werden.

Es hat sich weiterhin herausgestellt, dass die reine Poseschätzung basierend auf einer GPS-Position und einer Kompassrichtung nicht sonderlich präzise ist. Störungen der GPS-Empfänger tragen schnell zu einer falschen Positionsbestimmung bei. Für die Verfeinerung der Poseschätzung durch optisches markerloses Tracking wurden zwei Ansätze vorgestellt. Beide Konzepte machen sich dabei das Vorhandensein vieler markanter Linien in urbanen Umgebungen zunutze. Um die Ähnlichkeit zwischen einer realen und einer virtuellen Kamerapose zu bestimmen, wurde eine Vielzahl von Posen basierend auf der ersten groben Schätzung synthetisch generiert und die daraus gerenderten Bilder mit einem Foto der realen Szene verglichen.

Ein erster Ansatz war der auf einer Hough-Transformation basierende Bildvergleich. Sowohl auf die Aufnahmen der virtuellen Kamera als auch auf das Foto der realen Szene wurde die in OpenCV implementierte probabilistische Hough-Transformation durchgeführt. Das Ergebnis waren zwei Listen von Kanten, die durch ihren Start- und Endpunkt definiert sind. Mit diesen Informationen ließen sich weitere Eigenschaften der Kanten, wie beispielsweise der Abstand zum Bildursprung oder die Ausrichtung der Kante, berechnen. Durch den Vergleich aller gefundenen Kanten im Foto mit allen Kanten im synthetisch erzeugten Bild sollte ein Ähnlichkeitsmaß zwischen den beiden Bildern ermittelt werden. Die Ergebnisse überzeugten dabei leider nicht sonderlich, da eine robuste Detektion der Kanten nicht zuverlässig durch die von OpenCV bereitgestellte probabilistische Hough-Transformation gewährleistet werden konnte.

Der zweite Ansatz zur Definition eines Ähnlichkeitsmaßes führte über einen pixelweisen Vergleich der beiden besagten Bilder. Dabei ermöglichte die Anwendung des Sobel-Operators eine Berechnung der Kantenstärke und Kantenrichtung für jedes Pixel. Jedes Pixel im Bild der virtuellen Szene wurde mit dem entsprechenden Pixel an derselben Position im Bild der realen Szene verglichen. Durch künstlich erzeugte Unschärfe im Referenzfoto wurde eine gewisse Toleranz gegenüber einer leichten Deckungsungenauigkeit zwischen den zu vergleichenden Bildern gewährleistet. Mit dem Konzept des regelmäßigen Posestreuens wurde in den meisten Fällen eine gute Rekonstruktion der realen Kamerapose ermöglicht. Voraussetzung dafür ist unter anderem die nicht zu starke Abweichung der per GPS gemessenen Position von den tatsächlichen GPS-Koordinaten und ein adäquates Modell der realen Szene. Nicht nur die Präzision des pixelbasierten Verfahrens konnten überzeugen. Die Möglichkeit viele Berechnungen nicht zur Laufzeit durchführen zu müssen, erlaubte weitreichende Optimierungsmaßnahmen zur Steigerung der Performanz.

Die iterative Verteilung synthetischer Kameraposen stellte sich mit beiden vorgestellten Bildvergleichen als nicht praktikabel heraus, da eine Poseschätzung sehr nah an der gesuchten Pose liegen muss, damit ein hohes Maß der Ähnlichkeit ermittelt wird.

18 Ausblick

18.1 Weiterentwicklung

Der in dieser Arbeit implementierte PoseInitializer ist durch Angabe von GPS- und Kompassdaten sowie einem Einzelbild der realen Szene in der Lage, eine Schätzung der realen Kamerapose zu bestimmen. Sowohl das manuelle Eingeben der Daten über eine grafische Benutzeroberfläche als auch das Laden eines Einzelbildes durch den Benutzer selbst ist dabei für

eine Anwendung mit mobilen Endgeräten in urbanen Umgebungen ungeeignet. Die nächste Entwicklungsstufe des PoseInitializers wäre eine Übermittlung dieser Daten zur Laufzeit an einen stationären Rechner. Als Übertragungsmedium wären drahtlose Netzwerke durchaus vorstellbar. Ein mobiles Endgerät, wie beispielsweise ein modernes Smartphone, könnte über den integrierten GPS-Empfänger und elektronischen Kompass die Positionsdaten über das drahtlose Netzwerk zu einem zentralen Rechner übertragen. Gleichzeitig könnte man mit der integrierten Kamera des mobilen Endgerätes Videoaufnahmen der realen Szene erstellen. Somit wären alle nötigen Daten für eine initiale Poseschätzung vorhanden. Entsprechende Applikationen, welche die besagten Daten gleichzeitig ermitteln und an einen zentralen Rechner weitergeben können, sind noch zu entwickeln.

Ein pixelbasierter Bildvergleich in Kombination mit regelmäßigem Posestreuen stellte sich bereits als vielversprechender und performanter Ansatz heraus. Zur Bestimmung einer initialen Pose muss der Bereich, in dem weitere Posen erzeugt werden, relativ groß gewählt sein, damit Ungenauigkeiten des GPS-Systems ausgeglichen werden können. Ist eine erste Poseschätzung erfolgreich durchgeführt, so ist das weitere Tracking mit einer bewegten realen Kamera durchaus vorstellbar. Der Bereich des Posestreuens müsste nach einer erfolgreichen Initialisierung, also der Berechnung dieser ersten Pose, einfach sehr viel kleiner gewählt werden. Dies hätte eine erhebliche Reduzierung zu generierender Posen und somit auch sehr viel weniger benötigte Bildvergleiche zur Folge. Eine ruckelfreie Rekonstruktion der Kamerapose wäre mit dem hier verwendeten Testsystem zwar nicht unbedingt möglich, bei stärkeren Rechnern allerdings durchaus denkbar.

18.2 Zusätzliche Optimierung

Die Implementierung des PoseInitializers ist stark auf ein performantes Verhalten ausgelegt. Doch auch hier können weitere Verbesserungen vorgenommen werden. Zur Bildanalyse wäre eine Verwendung von Shadern ideal, sodass eine Berechnung der Bildähnlichkeiten auf einer im Vergleich zu einem Prozessor sehr viel schnelleren Grafikkarte ermöglicht wird. Für OpenCV wird zwar die Auslagerung einiger Algorithmen auf die Grafikkarte angeboten, dazu muss jedoch eine CUDA fähige Grafikeinheit vorhanden sein. Diese Bedingung wird nur von NVidia-Grafikkarten erfüllt. Da das zur Verfügung stehende Testsystem den Grafikchip eines anderen Herstellers nutzt, konnte diese Optimierung nicht getestet werden. Die Erstellung eigener Shader wurde aus zeitlichen Gründen im Rahmen dieser Arbeit nicht in Betracht gezogen.

Anhang

A Symbole und Bezeichner

Symbol	Bedeutung
$I(x, y)$	Bildfunktion für jede Position (x, y)
$E(x, y)$	Kantenstärkefunktion für jede Position (x, y)
$\Phi(x, y)$	Kantenrichtungsfunktion für jede Position (x, y)
$S_x(x, y), S_y(x, y)$	Ergebnis des Sobel-Operators in x- bzw. y-Richtung für jede Position (x, y)
T_x, T_y	Translation in x- bzw. y-Richtung
R_{GPS}, R_{OGRE}	Rotation um die Hochachse im GPS- bzw. OGRE-Koordinatensystem
$R_{GPS \to OGRE}$	Rotationsfaktor für eine Transformation vom GPS- ins OGRE-Koordinatensystem
T_{TR}	Transformation bestehend aus Translation und Rotation vom GPS- ins OGRE-Koordinatensystem
O_x, O_y	GPS-Koordinaten des definierten OGRE-Koordinatensystems
P_x, P_y	Position eines Betrachters in GPS-Koordinaten
P_{GPS}, P_{OGRE}	Position eines Betrachters in GPS- bzw. OGRE-Koordinaten
H_x^D, H_y^D	Koeffizientenmatrix zur partiellen Ableitung in x- bzw. y-Richtung
H_x^S, H_y^S	Filterkern zur Anwendung des Sobel-Operators in x- bzw. y-Richtung
ω	definierter Winkel des Suchfeldes zur Validierung von Kontrollpunkten
d_{min}, d_{max}	minimale und maximale erlaubte Entfernung von der Position des Betrachters zu einem Kontrollpunkt
$d(a, b)$	Entfernung zwischen den Punkten a und b
K	Kontrollpunkt
N	Nordpol
$\overrightarrow{P_{GPS}N}$	Vektor von der GPS-Position eines Betrachters zum Nordpol (entspricht der Kompassrichtung 0°)
l	Länge einer Linie
ρ	Entfernung einer Linie zum Ursprung des Bildkoordinatensystems
θ	Winkel zwischen einer Linie und der x-Achse des Bildkoordinatensystems

Symbol	Bedeutung
L	Linie im Bildraum
M	Linie im Parameterraum (auch Hough-Raum)
C_H, C_L	hoher und niedriger Schwellwert des Canny-Operators
σ	Standardabweichung zur Glättung eines Bildes vor Anwendung des Canny-Operators
HT	Abkürzung für „Hough-Transformation“

Literatur

[1] BLENDER FOUNDATION: *Blender*. Website. http://www.blender.org/ besucht am Stand 28.12.2011.

[2] C.G. BELL ET AL.: *Reduction of Speech Spectra by Analysis-by-Synthesis Techniques*. In: *The Journal of the Acoustical Society of America*, Band Vol. 33, 1961.

[3] CHARTCROSS: *GPSTest*. Website. https://market.android.com/details?id=com.chartcross.gpstest besucht am 27.12.2011.

[4] D. DEMENTHON, L. DAVIS: *Model-Based Object Pose in 25 Lines of Code*. In: *Proceedings of the Second European Conference on Computer Vision*, 1992.

[5] G. BRADSKI, A. KAEHLER: *Learning OpenCV*. O'Reilly Media Inc., 2008.

[6] G. BRADSKI ET AL.: *OpenCV - Open Source Computer Vision*. Website. http://opencv.willowgarage.com/wiki/Welcome besucht am 18.12.2011.

[7] G. JUNKER: *Pro OGRE 3D Programming*. Apress, 2006.

[8] G. XU: *GPS - Theory, Algorithms and Applications (2nd Edition)*. Springer, GeoForschungszentrum Potsdam, 2007. Kapitel 1.

[9] H. KATO, HUMAN INTERFACE TECHNOLOGY LABORATORY UNIVERSITY OF WASHINGTON, P.LAMB: *ARToolKit*. Website. http://www.hitl.washington.edu/artoolkit/ besucht am 12.12.2011.

[10] L. H. ROHWEDDER: *Transversale Mercator-Projektion für den Nullmeridian*. Wikipedia-Image. http://de.wikipedia.org/w/index.php?title=Datei:Transversal_Mercator_0.jpg&filetimestamp=20060820160459 besucht am 14.12.2011.

[11] L. THOMASON: *TinyXML*. Website. http://www.grinninglizard.com/tinyxml/ besucht am 04.01.2011.

[12] LANDESAMT FÜR VERMESSUNG UND GEOBASISINFORMATION RHEINLAND-PFALZ: *Geoportal Rheinland-Pfalz*. Website. http://www.geoportal.rlp.de/ besucht am 30.12.2011.

[13] LANDESAMT FÜR VERMESSUNG UND GEOBASISINFORMATION RHEINLAND-PFALZ: *Geoportal Rheinland-Pfalz Screenshots*. Website. http://www.geoportal.rlp.de/ ©GeoBasis-DE/LVermGeoRP2012-01-02.

[14] M. REIMPELL: *Blender Exporter*. Website. http://www.ogre3d.org/tikiwiki/Blender+Exporter besucht am 31.12.2011.

[15] N. EFFORD: *Digital Image Processing - A Practical Introduction Using Java*. Addison-Wesley, 2000.

[16] NINTENDO. Website. http://www.nintendo.de/NOE/de_DE/wii_54.html besucht am 12.12.2011.

[17] NOKIA CORPORATION: *Qt SDK*. Website. http://qt.nokia.com/products/ besucht am 27.12.2011.

[18] OGRE3D-COMMUNITY: *Ogre3D*. Website. http://www.ogre3d.org/ besucht am 27.12.2011.

[19] OGRE3D-COMMUNITY: *Raycasting to the polygon level*. Website. http://www.ogre3d.org/tikiwiki/Raycasting+to+the+polygon+level besucht am 02.01.2011.

[20] OPENCV-COMMUNITY: *OpenCV 2.3 Documentation - HoughLinesP*. Website. http://opencv.itseez.com/modules/imgproc/doc/feature_detection.html#houghlinesp besucht am 08.01.2011.

[21] OPENCV-COMMUNITY: *OpenCV 2.3 Documentation - Sobel*. Website. http://opencv.itseez.com/modules/imgproc/doc/filtering.html#sobel besucht am 11.01.2011.

[22] P. S. FORESMAN: *Längenkreise, Breitenkreise*. Wikipedia-Image. http://de.wikipedia.org/w/index.php?title=Datei:Longitude_(PSF).png&filetimestamp=20071204205703 und http://de.wikipedia.org/w/index.php?title=Datei:Latitude_(PSF).png&filetimestamp=20071206184415 besucht am 14.12.2011.

[23] PROFESSUR FÜR GEODÄSIE UND GEOINFORMATIK UNIVERSITÄT ROSTOCK: *ETRS'89*. Website. http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=655 besucht am 14.12.2011.

[24] PROFESSUR FÜR GEODÄSIE UND GEOINFORMATIK UNIVERSITÄT ROSTOCK: *Gauß-Krüger-Projektion/-Koordinaten*. Website. http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=743 und http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=29261976 besucht am 14.12.2011.

[25] PROFESSUR FÜR GEODÄSIE UND GEOINFORMATIK UNIVERSITÄT ROSTOCK: *Global Positioning System (GPS)*. Website. http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=427 besucht am 14.12.2011.

[26] PROFESSUR FÜR GEODÄSIE UND GEOINFORMATIK UNIVERSITÄT ROSTOCK: *Längengrad, Breitenkreise*. Website. http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=1097 und http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=984843496 besucht am 14.12.2011.

[27] PROFESSUR FÜR GEODÄSIE UND GEOINFORMATIK UNIVERSITÄT ROSTOCK: *UTM-Projektion/-Koordinaten*. Website. http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=1719 und http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=-740997337 besucht am 14.12.2011.

[28] PROFESSUR FÜR GEODÄSIE UND GEOINFORMATIK UNIVERSITÄT ROSTOCK: *WGS'84*. Website. http://www.geoinformatik.uni-rostock.de/einzel.asp?ID=1785 besucht am 14.12.2011.

[29] PROJEKTPRAKTIKUM MARKANTE MERKMALE 2 UNTER DER LEITUNG VON PROF. DR. LUTZ PRIESE UND FRANK SCHMITT: *3D-Modell des Campus Koblenz der Universität Koblenz-Landau*. Website. http://www.uni-koblenz.de/~lb/campusmodels/ besucht am 18.12.2011.

[30] R. HARTLEY, A. ZISSERMAN: *Multiple View Geometry in computer vision, Second Edition*. Cambridge University Press, 2003.

[31] R. LAGANIÈRE: *OpenCV 2 - Computer Vision Application Programming Cookbook*. Packt Publishing, 2011.

[32] S. ACHILLES: *Markerloses Tracking unter Verwendung von Analyse durch Synthese auf Basis von Featuredetektoren*. Diplomarbeit, 2008.

[33] W. ALT: *Nichtlineare Optimierung - Eine Einführung in Theorie, Verfahren und Anwendungen*. Vieweg Verlag, 2002.

[34] W. BURGER, M. J. BURGE: *Digitale Bildverarbeitung - Eine Einführung mit Java und ImageJ, 2. Auflage*. Springer, 2006.

[35] WIKI-USER ANTON (RP): *Zylinderprojektion*. Wikipedia-Image. http://de.wikipedia.org/w/index.php?title=Datei:Utmzylinderrp.jpg&filetimestamp=20050506225502 besucht am 14.12.2011.

[36] WIKI-USER LA2: *UTM-Zonen Europas*. Wikipedia-Image. http://de.wikipedia.org/w/index.php?title=Datei:LA2-Europe-UTM-zones.png&filetimestamp=20060901231054 besucht am 14.12.2011.