Kamerabasierte Stabilisierung von Lageinformationen auf Mobilgeräten

Bachelorarbeit

zur Erlangung des akademischen Grades

Bachelor of Science (B. Sc.)

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von: Marc Kewitz
geboren am: 8. Juli 1990
in: Berlin
Gutachter: Prof. Dr.-Ing. Peter Eisert
Prof. Dr. rer. nat. Ralf Reulke

eingereicht am: . . . . . verteidigt am: . . . . .

Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe. Weiterhin erkläre ich, eine Bachelorarbeit in diesem Studiengang erstmalig einzureichen.

Berlin, den

.....

Kurzbeschreibung

In jedem Smartphone, Tablet oder ähnlichem Gerät lassen sich heutzutage verschiedenste Sensoren finden, die unterschiedliche Aufgaben erfüllen. Darunter fällt auch die Bestimmung von Lage- sowie Positionsinformationen.
Diese Arbeit beschäftigt sich damit, einen weiteren, in dieser Hinsicht oft ungenutzen Sensor, den Kamera-Sensor, einzubringen. Aus konsekutiven Kamerabildern werden Informationen extrahiert, welche die Lageinformationen von vorhandenen Sensoren wie Beschleunigungs-, Magnetfeld- und Gyroskopsensor stabilisieren.
Die Implementierung erfolgt dabei auf einem Android-Tablet. Dabei wird Qt, eine plattformübergreifende C++-Bibliothek, zur Programmierung grafischer Benutzeroberflächen genutzt werden.

Abstract

Its a common fact that nowadays smartphones and tablets have several built-in sensors that serve and achieve different tasks. One of the most common tasks is to determine the user's position and orientation.
This thesis is dedicated to present a way to include an, in this respect, often unused sensor, the camera sensor. Information from consecutive camera frames will be used to stabilize the orientation information of the other sensors, accelerometer, magnetometer and gyroscope.
The implementation takes place on an android Tablet. Additionally, it will use the crossplatform c++ library Qt which offers several ways to create graphical users interfaces.

Inhaltsverzeichnis

1 Einleitung

Mobile Endgeräte sind heutzutage überall zu finden. Gerade in der jungen Generation besitzt so gut wie jeder ein Smartphone, Tablet oder sogar beides. Über 800 Millionen Geräte dieser Art wurden im Jahr 2012 verkauft und laut Schätzungen waren es im Jahr 2013 sogar um die 1,2 Milliarden[1].

Ein häufiger Anwendungsbereich im Mobilsegment ist die Navigation. Zur Navigation gehören sowohl Positions- als auch Ausrichtungsbestimmung. Für die Positionsbestimmung des Anwenders werden verschiedene Verfahren angewendet, wobei jeweils über ein Referenzsystem wie z.B WLAN oder GPS, die mehr oder weniger genaue Position bestimmt wird. Zur Bestimmung der Ausrichtung werden meist mehrere Sensoren verwendet, die miteinander fusioniert werden. Der Grund liegt einfach darin, dass die Einzelkomponenten oftmals relativ stark fehlerbehaftet sind. Durch die sogenannte Sensorfusion wird versucht die Schwächen der einzelnen Sensoren auszugleichen, sodass ein akzeptables Ergebnis erreicht wird.

Besonders interessant ist dieser Aspekt in Hinblick auf Augmented Reality. Mit hinreichend genauer Bestimmung von Ausrichtung sowie Position ließen sich beliebig kontextbezogene Informationen im Kamerabild anzeigen lassen.

Das Ziel der Arbeit liegt darin, neben denen für Ausrichtungsbestimmung häufig genutzten Sensoren Beschleunigungs-, Magnet sowie Gyroskopsensor, einen weiteren Sensor, nämlich die Kamera, mit einzubeziehen. Die Implementierung erfolgt auf einem Android-Tablet der Marke Asus, Modell TF700T. Daher werden sich die Implementierung sowie Fakten häufig auf das mobile Betriebssystem Android beziehen.

Im 3. Kapitel dieser Arbeit werden zunächst die vorhanden Sensoren näher beschrieben. Dabei wird sowohl auf den Aufbau und die Funktionsweise als auch ggf. die Kalibrierung der Sensoren eingegangen. Das nächste Kapitel beschäftigt sich näher mit dem Thema, wie aus den verschiedenen Sensoren die

1 http://www.gartner.com/newsroom/id/2227215

Orientierungen bestimmt werden können. Das fünfte Kapitel analysiert und vergleicht die Sensordaten sowie Orientierungen. Das sechste Kapitel beschreibt die Fusion der Sensordaten. Anschließend wird auf die Implementierung eingegangen sowie Probleme, die dabei entstanden. Im letzten Kapitel wird das Ergebnis kurz zusammengefasst und auf mögliche Verbesserungen für die zukünftige Arbeit eingegangen.

Die Gestaltung der Arbeit erfolgte nach dem „Leitfaden zur Gestaltung von Seminar-, Studien- und Diplomarbeiten“ des Fachbereichs „Informatik in Bildung und Gesellschaft“ der Humboldt-Universität zu Berlin.

2 Andere Arbeiten

Es gibt viele Arbeiten, die sich mit dem Thema der Orientierungsbestimmung in verschiedenen Situationen mit unterschiedlichster Sensorik beschäftigen, besonders in der Raum- und Luftfahrttechnik sowie Robotik.

Die Arbeiten von Xiaoping Yun und Bachmann sowie darauf aufbauende Arbeiten [1], [2], und [3] seien hier genannt. In ihnen werden verschiedene Methoden, z.B. Komplemenär- sowie Kalman-Filter, beschrieben, die das Problem mit Hilfe von Beschleunigungs-, Magnetfeld- sowie Gyroskopsensoren versuchen zu lösen.

Ähnliche Arbeiten existieren von Angelo M. Sabatini, [4], [5] und [6]. Auf diese wird im letzten Kapitel noch ein mal kurz eingegangen.

Ein interessantes, bereits weit fortgeschrittenes Projekt von Google[2], genannt „Tango“, ist erst vor wenigen Tagen publik gemacht geworden. Zusammen mit einem Tiefensensor, einer Motion-Tracking-Kamera sowie anderen Sensoren, bestimmt das Gerät Position, Orientierung sowie eine 3D-Rekonstruktion der Umgebung in Echtzeit. Das Vorgehen ähnelt dabei stark dem SLAM-Verfahren (Simultaneous Localization and Mapping) in der Robotik, nur in einem Smartphone verpackt.

2 http://www.google.com/atap/projecttango/

3 Sensoren in Mobilgeräten

Dieses Kapitel beschäftigt sich näher mit den verschiedenen Sensoren, die in heutigen Smartphones und Tablets vertreten sind. Auf Anfrage bei der Firma Asus wollte man nicht genau mitteilen, welche Hersteller für die Sensoren im Asus TF700T Tablet gewählt wurden. Unter Android lässt sich jedoch der Hersteller auslesen. Daher kann man auf die Firma Invensense schließen. Da die Sensoren (ausgeschlossen die Kamera) alle die selbe Spezifikation im Bezug auf Stromverbrauch, Auflösung sowie Aktualisierungsrate ausgeben, kann man davon ausgehen, dass das 9-Axen-MEMS MPU-9150 von Invense verbaut wurde[3]. MEMS ist im Grunde ein Prozess, bei dem kleine eingebettete Systeme oder Geräte hergestellt werden, die sowohl mechanische als auch elektrische Komponenten vereinen. Sie sind klein und meist energiesparend und finden daher oft in mobilen Geräten, aber natürlich auch in vielen anderen Bereichen, Anwendung.

Sensoren sind nicht perfekt. Sie unterliegen äußeren Einflüssen, die zu Ungenauigkeiten führen. Dazu zählen z.B. Temperatur und elektromagnetische Felder. Auch andere Bauteile in einem Mobilgerät können stören. Daher werden Sensoren bereits bei der Herstellung sowie nach dem Einbau im Gerät kalibriert, um den Störeinflüssen entgegenzuwirken. Trotzdem ist eine periodische Rekalibrierung nötig, da sich z.B. mechanische Bauteile durch Schläge verändern können und so Nullpunktverschiebungen (Bias) entstehen. Besonders wichtig ist die Entfernung des Bias bei Gyroskopsensoren, denn die Werte dieses Sensors müssen für brauchbare Informationen über die Zeit integriert werden. Bei starker Abweichung akkumulieren sich so schnell große Fehler. Laut Invensense [7] findet eine Kalibrierung im eigenen integrierten Sensorframework statt. Typischerweise geschieht dies, wenn das Gerät eine Ruhelage feststellt. Kalibrierung der klassischen Sensoren wird in dieser Arbeit nicht weiter behandelt, da es den Rahmen überschreiten würde.

Das Koordinatensystem ist in Abbildung 1 dargestellt, wobei das Gerät auf einer flachen Oberfläche liegt und der Bildschirm nach oben zeigt.

3 http://www.invensense.com/mems/gyro/nineaxis.html

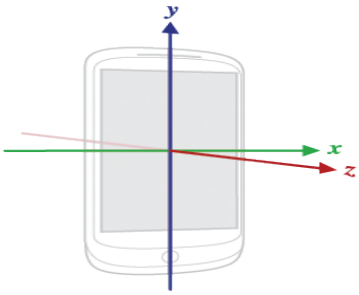

Abbildung 1: Koordinatensystem in Android

3.1 Beschleunigungssensor

Der Beschleunigungssensor wird im Bereich Navigation oftmals zur Bestimmung der Neigung eines Objektes genutzt. Damit ist gemeint, wie stark das Objekt um die x- und y-Achse, wie in Abbildung 1 zu sehen, gedreht ist.

Ein Beschleunigungssensor, auch Accelerometer, ist ein Gerät, das translatorische Beschleunigungen misst. Es existieren verschiedene Verfahren die Beschleunigung zu messen. Das Grundprinzip beruht jedoch auf dem Feder-Masse-Prinzip, bei dem eine Testmasse durch die Beschleunigungskraft ausgelenkt wird (siehe Abbildung 2). Die Stärke der Auslenkung lässt dann auf die eigentliche Beschleunigung zurückschließen.

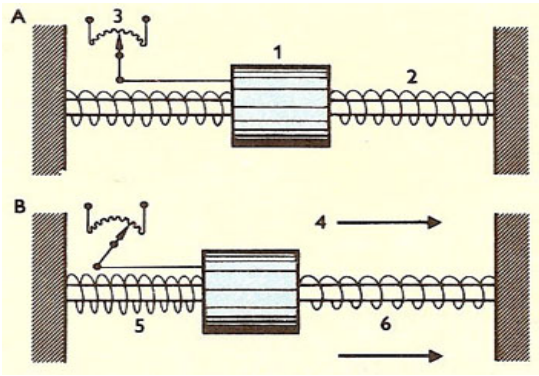

Abbildung 2: Vereinfachte Darstellung des Feder-Masse-Prinzips

Eine in Mobilgeräten häufig verwendete Art von Beschleunigungssensoren beruht auf dem Prinzip eines Differentialkondensators. Dabei sind mehrere bewegliche Kondensatorplatten an der Masse befestigt und nicht beweglichen Kondensatorplatten gegenüber kammartig angeordnet.

Die Bewegung der Masse sorgt für eine Veränderung der Plattenabstände, was wiederum eine Änderung der Kapazität bewirkt. Die entstehende Kapazität lässt sich aus der Formel $C = \epsilon \frac{A}{d}$ **, mit Dielektrizitätskonstante** ϵ **, Fläche A und Abstand d, bestimmen. Aus der Kapazität lässt sich die Stärke der Beschleunigung ermitteln.**

Der Vorteil der Anordnung wie in Abbildung 3 ist, dass zu jeder Bewegung 2 Kapazitäten entstehen. Wenn die eine steigt, sinkt die andere. Dies führt zu einer genaueren Messung der Kapazität, wie in [9] aufgeführt.

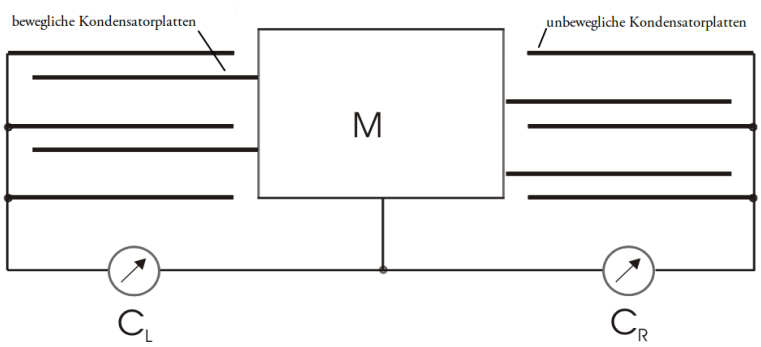

Abbildung 3: Differenzialkondensator

Der verbaute Beschleunigungssensor von Invensense arbeitet mit einer maximalen Frequenz von 50Hz, was unter Android ausgelesen wurde. Der Hersteller gibt nicht an, nach welchem Prinzip der Sensor arbeitet. Wenn das Mobilgerät ruhend auf einem flachen Tisch liegt, wird nur die Erdbeschleunigung g auf der z-Achse gemessen.

3.2 Magnetfeldsensor

Der Magnetfeldsensor, auch Magnetometer genannt, ist der Sensor, welcher für die eigentliche globale Ausrichtung im Bezug auf das magnetische Feld der Erde zuständig ist. Er ist unabdinglich, wenn es um die Navigation mit Mobilgeräten geht. Dieser Sensor wird oftmals nur für die Ausrichtung um die z-Achse genutzt.

Es gibt im Grunde 2 Arten von Magnetfeldsensoren, Skalar-Magnetometer sowie Vektor-Magnetometer. Erstere messen lediglich die Gesamtstärke des magnetischen Feldes, dem sie ausgesetzt sind. Vektor-Magnetometer hingegen messen die Stärke des magnetischen Feldes in eine bestimmte Richtung. Da die Orientierung im dreidimensionalen Raum stattfinden soll, sind 3 orthogonal ausgerichtete Vektor-Magnetometer nötig, die jeweils die Magnetfeldstärke auf einer der 3 orthogonalen Achsen messen.

Die Firma Invensense gibt in den Produktspezifikationen an, dass der 3-Achen-Magnetometer AK8975 der Firma AKM verbaut wurde. Dieser nutzt den Hall-Effekt zur Bestimmung der Magnetfeldstärke. Der Hall-Effekt beschreibt das Auftreten einer Spannung, auch Hall-Spannung genannt, wenn ein stromdurchflossener Leiter sich in einem zum Leiter senkrecht liegenden Magnetfeld befindet. Die Spannung bestimmt sich wie folgt:

$$U_h = \frac{R_h I B}{d}$$

wobei R_h die Hall-Konstante, I die Spannung, B die magnetische Flussdichte und d die Dicke des Leiters ist. R_h ist abhängig vom Material des Leiters und bestimmt sich durch $R_h = \frac{1}{ne}$, mit n als Dichte des Ladungsträgers und e als Elementarladung. Aus der Formel der Hall-Spannung geht hervor, dass diese Spannung proportional zur Stärke des Magnetfeldes (magnetische Flussdichte) ist. Also lässt sich bei Bekanntheit der geometrischen Eigenschaften des Leiters sowie Ladungsträgerdichte die Stärke des Magnetfelds bestimmen. In Abbildung 4 ist zu sehen, wie man sich den Aufbau eines auf dem Hall-Effekt basierenden Magnetometers vorstellen kann. Die Einheit unter Android wird in μT (10^{-6} Tesla) angegeben und das Koordinatensystem ist wie in Abbildung 1.

Es ist wichtig anzumerken, dass der Magnetometer das Magnetfeld zwischen den

magnetischen Polen, nicht den geographischen Polen, misst. Daher müsste für eine Endnutzeranwendung die Ausrichtung korrigiert werden. In dieser Arbeit spielt dies jedoch keine Rolle.

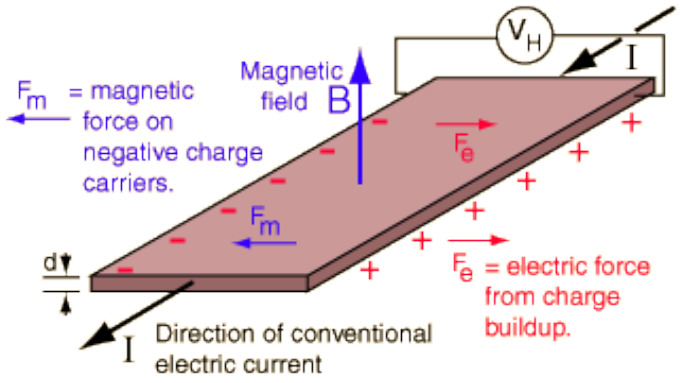

Abbildung 4: Hall-Effekt Magnetometer

3.3 Gyroskopsensor

Das Gyroskop ist ein weiteres Werkzeug in einem Mobilgerät, welches der Orientierung dient. Dabei lassen sich damit jedoch keine absoluten Orientierungen wie beim Magnetometer feststellen, sondern lediglich relative Rotationen bzw. Winkel zwischen aufeinander folgenden Sensoroutputs. Somit eignet sich dieser Sensor gut zur Unterstützung anderer Sensoren bzw. zur Sensorfusion. Im Grunde misst das Gyroskop die Winkelgeschwindigkeit entlang einer bestimmten Achse. Durch Integration über die Zeit lässt sich daraus der eigentliche Rotationswinkel bestimmen.

Zur Bestimmung der Winkelgeschwindigkeit wird der Coriolis-Effekt ausgenutzt. Diese besagt, dass auf einer nicht ruhenden Masse innerhalb eines rotierenden Bezugssystems eine Kraft einwirkt. Diese Kraft ist senkrecht zur Bewegungsrichtung des Körpers sowie zur Rotationsachse des Bezugssystems. Aus der Formel der Coriolis-Kraft $F_c = -2m(\vec{w} \times \vec{v})$ **, wobei m die Masse des bewegten Körpers,** $\vec{w}$ **die Winkelgeschwindigkeit des Bezugssystems und** $\vec{v}$ **der Geschwindigkeitsvektor der Bewegung des Körpers ist, lässt sich ableiten, dass die Kraft sich proportional zur den 3 Variablen der Formel verhält.**

Wie bei den anderen Sensoren gibt es im Grunde nicht nur eine Weise diesen Sensor zu bauen, sondern viele verschiedene, die alle ihre Vor- und Nachteile haben. Bei MEMS Gyroskopsensoren finden häufig Vibrationskreisel Anwendung, da diese bei gleicher Genauigkeit günstiger sowie einfacher produziert werden können als herkömmliche Gyroskopsensoren.

Auch Vibrationskreisel wiederum werden auf verschiedene Weisen hergestellt. Eine häufige Variante, welche auch von Invensense genutzt wird, ist der Tuning Fork Gyroskopsensor. Dabei sind 2 Masseobjekte in einem System verbunden. Diese sind diese so ausgerichtet, dass sie bei Rotation um eine Achse in entgegengesetzte Richtungen mit gleicher Stärke oszillieren. Durch diese Rotation wirkt jeweils die Coriolis-Kraft auf die Objekte. Diese Kräfte sind ebenfalls entgegengesetzt gerichtet und bewirken zur Ebene des Systems orthogonale (out-of-plane) Vibrationen, welche auf verschiedene Weisen gemessen werden, oftmals jedoch kapazitiv. Die Änderung der Kapazität ist dabei proportional zur Winkelgeschwindigkeit. Auf Grund der entgegengesetzten Ausrichtung, ist der Sensor robust gegenüber linearer Beschleunigung. Wirkt diese Beschleunigung auf die beiden Massen, bewegen sie sich in die gleiche Richtung und somit wird keine Differenz in der Kapazität gemessen. In Abbildung 5 ist eine schematische Darstellung eines solchen Sensors dargestellt. Es sind die 2 Massen sowie die entgegensetzen Kräfte zu sehen, welche die Vibrationen erzeugen. Da so ein Sensor nur die Rotation um eine Achse misst, werden im Normalfall 3 dieser Sensoren in einem orthogonalen System verbunden, damit die Rotationen um alle Koordinatenachsen gemessen werden können.

Die Einheit der Ausgabe des Sensors in Android ist rad/s. Um aus dieser Winkelgeschwindigkeit die eigentliche Rotation zu berechnen, muss die Winkelgeschwindigkeit über die Zeit t integriert werden.

$$\Phi(\tau) = \int_0^{\tau} w(t)\,dt$$

$\Phi(\tau)$ ist der Winkel, w(t) die Winkelgeschwindigkeit und τ die Zeit seit Start der Messung. Das Koordinatensystem des Sensors ist wieder wie in Abbildung 1.

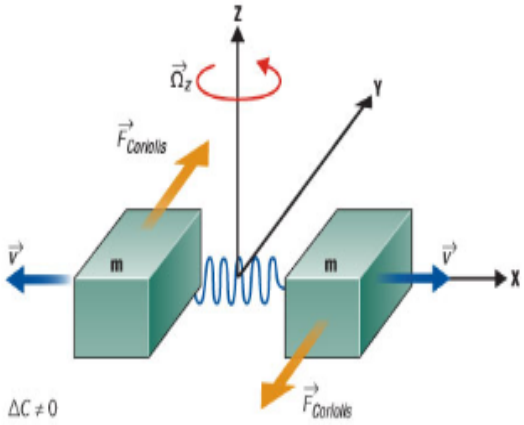

Abbildung 5: Darstellung eines Tuning Forks Vibrationskreisels

3.4 Kamera

Asus gibt auch bei der Kamera nicht an, wer Hersteller des Kamerasensors ist. Es wird lediglich angegeben, dass es sich um einen 8 Megapixel, rückwärtig beleuchteten CMOS-Sensor und einer Linse mit f/2,2 Öffnungsweite[4] handelt. CCD-Sensoren sind in Mobilgeräten unüblich, auf Grund höherer Produktionskosten sowie höherem Stromverbrauchs.

3.4.1 Aufbau- und Funktionsweise

Was digitale Kameras mit ihren analogen Kollegen auch heutzutage noch gemeinsam haben, ist, dass die optischen Aspekte noch immer die gleichen sind. Es befindet sich eine Linse oder ein Linsensystem vor der Bildfläche, welche das Licht fokussiert. Dabei wird versucht genügend Licht in kurzer Zeit zu sammeln, sodass Objekte sich im Bild nicht offensichtlich bewegen und das Bild ausreichend hell ist, um genügend Detail und Kontrast zu zeigen. Ein einfaches Modell, welches diesen Vorgang beschreibt, ist das Thin-Lens-Modell. Dieses Modell beschreibt den Vorgang wie folgt: Lichtstrahlen, welche von einem Punkt ausgesendet werden, wandern durch die Linse und konvergieren in einem Punkt

4 https://www.asus.com/Tablets_Mobile/ASUS_Transformer_Pad_TF700T

hinter der Linse. Dabei wird der Abstand zwischen der Linse und dem Punkt hinter der Linse, in dem die Lichtstrahlen sich fokussieren, Brennweite f genannt. Darüber hinaus wird durch z_0, dem Abstand zwischen Linse und Bildfläche und z_1, dem Abstand zwischen Linse und einem Punkt eines Objektes, die Thin-Lens-Gleichung definiert: $\frac{1}{|f|}=\frac{1}{z_0}+\frac{1}{z_1}$.

Damit diese Lichtstrahlen in ein Bild umgewandelt werden können, müssen sie auf der Bildfläche aufgefangen werden und anschließend die Bildinformationen daraus extrahiert werden. Früher war an dieser Stelle ein Kamerafilm, heute ein Sensor, wahlweise CCD- oder CMOS-Sensor. Der CMOS-Sensor ist ein Bildsensor, der seine ganze Funktionalität in einem Schaltkreis vereint. Ein solcher Sensor misst lediglich Lichtintensitäten. Daher wird dem Sensor ein Farbfilter vorgeschaltet. Durch diese Filterung sind die Farbsamples jedoch unvollständig, sodass die eigentlichen Pixelfarben im Nachhinein aus benachbarten Pixeln interpoliert werden. Dieser Vorgang wird Demosaicing genannt.

3.4.2 Kalibrierung

Die Kalibrierung der Kamera wird genutzt, um die geometrischen Eigenschaften der Kamera zu bestimmen. Dabei wird zwischen intrinsischen und extrinsischen Parametern unterschieden. Die extrinsischen Parameter beschreiben die Transformation zwischen einem bekannten Welt-Koordinatensystem und dem Kamera-Koordinatensystem, die normalerweise nicht gleich ausgerichtet sind.

Die intrinsischen Parameter beschreiben die Eigenschaften, die vom Kamera-Koordinatensystem zum eigentlichen 2D-Bild führen. Sie beinhalten die Brennweite f, das Seitenverhältnis, Bildmittelpunkt sowie radiale, ggf. tangentiale Verzeichnung.

Zum Verständnis wird ein vereinfachtes Kamera-Modell betrachtet. Die Koordinaten eines 3D-Punkts $X_i = [X\ Y\ Z]^T$ und dazugehörigen 2D-Punkts $x_i = [x\ y]^T$ werden mittels perspektivischer Projektion in Relation gesetzt. Daraus ergibt sich für die Bildpunkte x und y folgende Projektion: $x=f*\frac{X}{Z}$, $y=f*\frac{Y}{Z}$.

Die perspektivische Projektion in dieser Form beschreibt dabei die ideale

Lochkamera, was nicht der Realität entspricht. In Wahrheit entstehen durch das System aus Linsen Verzeichnungen im Bild. Dabei dominiert die radiale Verzeichnung, welche durch stärkere Krümmung des Lichts zum Rand der Linse hin entsteht. Diese Verzeichnung wird modelliert durch:

$$x = x_{dist}\left(1 + \sum_i k_i r^{2i}\right)$$

$$y = y_{dist}\left(1 + \sum_i k_i r^{2i}\right)$$

mit k_i als Verzeichnungskoeffizienten und dem Radius $r^2 = \left(\frac{x_{dist}}{f}\right)^2 + \left(\frac{y_{dist}}{f}\right)^2$. x_{dist} und y_{dist} sind die verzeichneten Koordinaten des Punkts x_i. Oftmals finden nur die ersten 2-3 Koeffizienten k_i bei der Kalibrierung Verwendung. Die Brennweite sowie Koordinaten werden in Pixeln und nicht mm oder m angegeben. Da Pixel nicht 100% quadratisch sind, werden für x und y verschiedene Skalierungsfaktoren s_x, s_y bestimmt. Außerdem befindet sich im Normalfall der Mittelpunkt, also die optische Achse, nicht im Ursprung des Bildes, sondern in einem Punkt $[c_x, c_y]^T$, optimalerweise der Bildmittelpunkt. Daraus ergibt sich:

$$x = f_x * \frac{X}{Z} + c_x$$

$$y = f_y * \frac{Y}{Z} + c_y$$

mit $f_x = s_x f$ und $f_y = s_y f$. Die Korrektur der Verzeichnung ist an dieser Stelle nicht eingerechnet, kann jedoch genau so auf Pixelkoordinaten ausgeführt werden.

Um diese Parameter nun zu bestimmen, wird im Normalfall ein Kalibrierungstestobjekt benötigt, oftmals ein Schachbrettmuster. Da die geometrischen Eigenschaften des Objekts bekannt sind, können die Parameter der Kamera aus 2D-Korrespondenzen zwischen den projizierten 3D-Punkten und 2D-Punkten im Bild errechnet werden. In dieser Arbeit fand eine Kalibrierung der Kamera nach [16] statt. Dabei wurde ein Kalibrierungsobjekt wie in Abbildung 6 genutzt. Der Unterschied ist, dass bei dieser Methode keine

Featurepunkte im Bild bestimmt werden, sondern ein 3D-Modell des Kalibrierungsobjekts wird erstellt und in verschiedenen Ansichten gerendert. Dabei werden alle Bildinformationen so genutzt, dass die Ansicht dem Kamerarahmen exakt angepasst ist. Durch Nutzung mehrerer Bilder können dabei Fehler minimiert werden.

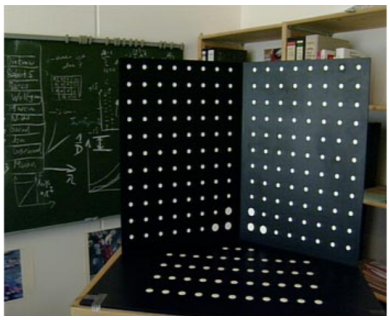

Abbildung 6: Testobjekt zur Kalibrierung der Kamera

Bei der Kalibrierung wurden das Seitenverhältnis a, der Öffnungswinkel der Höhe φ_{height} sowie 2 Verzeichnungskoeffizienten k_1, k_2 bestimmt. Aus dem Öffnungswinkel lässt sich f_y bestimmen, mittels der Formel $\varphi_{height} = 2\arctan\left(\frac{2f_y}{N_y}\right)$. Die Brennweite f_x in x-Richtung ist dabei identisch, wenn man von der Formel zur Bestimmung des Seitenverhältnisses $a = \frac{N_x}{N_y} \cdot \frac{f_y}{f_x}$ ausgeht, mit N_x, N_y als Anzahl der Pixelsensoren in x- und y-Richtung, wobei das Seitenverhältnis des Bildes gleich dem Verhältnis der Sensoren ist.

Die Verzeichnungskoeffizienten waren erstaunlicherweise relativ klein, sodass sie im späteren Anwendungsprogramm zumindest mit dieser Kamera nicht berücksichtigt wurden, da sie nur einen kleinen Genauigkeitsgewinn bringen, jedoch einen größeren Performanceverlust.

4 Bestimmung der Messwerte

Zunächst sollte gesagt sein, dass in dieser Arbeit die Verwendung von Euler-Winkeln, wie sie häufig verwendet werden, vermieden wurde. Dies hat den einfachen Grund, dass Euler-Winkel Probleme bereiten, wenn 2 Achsen durch die Aneinanderreihung von Rotationen aufeinander fallen, der so genannte Gimbal Lock, und somit ein Freiheitsgrad der Rotation wegfällt. Dieser Fall tritt hier auf, denn das Tablet wird bei Verwendung aufrecht gehalten, sodass die x-Achse um 90 Grad rotiert ist. Das Problem kann z.B. durch Verwendung von Quaternionen zur Darstellung von Orientierungen vermieden werden. Quaternionen werden hier wie folgt dargestellt:

$$q = a + b \cdot i + c \cdot j + d \cdot k = (a, b, c, d) \quad ,$$

wobei a, b, c und d reelle Zahlen sind und i, j, k Einheitsvektoren entlang der Achsen x, y und z. Der Skalar a ist dabei der Realteil des Quaternions und (b, c, d) der Imaginärteil.

Um mit Quaternionen zu arbeiten, muss definiert werden, wie sie aus den Sensordaten bestimmt werden.

4.1 Klassische Sensoren

Für ein Gyroskop ist diese Aufgabe relativ simpel. Sei nun der Quaternion $q_{\omega,t}$ definiert als Quaternion mit Realteil 0 und den Winkelgeschindigkeiten ω_x, ω_y und ω_z als Imaginärteil zum Zeitpunkt t, dann lässt sich die Orientierungsänderung einer Orientierung q_{t-1} wie folgt bestimmen [17]:

$$\dot{q_{\omega,t}} = \frac{1}{2} q_{t-1} \otimes q_{\omega,t} \quad , \qquad \textbf{(4.1)}$$

wobei $\otimes$ das Hamilton-Produkt zur Multiplikation von Quaternionen definiert. Mit Hilfe der Orientierungsänderung und dem Zeitintervall Δt seit der letzten Messung lässt sich die neue Orientierung bestimmen.

$$q_t = q_{t-1} + \dot{q}_{\omega,t} \Delta t$$

Um eine Orientierung aus den Messwerten des Beschleunigungssensors sowie Magnetfeldsensors zu bestimmen, muss ein anderes Verfahren angewendet werden. Es gibt leider keinen Quaternion, der die Messwerte der Sensoren direkt in die bekannten Werte(lokale Gravitation sowie lokales Magnetfeld) überführt. Daher muss mit Hilfe der Lösung eines Minimierungsproblems der bestmögliche Quaternion bestimmt werden [3]. Dabei findet der Gauß-Newton-Algorithmus zur Minimierung eines mittleren quadratischen Fehlers Anwendung. Der Fehler E_q sei definiert durch

$$E_q = \varepsilon^t \, \varepsilon = ({}^E y_1 - M \, {}^B y_0)^T ({}^E y_1 - M \, {}^B y_0) \quad ,$$

wobei ${}^E y_1$ ein 6x1 Vektor mit den Werten der Gravitation [0 0 g] sowie des Erdmagnetfelds und ${}^B y_0$ ein 6x1 Vektor mit den Messwerten des Beschleunigungs- und Magnetfeldsensors ist. Die Matrix M definiert sich wie folgt:

$$M = \begin{pmatrix} R & 0 \\ 0 & R \end{pmatrix} \text{ , mit } \quad R = \begin{pmatrix} a^2+b^2-c^2-d^2 & 2bc-2da & 2bd+2ca \\ 2bc+2da & a^2+c^2-b^2-d^2 & 2cd-2ba \\ 2bd-2ca & 2cd+2ba & a^2+d^2-b^2-c^2 \end{pmatrix} .$$

Der Gauß-Newton-Algorithmus verwendet zur Lösung des Problems partielle Ableitungen 1. Ordnung von ε, also Jacobi-Matrizen. Der Algorithmus arbeitet iterativ und eine Iteration für einen Quaternion q_k lautet wie folgt:

$$q_{k+1} = q_k - \left[J^T(q_k) J(q_k) \right]^{-1} J^T(q_k) \, \varepsilon(q_k)$$

mit

$$J = -\left[\left(\frac{\delta M}{\delta a} \, {}^B y_0 \right) \left(\frac{\delta M}{\delta b} \, {}^B y_0 \right) \left(\frac{\delta M}{\delta c} \, {}^B y_0 \right) \left(\frac{\delta M}{\delta d} \, {}^B y_0 \right) \right]$$

Der Algorithmus konvergiert relativ zügig, sodass nach 3 bis 4 Iterationen keine nennenswerte Änderung des Quaternions mehr stattfindet und somit eine in diesem Fall optimale Lösung gefunden wurde.

Da die Werte des lokalen Magnetfelds nicht zwangsläufig bekannt sind und man

vermeiden möchte mit falschen Werten zu rechnen, kann man magnetische Verzerrungskompensation wie in [17] anwenden. Dabei werden die aktuellen Messwerte m_t des Magnetfeldsensors um die zuvor bestimmte Orientierung q_{t-1} rotiert.

$$h_t = \begin{bmatrix} 0 & h_x & h_y & h_z \end{bmatrix} = q_{t-1} \otimes m_t \otimes q_{t-1}^{-1}$$

Der so entstandende Vektor h_t definiert die Richtung des Erdmagnetfelds. Um zu verhindern, dass die gemessene Richtung eine fehlerhafte Neigung enthält und der Magnetfeldvektor die Neigungsbestimmung des Beschleunigungssensors stört, reduziert man den Vektor h_t, sodass er lediglich Komponenten in der x- und z-Achse des Erdmagnetfelds besitzt.

$$b_t = \begin{bmatrix} 0 & \sqrt{h_x^2 + h_y^2} & 0 & h_z \end{bmatrix}$$

Dieser Vektor wird an Stelle des Erdmagnetfelds in $^{E}y_1$ eingesetzt. Der Vorteil ist, dass dadurch Störungen im Magnetfeld nur die Ausrichtung um die z-Achse betreffen und man, wie bereits erwähnt, das lokale Erdmagnetfeld nicht bestimmen muss.

Der Algorithmus zur Bestimmung der Rotation zwischen konsekutiven Kamerabildern, welcher in Kapitel 4.2 vorgestellt wird, hat als Endergebnis eine Rotationsmatrix. Diese wird mittels [19] in einen Quaternion $q_{i.t}$ umgewandelt. Dieser Quaternion beschreibt lediglich die Änderung der Orientierung. Um nun einen Quaternion q_{t-1} in die neue Ausrichtung q_t zu rotieren, wird der Quaternion wie bei Rotationsmatrizen multipliziert. Auch hier ist die Reihenfolge wichtig.

$$q_t = q_{t-1} \otimes q_{i,t}$$

4.2 Rotationsschätzung aus Bildinformationen

Dieses Kapitel beschreibt, wie man aus konsekutiven Bildern eine Rotation schätzen kann. Dabei werden in den Bildern Featurepunkte bestimmt, ausgewertet und in aufeinander folgenden Bildern verglichen. Daraus wird versucht gute Korrespondenzen zu bestimmen, die anschließend zur Rotationsschätzung herangezogen werden. Die Bestimmung von Featurepunkten ist allgemein eine rechenintensive Operation. Daher muss beachtet werden, dass man nicht unbegrenzt Rechenressourcen sowie Zeit hat. Bekannte Algorithmen wie SIFT und SURF sind daher ungeeignet, da sie selbst auf modernen Computern nur bedingt echtzeitfähig sind. Das trifft erst recht auf Mobilgeräte zu, die trotz 4 Kernen und mittlerweile mehreren GB RAM eher Leistungen von Rechnern von vor 6-7 Jahren erbringen[5]. Der Vergleich ist relativ schwach, da es andere Plattformen sind, aber es trifft in etwa zu. Die Wahl des Algorithmus fiel auf ORB. Die Gründe liegen hier größtenteils beim Performance-Vorteil gegenüber anderen Algorithmen. Algorithmen wie SIFT, SURF, BRISK, FREAK, MSER lieferten selbst bei geringerer Auflösung nur wenige Bilder pro Sekunde. Das Ziel war zwischen 5 und 10 Bilden zu bleiben, damit auch schnelle Bewegungen gut erkannt werden können. Außerdem bietet ORB eine höhere Rotationsinvarianz gegenüber BRIEF, welcher ähnlich schnell arbeitet. Diese Invarianz ist gerade gefragt. Ein möglicher Nachteil ist, dass ORB manchmal im Vergleich zu anderen Algorithmen weniger Featurepunkte erkennt, was zu Ungenauigkeiten führen kann.

4.2.1 ORB

ORB ist eine Erweiterung der Algorithmen FAST und BRIEF, bei der versucht wird eine höhere Robustheit gegenüber Rotationen sowie Skalierung zu erzielen. ORB zählt zu den binären Featurepunkt-Algorithmen, welche hohe Performance versprechen. Zur Bestimmung von Featurepunkten gehören 2 Schritte: die Erkennung(Detektion) sowie Beschreibung(Deskription) der Punkte.

Zur Detektion der Featurepunkte in ORB wird der Corner-Detector FAST benutzt.

5 http://www.anandtech.com/show/6877/the-great-equalizer-part-3/2

Da FAST für Featurepunkte keine Orientierung berechnet, erweitert ORB FAST um eine Orientierungsberechnung. Zur Anwendung kommt dabei der „Intensity Centroid“, Intensitätsschwerpunkt. Dabei geht man davon aus, dass die Intensität einer Ecke von deren Zentrum aus abweicht und berechnet darüber die Orientierung. Zur Berechnung des Schwerpunktes nutzt man Momente der Pixel, wobei das Moment (p+q)-ten Grades als $m_{pq}=\sum_x \sum_y x^p y^q I(x,y)$ **, mit I(x,y) der Intensität an Stelle (x,y), definiert ist. Die Größe des Patches, auf dem gearbeitet wird, ist dabei gleich dem Radius r des Kreises, auf dem Fast arbeitet. Der Schwerpunkt C ist dann definiert als** $C=\left(\frac{m_{10}}{m_{00}},\frac{m_{01}}{m_{00}}\right)$ **. Aus der Ecke 0 und dem Schwerpunkt C wird dann einfach ein Vektor** $\vec{OC}$ **gebildet, dessen Orientierung φ durch atan2(m_{10},m_{01}) definiert ist.**

Dies ist nicht die einzige Erweiterung zu FAST in ORB. Im Original berechnet FAST keine Multiscale-Featurepunkte, was bedeutet, dass die Featurepunkte so gut wie keine Invarianz gegenüber Skalierung zeigen. Um dies zu verbessern, wird eine Scale-Pyramide angewendet. Das Bild wird in verschiedene Pyramidenlevel mit unterschiedlicher Skalierung berechnet. Für jedes Level werden anschließend die Featurepunkte berechnet. Dabei bestimmen ein Skalierungsfaktor und die Anzahl der Level über die Qualität der Features.

Ein weiteres Problem in FAST ist, dass den Features keine Wertung im Bezug auf Qualität der Ecken zugeordnet wird. Daher wird der Harris-Corner-Filter auf diese angewendet, um sowohl eine Reihenfolge der Punkte zu erzeugen als auch mögliche Kanten zu eliminieren.

Wie bereits erwähnt, verwendet ORB BRIEF als Deskriptor. BRIEF ist so gut wie nicht invariant gegenüber Rotationen. Das Ziel ORBs ist es, dies zu kompensieren. Dabei wird versucht eine möglichst hohe Varianz sowie eine geringe Korrelation zwischen den Punktepaaren zu erreichen. Hohe Varianz bringt eine bessere Unterscheidung der Featurepunkte, geringe Korrelation bewirkt, dass mehr Informationen aus jedem Punktepaar im Patch gewonnen werden. rBRIEF, die Erweiterung, versucht die optimale Verteilung der Punktepaare in den Patches zu erlernen. Dafür wurden aus den Bildern des PASCAL VOC 2006 Datensets etwa 300000 Featurepunkte extrahiert. Dann

wurden alle möglichen Binärtests, die in einem 31 x 31 Patch möglich sind, bestimmt und mit Hilfe eines Greedy-Algorithmus die Binärpatches so angeordnet, dass die gewünschten Eigenschaften, hohe Varianz sowie kleine Korrelation, erreicht werden.

4.2.2 Matching von Featurepunkten

Beim Matching werden Descriptoren von 2 verschiedenen Bildern verglichen, um so Featurepunkte wiederzuerkennen. Im Fall von Binärdescriptoren ist dies relativ simpel, da man lediglich die Hammingdistanz zwischen den einzelnen Binärstrings berechnen muss. Die Hammingdistanz lässt sich mit dem exklusiven Oder wie folgt berechnen:

$$d_H = \sum_{i=0}^{N-1} a[i] \quad XOR \quad b[i] \tag{4.2}$$

wobei a[i] und b[i] die Werte der Binärstrings der Länge n an Stelle i sind. Die Hammingdistanz muss für jeden Deskriptor des ersten Bildes mit allen Deskriptoren des zweiten Bildes berechnet und verglichen werden. Das Match mit der geringsten Hammingdistanz ist mit höchster Wahrscheinlichkeit der gleiche Punkt. Dieser Vorgang wird auch Bruteforce Matching genannt.

Wird der Descriptor in einem anderen Kontext, wie z.B. Bilderkennung in großen Bilddatenbanken, genutzt, ist das Bruteforce Matching ineffizient. In diesem Fall werden Nearest-Neighbour-Algorithmen, wie z. B. LSH, verwendet, da diese auf großen Featurepunktmengen effizienter arbeiten.

4.2.3 Verbesserung der Auswahl an Featurepunkten

In Normalfall sind nicht alle Featurepunktpaare korrekt. Veränderungen der Bildposition, Lichtintensität sowie die geringe Auflösung tragen dazu bei, dass Featurepunkte teilweise erst gar nicht im 2. Bild wiedergefunden oder falsch zugeordnet werden. Daher wurden 2 Verfahren angewendet, um diese Fehler zu minimieren, Crosscheck sowie Thresholding.

Bei Crosscheck werden sowohl die Featurepunkte aus Bild 1 mit den Punkten aus Bild 2 verglichen als auch umgekehrt. Dabei kann es passieren, dass ein Punkt x_1

aus Bild 1 mit einem Punkt y_1 aus Bild 2 zugewiesen wird, umgekehrt der Punkt y_1 jedoch nicht mit x_1, sondern x_2 gepaart wird. Wenn dies der Fall ist, kann man diese Korrespondenz offensichtlich verwerfen. Es müssen vorwärts sowie rückwärts die gleichen Punktepaare entstehen. In Abbildung 7 ist dieser Fall dargestellt. Nur der Punkt x_1, welcher mit y_1 gepaart wird, ist auch umgekehrt der am nächsten liegende Punkt. Daher fallen die Korrespondenzen (x_2, y_1) sowie (x_3, y_1) heraus.

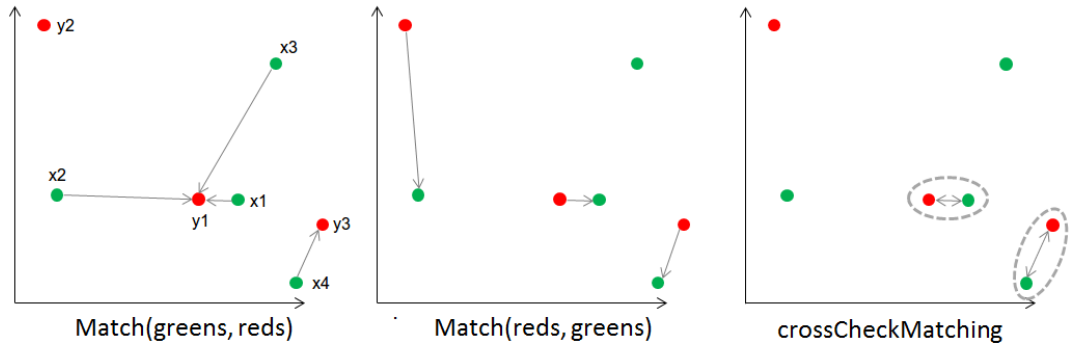

Abbildung 7: Veranschaulichung des Crosscheck-Verfahrens

Nach dem Crosscheck sind ebenfalls nicht alle Paare zwangsläufig korrekt zugeordnet. Crosscheck kann die Menge an falschen Punktepaaren lediglich reduzieren. Um die Auswahl weiter zu verbessern, kann man einen Grenzwert für den Abstand d_H, Hammingdistanz, aus Gleichung (4.2) einführen, bei dem die Korrespondenz noch als glaubwürdig interpretiert wird. In Abbildung 8 ist eine typische Verteilung der Abstände zu sehen. In diesem Fall wurden die Abstände über 10 Durchläufe akkumuliert. Es ist zu sehen, dass die Abstände sich im Bereich von 20-40 häufen. Dies ist kein Zeichen dafür, dass alle Korrespondenzen in diesem Bereich korrekt sind, jedoch kann man Korrespondenzen mit größerem Abstand verwerfen und hat trotzdem noch eine große Menge an Punktepaaren, die für die anschließende Rotationsschätzung genutzt werden kann. Dabei ist der Anteil korrekter Paare mit hoher Wahrscheinlichkeit größer als zuvor.

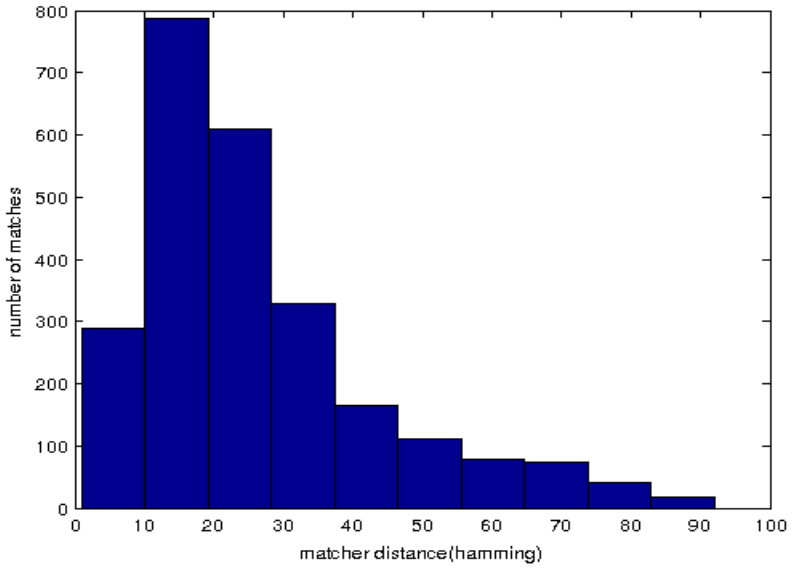

Abbildung 8: Verteilung der Abstände der Korrespondenzen

4.2.4 Schätzung der Rotation aus Featurepunkten

Für die spätere Verwendung zur Datenfusion muss nun die Rotation aus den Korrespondenzen bestimmt werden. Dies geschieht nach [20] und [21]. Da die Kameramatrix K durch die Kalibrierung bekannt ist, lässt sich das Problem mit Hilfe normierter Entstehungsstrahlen lösen. Der Strahl für einen Bildpunkt x_i bestimmt sich wie folgt:

$$\hat{r}_i = \frac{r_i}{|r_i|} \quad \textbf{mit} \quad r_i = K^{-1} x_i$$

Die Rotation wird durch Minimierung der Abstände zwischen allen korrespondierenden Strahlen geschätzt. Folgende Minimierungsfunktion beschreibt diesen Vorgang:

$$E = \sum_i \| R_1^{-1} \hat{r}_{i,1} - R_2 \hat{r}_{i,2} \|^2$$
$$= \sum_i \| \hat{r}_{i,1} - R_{12} \hat{r}_{i,2} \|^2 = \sum_i \| \hat{r}_{i,2} - R_{21} \hat{r}_{i,1} \|^2$$

Das Problem der Minimierung lässt sich mit Hilfe der Singulärwertzerlegung lösen. Dazu wird zunächst die Korrelationsmatrix C zwischen korrespondieren Strahlen bestimmt.

$$C = \sum_i \hat{r}_{i,2} \hat{r}_{i,1}^T$$

Die Singulärwertzerlegung von C ergibt $C = U \Sigma V^T$. **Die Schätzung der Matrix R_{21} bestimmt sich dann aus:**

$$R_{21} = U \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & s \end{pmatrix} V^T \quad \textbf{, mit} \quad s = sgn(det(U V^T))$$

Es ist wichtig zu sagen, dass bei der Schätzung der Rotation im dreidimensionalen Raum 3 Korrespondenzen nötig sind, die zufällig aus der Menge an Korrespondenzen gewählt werden.

Die Schätzung der Rotation, die dabei erreicht wird, ist nicht zwangsläufig für alle Korrespondenzen optimal, denn wie zuvor erwähnt, sind nicht alle Paare korrekt. Wenn falsche Korrespondenzen ausgewählt werden, bedeutet dies automatisch, dass die Rotationsschätzung schlecht wird. Um dem entgegenzuwirken, wird der Random Sample Consensus Algorithmus(RANSAC) wie in [22] und [23] angewendet. Dieser nutzt aus, dass die falschen Punktepaare, auch Ausreißer genannt, keine einheitlichen Eigenschaften wie Betrag oder Richtung besitzen und diese im Vergleich zu den korrekten Korrespondenzen zufällig erscheinen. Der Algorithmus geht dabei folgendermaßen vor:

Zunächst wird eine zufällige Menge an Korrespondenzen ausgewählt und damit die Rotation wie zuvor vorgestellt geschätzt. Mit Hilfe der Kameramatrix K kann daraus der mittlere Pixelabstand d_p zwischen allen Verbindungen bestimmt werden.

$$d_p = \frac{d_{p1} + d_{p2}}{2} \quad \textbf{, mit} \quad d_{p1} = \|x_2 - H_{21} x_1\|,\ d_{p2} = \|x_1 - H_{12} x_2\|$$

Die Homographien H_{21} bzw. H_{12} lassen sich unter der Voraussetzung, dass die Kamera lediglich einer Rotation unterliegt, wie folgt bestimmen:

$$H_{21} = K R_{21} K^{-1},\ H_{12} = K R_{12} K^{-1}$$

Die Pixelabstände werden in beide Richtungen bestimmt, da d_{p1} und d_{p2} ungleich sind auf Grund von erhöhter Auflösung der Pixel pro Winkeleinheit mit zunehmenden Abstand von der optischen Achse.

Sollten die Korrespondenzen, die zur Schätzung der Rotation gewählt wurden, aus der Menge der korrekten Punktepaare stammen, so ergibt sich für alle Paare dieser Menge ein kleiner Pixelabstand d_p. Wurden im Gegenzug falsche Korrespondenzen herangezogen, ergeben sich relativ große Pixelabstände für jede Verbindung. Damit nun eine gute Rotation gefunden wird, wird die Rotationsschätzung wiederholt durchgeführt und am Ende diejenige Rotation ausgewählt, bei der die meisten Korrespondenzen, die Inlier, einen Pixelabstand kleiner einem Grenzwert ε, besitzen. Der Grenzwert sollte nicht zu niedrig gewählt, da auch bei guten Schätzungen die Pixelabstände im niedrigen einstelligen Bereich liegen werden und nicht im Subpixelbereich. Die Gründe dafür sind vielfältig. Einerseits ist die genutzte Auflösung mit 640x480 gering, sodass Informationen durch Herunterskalierung verloren gehen, andererseits die fehlende Verzeichnungskorrektur sowie die Annahme, dass die Kamera lediglich Rotationen ausgesetzt ist, was nicht vollständig korrekt ist.

5 Auswertung der Messdaten

Der verbaute Gyroskopsensor ist relativ wenig gestört. Wie bereits erwähnt korrigiert der interne Algorithmus des Chips der Firma Invensense Fehler wie den Gyro-Bias und kann so den Gyro-Drift minimieren. Die Korrektur findet bei der Feststellung von Ruhelage statt. In Abbildung 9 kann man diese Korrektur beispielhaft sehen. Der Wert auf allen Achsen liegt anschließend nahe 0, genauer gesagt -5.3E^{-7}.

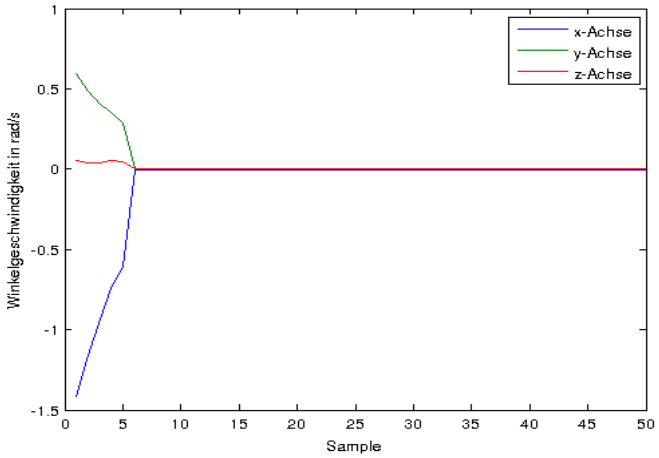

Abbildung 9: Bias-Korrektur des Gyroskopsensors

Die Werte bleiben anschließend konstant gut und verändern sich nur, wenn sehr hohe Temperaturunterschiede herrschen. Diese Änderung wird in Ruhelage erneut korrigiert.

Dieses Ergebnis führt zu einer sehr geringen Abweichung, wenn die Sensordaten über die Zeit integriert werden. Abbildung 10 zeigt den Drift über 60 Sekunden. Die Sprünge können durch leichte Vibrationen entstanden sein. Die Abweichung liegt in diesem Beispiel bei maximal 0.05 Grad, was sehr gut ist.

Auch der Beschleunigungssensor liefert relativ gute Ergebnisse. Die Messungen auf den Achsen liefern bei Ruhelage nicht exakt die Erdfallbeschleunigung g.

Durch eine einfache Kalibrierung wurde das jedoch behoben. Dafür wurden die Werte auf den Achsen über einen längeren Zeitraum gemittelt. Dabei wurde so gemessen, dass die Achsen jeweils für einen längeren Zeit der Gravitation ausgesetzt sind, positiv sowie negativ. Das bedeutet einfach ausgedrückt, dass die getestete Achse in unserem Erdkoordinatensystem nach oben zeigte. Die Korrekturwerte lagen dabei zwischen 0.01 und 0.09*g.

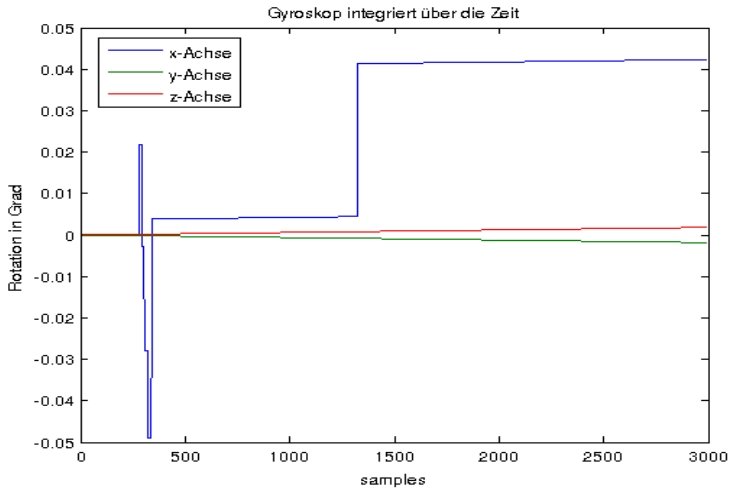

Abbildung 10: Gyroskop-Drift innerhalb von 60 Sekunden

Ein Problem, das dem Beschleunigungssensor jedoch anhängt, ist die Messung der Gravitation g sowie translatorischen/linearen Beschleunigung a. Für die Bestimmung der Orientierung mittels eines Beschleunigungssensor wird jedoch nur die Gravitation benötigt. Diese beiden Beschleunigungen müssen also getrennt werden. Oft werden dafür Tiefpassfilter benutzt, welche hohe Frequenzen (in diesem Fall die lineare Beschleunigung) aus den Messungen filtern. Die Ergebnisse sind meistens nicht schlecht, wenn die Beschleunigung nicht zu lang auf das Objekt einwirkt. An dieser Stelle wurde jedoch eine andere Idee verfolgt. Da der Gyroskopsensor relativ stabil ist, kann man diesen nutzen, um den Beschleunigungssensor im Störfall zu simulieren. Zur Detektion von Störfällen wird einfach die Norm des Beschleunigungsvektor benutzt. Im Normalfall sollte dieser immer in etwa der Erdfallbeschleunigung g entsprechen. Wenn die Norm von g plus einem kleinen Epsilon abweicht, wirkt auf das Objekt mir sehr hoher Wahrscheinlichkeit eine lineare Beschleunigung.

Für die Korrektur wird die aktuell durch den Gyroskopsensor gemessene Rotation

genutzt, jedoch entgegengesetzt der eigentlichen Richtung, denn bei Rotation bewegt sich der Beschleunigungsvektor entgegen der Rotationsrichtung. Die Korrektur sieht wie folgt aus:

$$g_t = q_{-\omega} \otimes a_t \otimes q_{-\omega}^{-1} \quad ,$$

wobei a_t eine Quaternion mit Realteil 0 und den Messwerten des Beschleunigungssensors zum Zeitpunkt t, $q_{-\omega}$ die entgegengesetzte Rotation und g_t die korrigierte Gravitation ist.

Sollte die Norm der Beschleunigungssensorwerte wieder der Erdfallbeschleunigung entsprechen, können die Sensorwerte wieder direkt verwendet werden, mit der Einschränkung, dass die Norm für mehrere Samples hintereinander g entsprechen muss. Dadurch wird verhindert, dass zufällige Konstruktionen entstehen, bei denen die Norm g entspricht, obwohl das Objekt lineare Beschleunigung erfährt.

Das Ergebnis dieser Korrektur wurde in Abbildung 11 dargestellt, in Abbildung 12 hingegen die unkorrigierten Ergebnisse. Das Gerät wurde zunächst aufrecht gehalten, sodass g auf der y-Achse wirkt, und dabei in Richtung der Gravitation geschüttelt. Anschließend wurde das Gerät 90 Grad um die x-Achse rotiert, sodass nun g auf die z-Achse wirkt, und erneut in Richtung der Gravitation geschüttelt. Man sieht deutlich, dass die korrigierten Werte durch die lineare Beschleunigung nur leicht beeinflusst werden und somit lediglich die Gravitation in den Messdaten enthalten ist. Die Abweichungen auf den anderen Achsen entstehen dadurch, dass nur schwer perfekt in Richtung der Gravitation geschüttelt werden kann. In Abbildung 13 wird der Einfluss der Korrektur auf die Orientierung gezeigt, welche mittels des Gauß-Newton-Algorithmus bestimmt wird. Die Orientierungen, die mit einem c versehen sind, sind die korrigierten Werte(dunkelblau, dunkelgrün, rot). Ab der Mitte des Tests wurde das Gerät erneut geschüttelt und dabei versucht möglichst nicht die Richtung zu ändern. Die korrigierten Winkel sind deutlich stabiler und zeigen eine geringere Abweichung.

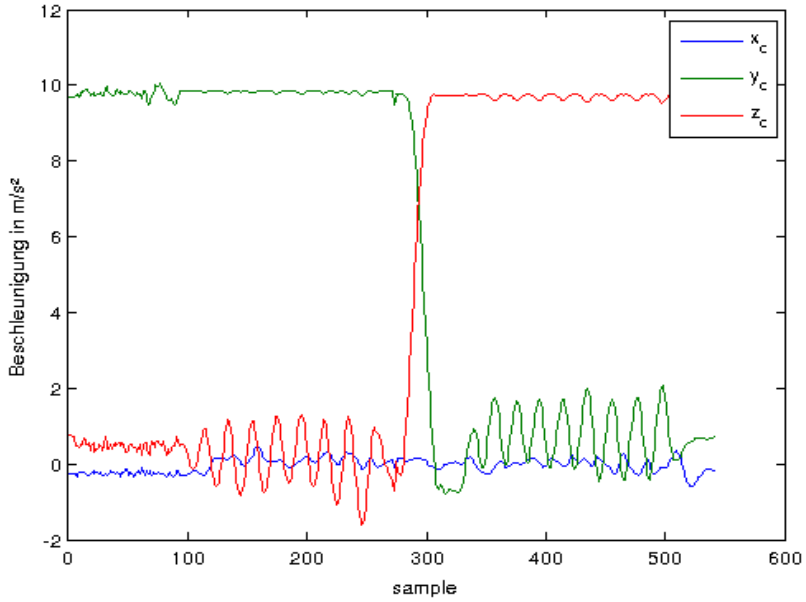

Abbildung 11: korrigierte Beschleunigungswerte, sodass nur g enthalten ist

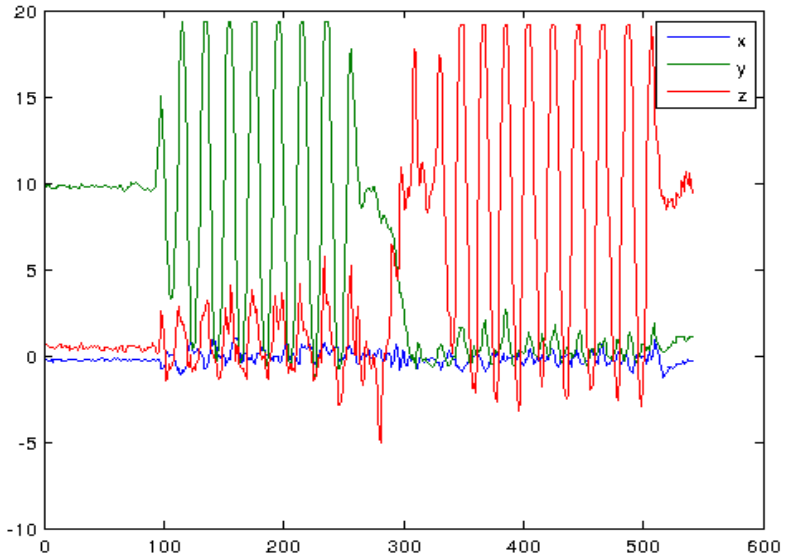

Abbildung 12: unkorrigierte Beschleunigungswerte

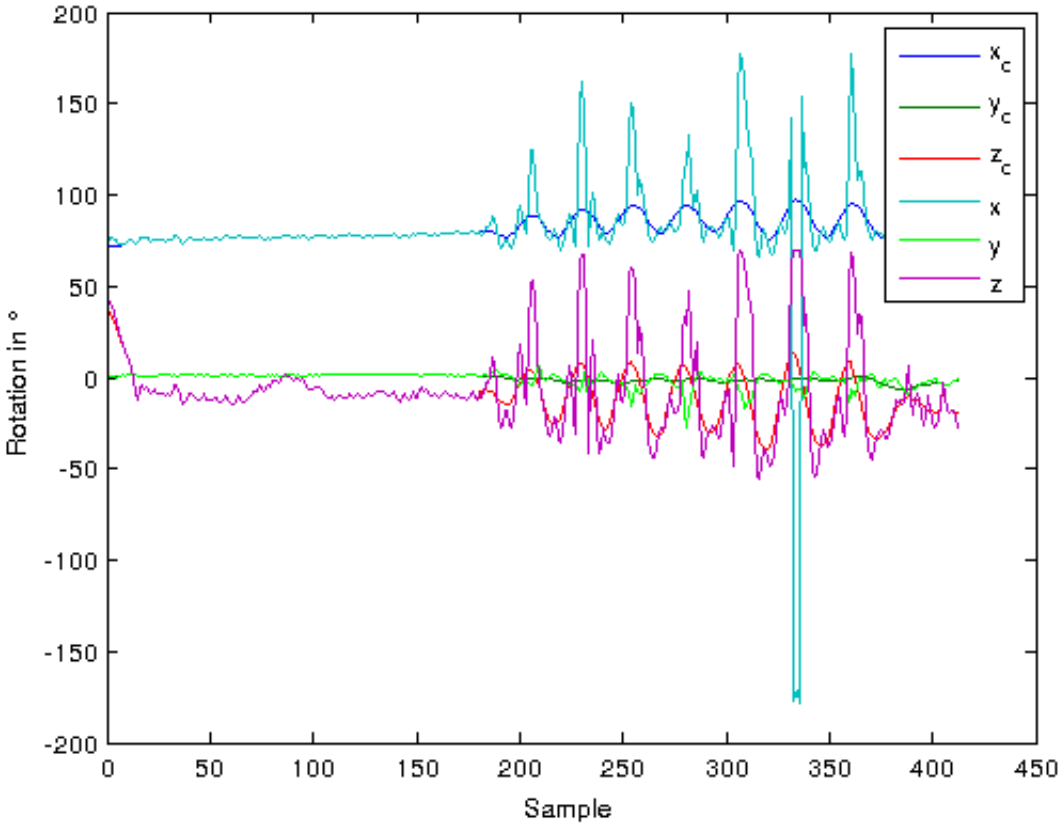

Abbildung 13: Unterschied korrigierte und nicht korrigierte Orientierung

Der Magnetfeldsensor ist der Sensor, welcher am meisten Probleme bereitet. Wie bereits zuvor erwähnt, wird er gerade innerhalb von Gebäuden durch jegliche elektrische Geräte und ferromagnetische Stoffe gestört. Ein Problem, welches häufig auftrat, war die plötzliche, starke Änderung der Magnetfeldmessungen auf den Achsen, sobald man das Tablet bewegt hatte. So lang das Tablet jedoch stabil gehalten wurde, gab es keine Veränderungen. Der Grund liegt vermutlich in den Kompensations- sowie Kalibrierungsalgorithmen, die Invensense in den MEMS-Sensor eingebaut hat. Diese werden verhindern, dass die Werte des Magnetfeldsensors sich ändern, so lang keine Bewegung in den anderen Sensoren festgestellt wird. Ein weiteres unerklärliches Problem war der plötzliche Totalausfall des Magnetfeldsensors. Totalausfall bedeutet hier, dass er so gut wie keine Änderung im Magnetfeld mehr feststellte und somit die Rotation um die z-Achse lediglich in einem kleinen Bereich stattfand. Hier half lediglich das Zurücksetzen des Tablets auf den Werkszustand. Auch hier kann nur vermutet werden, dass es ebenfalls an den implementierten Algorithmen von Invensense gelegen haben könnte und der Sensor in einen Fehlerzustand fiel. Auf die Algorithmen hat man jedoch keinen Einfluss bzw. kann man sie nicht ausschalten. Eine Korrektur für fehlerhafte Zustände des Magnetfeldsensors wurde in dieser Arbeit nicht verwirklicht. Im letzten Kapitel wird jedoch kurz auf Möglichkeiten eingegangen, diese zu bestimmen und ihnen entgegenzuwirken.

Zur Auswertung der Rotation aus Bildern wurden Tests in verschiedenen Szenarien durchgeführt. Da kein technisches Werkzeug zur genauen Messung vorhanden war, wurde auf einem Tisch ein 90-Grad-Winkel abgetragen und mit möglichst genauer Rotation des Tablets um diese 90 Grad gemessen. Die Szenarien waren ein eher strukturierter Hintergrund mit relativ vielen Kanten in der Szene, eine relativ Flache Szene mit wenigen Kanten, eine Szene mit viel Tiefe, also 3D-Informationen, und eine Mischung daraus. Die Szenen sind dabei normal ausgeleuchtet. Es wurden zusätzliche Tests mit verringerter Ausleuchtung durchgeführt. Die Messungen waren jedoch bei dieser Testmethode unbrauchbar. Die Gründe sind hierbei, dass die Ausleuchtung nicht so hinbekommen werden konnte, sodass bei den Messungen keine Situationen auftraten, bei denen die Rotationsschätzung aus Bildern zwischendurch nicht komplett ausfällt. Eine Wertung der Güte der Rotation ist jedoch nur möglich, wenn bei der Rotation um 90 Grad alle Zwischenschritte betrachtet werden konnten. Es fehlt sonst ein

Referenzwert, der die Qualität bestimmt. Mit Zeitsynchronisation und genauer Ausrichtung zu allen Zeiten könnte man dies umgehen. Das ist hier jedoch nicht möglich. Die Szenarien wurden mit schneller sowie langsamer Rotation getestet und das jeweils drei mal. Dabei wurden erreichte Orientierung sowie Anzahl der gefundenen Featurepunkte, Anzahl der Inliers und der durchschnittliche mittlere Pixelabstand jeden Zwischenschritts bestimmt. Die Ergebnisse sind in folgender Tabelle zusammengefasst. Die Featurepunkte sind aus Gründen der Performance auf maximal 325 beschränkt.

Szenario		Langsam	Schnell
3D	⌀ Rotation (in Grad)	90.68	87.98
	⌀ Inliers	230.24	110.49
	⌀ Featurepunkte	324.19	251.85
	⌀ mittlerer Abstand	1.78	6.1717
Mischung	⌀ Rotation	90.92	88.01
	⌀ Inliers	225.69	126.01
	⌀ Featurepunkte	325	264.73
	⌀ mittlerer Abstand	1.73	5.29
Flach, wenig Kanten	⌀ Rotation	88.31	86.27
	⌀ Inliers	146.39	103.21
	⌀ Featurepunkte	233.00	187.93
	⌀ mittlerer Abstand	2.33	3.39
Viel Struktur	⌀ Rotation	90.85	88.12
	⌀ Inliers	206.54	111.17

	⌀ mittlerer Abstand	324.73	212.32
	⌀ Rotation	2.074	2.38

Grundsätzlich ist zu sagen, dass für gute Rotationen, also mit kleinem Abstand zu 90 Grad, immer eine hohe, durchschnittliche Anzahl an Inliers sowie Gesamtzahl an Featurepunkten vorherrscht. Beispiele dafür sind die Tests „Mischung“, „3D“ und „viel Struktur“ bei langsamer Rotation. Das Szenario mit wenigen Kanten zeigt erwartungsgemäß bei langsamer Rotation die größte Abweichung und auch am wenigsten Inliers/Featurepunkte. Bei schnellen Rotationen sind die Abweichungen durchweg größer als bei langsamer Rotation. Dies geht einher mit verringerter Anzahl an Inliers und Featurepunkten. Ein Test auf Korrelation zwischen den Abweichungen der Rotation von 90 Grad und jeweils den Inliers und Featurepunkten ergab -0.835 und -0.883. Diese Werte zeigen, dass kleinere Differenzen mit mehr Inliers und Featurepunkten einhergehen. Das Ergebnis ist

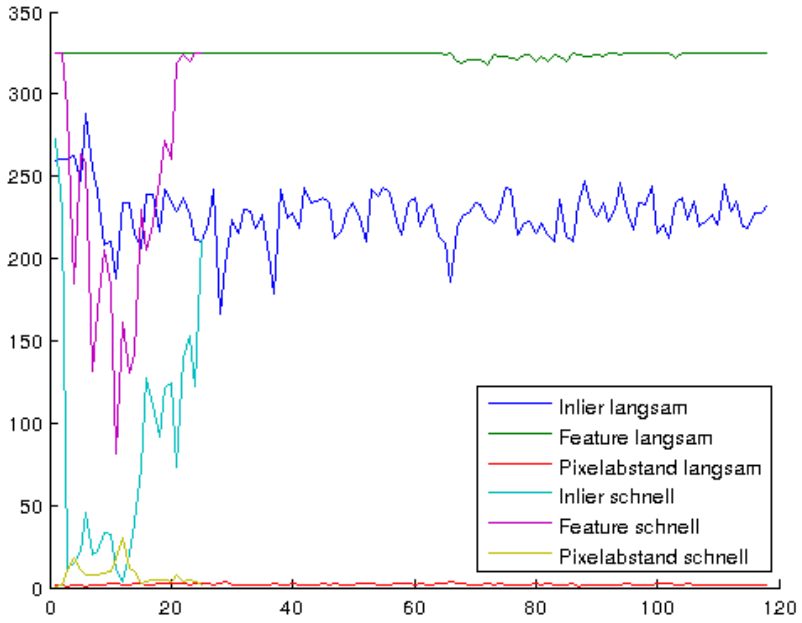

Abbildung 14: Vergleich zwischen langsamer und schneller Rotation

bei dieser kleinen Menge natürlich mit Vorsicht zu genießen, aber man kann

Tendenzen erkennen. In Abbildung 14 wird beispielhaft der Verlauf der Inlier und Featurepunkte gezeigt. Die Ergebnisse ähneln sich bei anderen Tests stark. Zu Beginn und Ende ist die Anzahl bei schneller Rotation auf Grund von langsameren Bewegungen hoch. Zwischendurch sinkt die Anzahl jedoch stark und die Gesamtrotation weicht stärker ab. Die Verläufe von Inliers und Featurepunkten sind sehr ähnlich. Bei langsamer Rotation und guten Ergebnissen sind die Werte etwa gleichbleibend. Aus der Anzahl der Featurepunkte und Inlier sollte sich eine Güte für die Messungen finden lassen. Die Betrachtung des Pixelabstandes zeigt ähnliche Ergebnisse, zu sehen in Abbildung 14. Bei guten Rotationen bleibt der Wert konstant niedrig. Bei gestörten Rotationen sind die Werte genau dann erhöht, wenn die Anzahl an Inliern und Featurepunkten sinken. Der mittlere Abstand könnte also zur Bestimmung der Qualität einer Rotation ebenfalls herangezogen werden. Eine Möglichkeit, wie dies geschehen könnte, wird im nächsten Kapitel zur Sensorfusion beschrieben.

Dann wurde noch überprüft, ob denn die Rotation aus Bilddaten möglicherweise driftet, also sich ein Fehler über die Zeit akkumuliert. Das ist zumindest über den kurzen Zeitraum von 2-3 Minuten nicht der Fall. Die Abbildung 15 zeigt, dass die Rotation immer wieder negativ und positiv bei keiner Bewegung abweicht, aber nicht driftet.

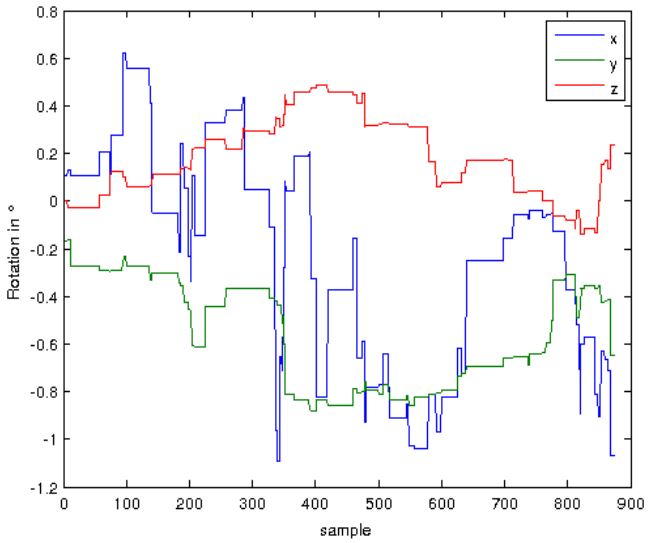

Abbildung 15: Verhalten der Rotation aus Bilddaten über die Zeit

An dieser Stelle wäre es nun interessant zu sehen, ob die Daten aus den verschiedenen Sensoren denn ähnliche Resultate erzielen. Wären alle Sensoren perfekt, dann lägen die errechneten Orientierungen immer an der gleichen Stelle. Getestet wurde ähnlich wie zuvor bei der Klassifizierung der Fehler der Bildrotationen. Aus verschiedenen Startpositionen wurde jeweils 90 Grad um die z-Achse rotiert und das jeweils schnell und langsam 3 mal. In der folgenden Tabelle sind die durchschnittlichen Abweichungen von 90 Grad festgehalten. Erwartungsgemäß ist der Gyroskopsensor durch die Selbstkalibrierung am verlässlichsten. Er zeigt durchweg die niedrigsten Abweichungen von 90 Grad. Außerdem scheint er nicht wirklich von schnelleren Rotationen beeinflusst zu werden. Die Abstände liegen in einem Bereich, der auch als Messfehler betrachtet werden kann. Die Rotation aus Bildern verhält sich ebenfalls erwartungsgemäß. Bei langsamer Rotation wird eine niedrige Abweichung erreicht, die sich bei schnellerer Bewegung erhöht. Die schlechtesten Ergebnisse erreicht der Gauß-Newton-Algorithmus, also die Orientierung aus Magnetometer und Accelerometer. Das liegt mit sehr hoher Wahrscheinlichkeit an dem relativ unzuverlässigen Magnetfeldsensor. Für das nächste Kapitel, also die Sensorfusion, bedeutet dies, dass der Quaternion aus dem Gauß-Newton-Algorithmus die kleinste Gewichtung bekommt, der Gyroskopsensor hingegen die größte.

Orientierung	Rotationsgeschwindigkeit	⌀ Differenz von 90 Grad
Gauß-Newton	Langsam	3.734
	Schnell	3.412
Gyroskop	Langsam	0.549
	Schnell	0.786
Bildinformationen	Langsam	0.869
	Schnell	2.449

6 Sensordatenfusion

In diesem Kapitel wird der Ansatz beschrieben, wie die Sensordaten fusioniert werden können, um ggf. bessere Ergebnisse zu erzielen. Das Vorgehen war dabei einen Kalman-Filter, der mit den klassischen Sensoren arbeitet, durch die Informationen aus den Bilddaten zu erweitern. Mit geeigneten Tests soll gezeigt werden, dass dieser erweiterte Filter Fehler weiter ausgleichen kann. Der Grund für die Verwendung eines Kalman-Filters ist, dass er bereits in der Vergangenheit oft gezeigt hat, dass er sich gut für die Fusion von Informationen aus verschiedenen Quellen, die unterschiedlich fehlerbehaftet sind, eignet.

6.1 Kalman-Filter

Der Kalman-Filter ist der optimale lineare Filter bzw. Schätzer, wenn es darum geht lineare Systeme mit ausschließlich gaußverteilten Fehlern zu modellieren. In diesem Fall minimiert er den mittleren quadratischen Fehler (MSE) für alle geschätzten Parameter optimal. Er besitzt einen Prädiktions- sowie Korrekturschritt, in dem die vorhergesagten Parameter mit Messwerten korrigiert werden.

An dieser Stelle wird der Kalman-Filter nicht genauer beschrieben. Er wurde in der Literatur bereits oft dokumentiert und kann z.B. hier [24] in Erfahrung gebracht werden.

Der hier verwendete Kalman-Filter basiert auf [25] und wurde um die Messung der Orientierung aus den Bilddaten erweitert.

6.1.1 Prädiktion

Der Zustandsvektor des Systems ist einfach der Quaternion, welcher die Orientierung bestimmt, denn genau dieser soll möglichst genau vorhergesagt und im nächsten Schritt durch die Messungen korrigiert werden.

$$x = \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix}$$

Zur Vorhersage der neuen Orientierung wird die Winkelgeschwindigkeit herangezogen, denn genau diese beschreibt die Änderung der Orientierung seit der letzten Messung. Mit Hilfe der Gleichung (4.1) lässt sich der vorhergesagte Zustand $x_{t|t-1}$ über die Winkelgeschwindigkeit bestimmen.

$$x_{t|t-1} = x_{t-1|t-1} + \dot{q}_{\omega,t}\,\Delta t = F \cdot x_{t-1|t-1} \qquad \textbf{(6.1)}$$

Durch Ausformulieren des Hamilton-Produkts erhält man folgendes:

$$x_{t|t-1} = \frac{1}{2}\begin{bmatrix} a - b\,\omega_x \Delta t - c\,\omega_y \Delta t - d\,\omega_z \Delta t \\ b + a\,\omega_x \Delta t + c\,\omega_z \Delta t - d\,\omega_y \Delta t \\ c + a\,\omega_y \Delta t - b\,\omega_z \Delta t + d\,\omega_z \Delta t \\ d + a\,\omega_z \Delta t + b\,\omega_y \Delta t - c\,\omega_x \Delta t \end{bmatrix}$$

Mittels Extraktion des Quaternions (a,b,c,d) aus dieser Gleichung erhält man die Zustandsübergangsmatrix F.

$$F = \begin{bmatrix} 1 & -\frac{1}{2}\omega_x \Delta t & -\frac{1}{2}\omega_y \Delta t & -\frac{1}{2}\omega_z \Delta t \\ \frac{1}{2}\omega_x \Delta t & 1 & \frac{1}{2}\omega_z \Delta t & -\frac{1}{2}\omega_y \Delta t \\ \frac{1}{2}\omega_y \Delta t & -\frac{1}{2}\omega_z \Delta t & 1 & \frac{1}{2}\omega_x \Delta t \\ \frac{1}{2}\omega_z \Delta t & \frac{1}{2}\omega_y \Delta t & -\frac{1}{2}\omega_z \Delta t & 1 \end{bmatrix}.$$

Zur Bestimmung der Fehlerkovarianz Q, also dem Maß des Fehlers bzw. Rauschens des Systems, werden die Achsen des Gyropskopsensors betrachtet, welche zur Vorhersage der Quaternion-Komponenten herangezogen werden. Aus F ergibt sich für Q daher folgende Gleichung:

$$Q = E\left[\begin{bmatrix} -\omega_x-\omega_y-\omega_z \\ \omega_x-\omega_y+\omega_z \\ \omega_x+\omega_y-\omega_z \\ -\omega_x+\omega_y+\omega_z \end{bmatrix}\begin{bmatrix} -\omega_x-\omega_y-\omega_z & \omega_x-\omega_y+\omega_z & \omega_x+\omega_y-\omega_z & -\omega_x+\omega_y+\omega_z \end{bmatrix}\right]$$

Wenn man davon ausgeht, dass der Erwartungswert der Winkelgeschwindigkeiten 0 ist, also $E[\omega_i]=0$, was bedeutet, dass der Gyroskopsensor sehr gut kalibriert ist und so gut wie keine Abweichungen bzw. keinen Bias besitzt, was hier der Fall ist, und außerdem die Winkelgeschwindigkeiten untereinander unkorreliert sind, also $E[\omega_i \omega_j]=0$, dann ergibt sich Q mit Hilfe der Varianzen der Winkelgeschwindigkeiten auf den Achsen wie folgt:

$$Q = \begin{bmatrix} \sigma_x^2+\sigma_y^2+\sigma_z^2 & -\sigma_x^2+\sigma_y^2-\sigma_z^2 & -\sigma_x^2-\sigma_y^2+\sigma_z^2 & \sigma_x^2-\sigma_y^2-\sigma_z^2 \\ -\sigma_x^2+\sigma_y^2-\sigma_z^2 & \sigma_x^2+\sigma_y^2+\sigma_z^2 & \sigma_x^2-\sigma_y^2-\sigma_z^2 & -\sigma_x^2-\sigma_y^2+\sigma_z^2 \\ -\sigma_x^2-\sigma_y^2+\sigma_z^2 & \sigma_x^2-\sigma_y^2-\sigma_z^2 & \sigma_x^2+\sigma_y^2+\sigma_z^2 & -\sigma_x^2+\sigma_y^2-\sigma_z^2 \\ \sigma_x^2-\sigma_y^2-\sigma_z^2 & -\sigma_x^2-\sigma_y^2+\sigma_z^2 & -\sigma_x^2+\sigma_y^2-\sigma_z^2 & \sigma_x^2+\sigma_y^2+\sigma_z^2 \end{bmatrix}$$

Denn da keine anderen Fehler auf Grund der Kalibrierung auftreten, ist lediglich die Varianz, das gaußverteilte Rauschen, für den Fehler verantwortlich.

Die Varianz bzw. Standardabweichung des hier verwendeten Gyroskops ist, wie in Kapitel 5 gezeigt, eigentlich so gut wie gar nicht vorhanden. Um dem Gyroskopsensor nicht ohne Fehler zu modellieren, wurde ein sehr kleiner Wert von 0.001 Rad für die Standardabweichung gewählt.

Die Zustandskovarianz $P_{t|t-1}$ definiert sich wie folgt:

$$P_{t|t-1} = F\,P_{t-1|t-1}\,F^T + Q \qquad \textbf{(6.2)}$$

Gleichung (6.1) und (6.2) definieren zusammen die Vorhersage des Zustands.

6.1.2 Korrektur

Im Korrekturschritt werden die Messungen des Zustands herangezogen, um den vorhergesagten Zustand zu korrigieren. In diesem Fall gibt es 2 Messungen, den Quaternion aus dem Gauß-Newton-Algorithmus sowie aus den Bildinformationen. Die Messung z_t ergibt sich also als Kombination der Quaternionen.

$$z_t = \begin{bmatrix} q_{gauss} \\ q_{image} \end{bmatrix} \quad .$$

Die Beobachtungsmatrizen H_{gauss} und H_{image} sind für beide Messungen die Einheitsmatrix I_4, denn beide Messungen entsprechen dem Zustandsvektor und sollen auch dementsprechend in die Messung eingehen. Damit die beiden Messungen kombiniert werden, schreibt man die Beobachtungsmatrizen übereinander, denn dadurch werden in den Berechnungen des neuen Zustands sowie Kalman-Gains K beide Messungen einbezogen. H ergibt sich also wie folgt:

$$H = \begin{bmatrix} H_{gauss} \\ H_{image} \end{bmatrix} = \begin{bmatrix} I_4 \\ I_4 \end{bmatrix}$$

Der neue Zustand bestimmt sich wie folgt:

$$x_{t|t} = x_{t|t-1} + K\left(z_t - H\,x_{t|t-1}\right)$$

mit dem Kalman-Gain K, das wie folgt definiert ist.

$$K = P_{t|t-1} H^T \left(H\,P_{t|t-1} H^T + R\right)^{-1}$$

Die neue Zustandskovarianz $P_{t,t}$ definiert sich durch

$$P_{t|t} = \left(I_4 - K\,H\right) P_{t|t-1} \quad .$$

Die Fehlerkovarianzmatrix der Messung R beschreibt dabei das Rauschverhalten der Messungen, also wie stark die einzelnen Messungen fehlerbehaftet sind. In der Literatur wird oftmals davon ausgegangen, dass die Messungen unkorreliert sind. Daher ergibt sich R wie in Gleichung (6.3). Nehmen wir zusätzlich wie in [1] und [2] an, dass die Fehler nur gaußverteilt sind, dann sind die Diagonalelemente der Matrix R die Varianzen der einzelnen Messungen. Diese werden experimentell für die genutzte Hardware bestimmt.

In Kapitel 5 wurden mögliche Fehlerzustände des Quaternions q_{image} gezeigt. An dieser Stelle soll eine einfache Mõglichkeit gezeigt werden, welche diese Fehler in R mit einbezieht. Nehmen wir an, dass bei guter Rotationsschätzung das Rauschen der Messung gaußverteilt ist. Daher entsprechen in diesen Fällen die Komponenten aus R, die für den Quaternion q_{image} zuständig sind, den Varianzen der Komponenten des Quaternions. Wenn einer der bestimmten Werte aus

Kapitel 5 von seinem Erwartungswert negativ abweicht, wird die Varianz um einen Faktor erhöht, der sich aus den gemessenen Werten und deren Erwartungswerten bestimmt. Wenn die Messung der Rotation aus Bildern komplett entfällt, dann werden die Diagonalelemente auf den Achsen der Fehlerkovarianzmatrix auf einen sehr hohen Wert gesetzt, damit diese Messung ignoriert wird.

$$R=\begin{bmatrix} r_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & r_{22} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & r_{33} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & r_{44} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & r_{55} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & r_{66} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & r_{77} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & r_{88} \end{bmatrix} \quad \textbf{(6.3)}$$

Die Erwartungswerte sowie der Faktor müssten noch näher bestimmt und getestet werden. Dies ist an dieser Stelle noch nicht geschehen.

Auch die Varianzen der Orientierungen q_{gauss} sowie q_{image} wurden noch nicht bestimmt. Da die Tests in Kapitel 5 zeigten, dass der Quaternion aus Magnetometer und Accelerometer schlechtere Ergebnisse aufweist als die Bildorientierung, wird ihm ein höherer Fehler zugeordnet. Für die Tests haben sich 0.05 * I_4 für q_{image} und 0.15 * I_4 für q_{gauss} als Fehlerkovarianzmatrizen als annehmbar erwiesen.

Zuletzt müssen der Zustandsvektor sowie die Zustandskovarianz noch initialisiert werden. Für den Zustandsvektor hat sich der Einheitsquaternion $q=(1,0,0,0)$ als gut herausgestellt. Für die Zustandskovarianz 2 * I_4. Der Kalman-Filter konvergiert so innerhalb weniger Sekunden gegen eine vernünftige Orientierung.

6.2 Ergebnisse

An dieser Stelle wurden noch ein mal Tests durchgeführt und Ergebnisse ermittelt. Die Testmethode hat sich dabei nicht verändert. Es wird 90 Grad um

die z-Achse notiert. Die Referenz ist ein abgetragener Winkel auf einem Tisch. Daher sind Messfehler möglich. Um diese zu vermeiden, werden verschiedene Szenarien getestet. Im Grunde bedeutet dies, dass in die verschiedenen Himmelsrichtungen getestet wird. Dabei dominieren unterschiedliche Eigenschaften des Raumes wie in Kapitel 5 beschrieben. Der Kalman-Filter wurde zusätzlich ohne Bildinformationen implementiert. Er dient zum Vergleich.

Die Ergebnisse sind in folgender Tabelle dargestellt. Bestimmt wurde erneut der durchschnittliche Abstand zu 90 Grad für schnelle sowie langsame Rotation.

Orientierung	Rotationsgeschwindigkeit	⌀ Differenz von 90 Grad
Gauss-Newton	Langsam	2.929
	Schnell	2.841
Gyroskop	Langsam	0.621
	Schnell	0.513
Kalman-Filter	Langsam	1.504
	Schnell	0.992
Kalman-Filter mit Bild	Langsam	0.911
	Schnell	0.573
Bildinformationen	Langsam	0.842
	Schnell	1.77

Zunächst kann man sagen, dass die Ergebnisse aus Kapitel 5 bestätigt wurden. Das Verhältnis zwischen Gyroskop, Rotationsschätzung aus Bildern sowie Gauß-Newton ist immer noch in etwa gleich. Der Gyroskopsensor ist am verlässlichsten, die Orientierung aus Magnetometer und Accelerometer zeigt größere Abweichungen. Die Kalman-Filter arbeiten beide besser als die Orientierungen des Gauß-Newton-Algorithmus sowie aus Bildinformationen. An das Gyroskop kommen die Ergebnisse jedoch nicht heran. Der Grund liegt

hierbei bei dem so gut wie ungestörten Gyroskopsensor. Das ist natürlich nicht bei jedem Mobilgerät der Fall. Für zukünftige Tests sollte man noch andere Geräte testen.

Der Vergleich zwischen den Kalman-Filtern fällt zugunsten des bildgestützten Kalman-Filters aus. Dieses Ergebnis war von vornherein zu erwarten, denn die Bildinformationen zeigen bessere Ergebnisse als Magnetometer und Accelerometer. Die schlechten Messungen des Magnetometers haben daher bei dem bildgestützten Kalman-Filter einen kleineren Einfluss auf die Orientierung. Ein weiterer, interessanter Aspekt ist der Fakt, dass die Ergebnisse bei beiden Kalman-Filtern bei schneller Rotation besser sind. Mit großer Wahrscheinlichkeit liegt es daran, dass die schlechten Messungen des Magnetfeldsensors weniger Zeit haben negativen Einfluss auf die Orientierung zu nehmen.

Zu weiteren Testzwecken wurden die Messungen des Magnetometers zufällig mit Abweichungen zwischen -20 und 20 gestört. Eine beispielhafte Darstellung des Verlaufs der Orientierung wurde in Abbildung 16 dargestellt. Der Kalman-Filter mit Bild weicht weniger von der Gyroskop-Orientierung ab als der ohne. Dieser Test ist jedoch mit Vorsicht zu betrachten. Solche großen Fehler des Magnetfeldsensors sollten von vornherein ignoriert werden. Im letzten Kapitel wird darauf kurz eingegangen.

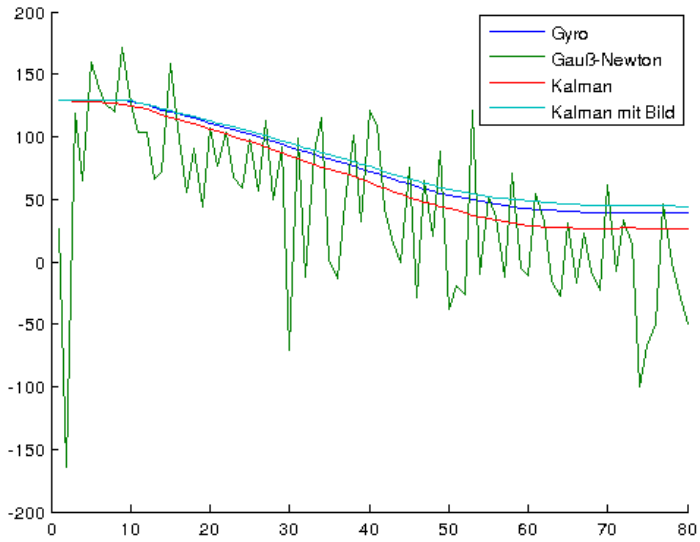

Abbildung 16: Verlauf der Orientierung mit gestörtem Magnetfeldsensor

7 Implementierung

Die Implementierung fand, wie bereits mehrfach erwähnt, auf einem Android-Tablet statt. Im Normalfall findet Programmierung unter Android mit Java statt. In dieser Arbeit wurde dies komplett vermieden, zum größten Teil auf Grund von erhöhter Performance unter c++. An dieser Stelle soll auf die besondere Vorgehensweise sowie Probleme eingegangen werden, die bei der Implementierung auftraten.

7.1 Vorgehen

Zunächst wurde sich mit den Sensoren des Tablets beschäftigt. Dabei entstand eine kleine App, welche die aktuellen Werte der Sensoren ausliest und den Verlauf grafisch darstellt. Die Implementierung fand zu dem Zeitpunkt in der normalen Java-Entwicklungsumgebung für Android statt. Die Verwendung der Sensoren stellte sich als relativ simpel heraus. Man findet alle Informationen dazu auf den Entwicklerseiten von Android[6].

Von Anfang an stand fest, dass OpenCV zur Implementierung der Algorithmen herangezogen werden soll. OpenCV ist eine Programmbibliothek, die hauptsächlich Algorithmen im Bereich Computergrafik in einem großen Projekt vereint. Die Gründe der Verwendung lagen hauptsächlich an dem breiten Spektrum an Featurepunkt-Algorithmen, Verfügbarkeit auf Android sowie optimierten Matrizenoperationen, die OpenCV bietet. Die Einbindung von OpenCV in normale Android-Projekte ist ebenfalls einfach und wird auf deren Internetpräsenz erklärt. Das Problem bei dieser Verwendung ist, dass die Nutzung der OpenCV-Klassen und -Algorithmen immer wieder den Aufruf des Java Native Interface JNI benötigt, da die Algorithmen OpenCVs in c++ bzw. c geschrieben sind. Das JNI ermöglicht eben diesen Aufruf von Funktionen, die in anderen „nativen“ Sprachen geschrieben wurden. Dieser Aufruf kostet Zeit und sollte daher vermieden werden. Dieses Problem kann man umgehen. Es besteht die

6 http://developer.android.com/guide/topics/sensors/sensors_overview.html

Möglichkeit am Start einer Android-App eine sogenannte Android Native-Activity[7] zu starten. Das besondere ist, dass diese vollständig in nativem Programmcode geschrieben ist und trotzdem dem Lebenszyklus von Android-Apps [27] folgt. Zur Verwendung dieser benötigt man ebenfalls das Android Native Development Kit NDK, welches die APIs bietet, um mit anderen Programmiersprachen zu arbeiten. An dieser Stelle gab es 2 Probleme. Über das NDK gibt es keinen Zugriff auf die Kamera sowie keine Menüs, um ggf. Einstellungen zu tätigen. Die Lösung des ersten Problems wird im Teil Probleme behandelt. Das zweite wird hier behandelt, denn es gehört implizit zum Vorgehen.

Es wurde also nach einer Lösung gesucht, Menüs unter Android zu nutzen sowie gleichzeitig möglichst mit c++ zu programmieren. Der Zufall ergab, dass die Firma Digia zu diesem Zeitpunkt damit begonnen hatte, das Qt-Framework auch für mobile Betriebssysteme, darunter Android, verfügbar zu machen. Qt ist ein in c++ programmiertes Framework, welches die Möglichkeit bietet Oberflächen auf verschiedene Weisen zu implementieren. Zunächst musste dieses noch selbst kompiliert werden, mit Version 5.1 gab es jedoch vorkompilierte Pakete. Die Version 5.1.1 wird aktuell in dieser Arbeit verwendet. Die Unterstützung von Android war in dieser Version jedoch noch vorläufig. Welche Probleme dies mit sich brachte, wird im nächsten Kapitel besprochen.

An dieser Stelle wird beschrieben, wie man mit Qt unter Android programmieren kann und dabei OpenCV einbindet. Dazu benötigt man das Android SDK, Android NDK , Qt, Apache Ant sowie OpenCV. Das Vorhandensein von Java sowie das offensichtliche Entpacken der zuvor genannten Pakete werden vorausgesetzt. Mit dem Android SDK muss die gewünschte Android API, also Version des Betriebssystems, heruntergeladen werden. Dies geschieht über den Android SDK Manager, der im Android SDK enthalten ist. Andere Vorbereitungen sind nicht nötig. Anschließend müssen im Qt-Creator, der Entwicklungsumgebung von Qt, die oben genannten Pakete hinzugefügt werden. Dies geschieht über die Einstellungen und dem dortigen Unterpunkt Android. Damit ist man bereit, mit der Qt-Programmierung unter Android zu starten. Man startet ein neues Projekt und wählt dort Android sowie die gewünschte Toolchain aus. Die anschließende Programmierung unterscheidet sich im Vergleich zur Desktop-Programmierung nicht. Auch die Einbindung von OpenCV war einfacher als gedacht. Qt besitzt für

7 http://developer.android.com/reference/android/app/NativeActivity.html

Projekte eine Projekt-Datei, welche Informationen beinhaltet, die für die Kompilierung benötigt werden. In dieser muss auf die Bibliotheken sowie Header-Dateien von OpenCV verlinkt werden, damit diese einbezogen werden. Der Qt-Creator bietet dafür eine einfache Möglichkeit. In der Projekt-Datei kann man über den Rechtsklick Bibliotheken hinzufügen. Dort fügt man einfach die Bibliotheken sowie dazugehörige Header-Dateien von OpenCV hinzu. Damit kann OpenCV normal verwendet werden.

Die grundsätzlichen Erfahrungen mit Qt waren positiv. Obwohl es sich noch um einen vorläufigen Support für Android handelte, funktionierten die meisten Dinge bereits, insbesondere die Menüstrukturen waren gut integriert. Der Vorteil der Verwendung von Qt ist die einfache Portierbarkeit, wenn man größtenteils nur Konstrukte dieser Bibliothek benutzt.

Nachfolgend noch ein paar Dinge, die an dieser Stelle erwähnenswert sind. Die implementierten Algorithmen dieser Arbeit nutzen alle Standard c++ sowie OpenCV-Konstrukte, insbesondere die OpenCV-Klasse cv::Mat, welche für jegliche Operationen auf Matrizen genutzt wurde. Basierend auf der Klasse Mat entstand ebenfalls eine Quaternionen-Klasse, die verschiedene Operationen auf diesen bietet.

Eine Besonderheit der OpenCV-Bibliothek ist eine gesonderte Version für Nvidia Tegra-Chips. Im verwendeten Tablet befindet sich ein Tegra 3 Chip. Diese Version beinhaltet teilweise speziell auf diese Chips optimierte Algorithmen, sodass ein wenig mehr Performance aus dem Algorithmus zur Bestimmung der Rotation aus Kamerabildern herausgeholt werden konnte.

7.2 Probleme

Das größte Problem war die Kamera mit Qt anzusprechen. Eigentlich verspricht Qt die Unterstützung eben dieser. In Version 5.1.1 war diese aber noch nicht funktionstüchtig. Mit der aktuell verfügbaren Version 5.2[8] soll dies behoben sein. Der Umstieg wurde noch nicht getätigt. Daher musste die Funktionalität wiederhergestellt werden. Dies wurde mit OpenCV erreicht. Dafür wurde einfach die Kamera-Klasse genutzt, welche aus OpenCV verfügbar ist. Diese implementiert

8 http://qt-project.org/wiki/New-Features-in-Qt-5.2

dabei eine Android API, die so gut wie nicht dokumentiert ist und sich bei verschiedenen Herstellern unterscheiden kann. Die Funktionalität der Kamera ist also nicht immer gegeben, gerade wenn Hersteller eigene Anpassungen vornehmen. Auf dem Asus TF700T gab es keine Probleme. Ein kleiner Trick war trotzdem noch nötig, um die Kamera zu nutzen. OpenCV nutzt eigene Bibliotheken für die Kamera unter Android. Das Einbinden in der Projekt-Datei funktionierte nicht. Qt erstellt jedoch bei der Kompilierung des Projekts einen Unterordner mit Namen „Android". Dieser enthält die kompilierten Libraries, die App sowie andere Dinge. Zu den kompilierten Libraries fügt man die Kamera-Bibliothek einfach nachträglich hinzu. Beim nächsten Kompilieren wird diese mit in die App kopiert und steht dem Programm zur Verfügung. Auf diese Weise ist die Kamera jedoch stark abhängig von der genutzten Version von Android und funktioniert, wie bereits erwähnt, nicht auf jedem Gerät.

Ein weiteres Problem war die Darstellung der Kamerabilder mit Qt. Die Bilder aus OpenCV, die in Matrizen gespeichert werden, sind nicht direkt kompatibel mit den Bildcontainern aus Qt, lassen sich jedoch recht einfach umwandeln, indem man einfach nur auf die gespeicherten Bilddaten der Matrix verweist und damit ein Qt-Bild unter Angabe des genutzten Formats erstellt. Dieses lässt sich anschließend in Qt darstellen.

Qt bietet eigentlich auch die Möglichkeit Sensoren eines Mobilgeräts anzusprechen. Diese Funktion war jedoch noch nicht zufriedenstellend implementiert, denn manche Sensoren funktionierten einfach nicht. Die Nutzung der Sensoren wurde mit Hilfe des NDKs[7] erreicht.

Das letzte erwähnenswerte Problem entstand durch die Verwendung von Threads. Diese wurden genutzt, um die Aufgaben auf die einzelnen Kerne des Tablets zu verteilen. Aktuell sind es 3 Threads, einer für die Kamerazugriffe sowie Rotationsschätzung, einer für den Sensorzugriff und dem Gauß-Newton-Algorithmus und der dritte für die Fusion der Sensordaten. Die Threads sind ebenfalls hilfreich die Qt-Oberfläche ansprechbar zu halten. Berechnungen im Mainthread, der hauptsächlich für die Oberfläche zuständig ist, erhöhen die Ansprechzeit. Das Problem wiederum ist, dass nur aus dem Mainthread die Oberfläche verändert werden kann. Versucht man auf Menüobjekte aus anderen Threads zuzugreifen, wird der Zugriff verwehrt. Um

dies zu umgehen wurde das Qt-Konzept Signals & Slots verwendet. Ein Thread sendet ein Signal aus, das ebenfalls Objekte mit sich tragen kann, und ein Slot, also eine Funktion, eines anderen Threads empfängt dieses und agiert dementsprechend. Das wird insbesondere für die Aktualisierung der Kamerabilder in der Oberfläche genutzt.

8 Zusammenfassung und Ausblick

In dieser Arbeit wurde eine Möglichkeit aufgezeigt, die verschiedenen Sensoren eines heutigen mobilen Endgeräts zu vereinigen, um ein besseres und stabileres Ergebnis bei der Orientierungsbestimmung zu erzielen. Die Analyse der Bildinformationen ergab, dass diese sich zur Fusion eignen. Der vorgestellte Kalman-Filter ist in der Lage die Sensoren gut zu fusionieren und Fehler in den Messdaten zu korrigieren. Die Verwendung von Qt zur Programmierung auf Android-Geräten hat sich als eine gute Alternative herausgestellt, gerade wenn man Programmierung in c++ gewöhnt ist. Es existierten noch Probleme an der ein oder anderen Stelle, aber diese waren alle lösbar.

Einige Aspekte blieben jedoch noch offen, zum Beispiel die Analyse der Messdaten im Hinblick auf ihre Fehlermatrizen im Kalman-Filter. Die Modellierung benutzt aktuell relativ einfach bestimmte Erfahrungswerte. Es wurde jedoch bereits beschrieben, was dazu noch getan werden muss.

Ein Aspekt, der in dieser Arbeit gar nicht betrachtet wurde, war die vorzeitige Erkennung von Fehlern in den Magnetfeldsensordaten. Auf diese Weise ließen sich ggf. die Fehler in Zukunft weiter minimieren. Der Hersteller von Sensoren Invensense [7] gibt z.B. an, dass die Norm des Magnetfeldvektors immer zwischen 25 und 65 liegen sollte. Bei Abweichungen könnte man die Sensordaten ignorieren. Einen anderen Ansatz liefert Angelo M. Sabatini in [4]. Dort werden Magnetfeldmessungen ignoriert, wenn der gemessene Inklinationswinkel des Magnetfeldes, also der Neigungswinkel des lokalen Erdmagnetfeldes zur Horizontalen, zu stark vom eigentlichen lokalen Inklinationswinkel abweicht. Ein interessanter, relativ neuer Ansatz zur Lösung des Problems der Orientierungsfusion, ebenfalls von A. M. Sabatini, wurde in [6] vorgestellt. Dieser verwendet zur Bestimmung der Orientierung die gleichen Sensoren wie in dieser Arbeit. Die Ergebnisse sind vielversprechend und könnten helfen den hier vorgestellten Kalman-Filter weiter zu verbessern.

Quellenverzeichnis

[1] Xiaoping Yun, Conrado Aparicio, Eric R. Bachmann und McGhee Robert B.: Implementation and Experimental Results of a Quaternion-Based Kalman-Filter for Human Body Motion Tracking; Proceeding of the 2005 IEEE Internation Conference on Robotics and Automation; S. 317-322; Barcelona, Spain; April 2005

[2] Xiapoing Yun and Eric R. Bachmann: Design, Implementation, and Experimental Results of a Quaternion-Based Kalman Filter for Human Body Motion Tracking; IEEE Transactions on Robotics, Vol. 22, No. 6; S. 1216-1226; Dezember 2006

[3] João Luís Marins, Xiaoping Yun, Eric R. Bachmann, Robert B. McGhee und Michael J. Zyda: An Extended Kalman Filter for Quaternion-Based Orientation Estimation Using MARG Sensors; Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems; S. 2003-2011; Maui, Hawaii, USA, Okt. 29. - Nov. 03, 2001

[4] Angelo M. Sabatini: Quaternion-Based Extended Kalman Filter for Determing Orientation by Intertial and Magnetic Sensing; IEEE Transactions on Biomedical Engineering, Vol. 53, No. 7; S. 1346-1355; Juli 2006

[5] Angelo M. Sabatini: Kalman-Filter-Based Orientation Determination Using Inertial/Magnetic Sensor: Observability Analysis and Performance Evalutation; Sensors 2011, 11; S. 9182-9206; ISSN 1424-8220

[6] Gabriele Ligorio und Angelo M. Sabatini: Extended Kalman Filter-Based Methods for Pose Estimation Using Visual, Inertial and Magnetic Sensors: Comparative Anaylsis and Performance Evaluation; Sensors 2013, 13; S. 1919-1941; ISSN 1424-8220

[7] Invensense Inc.; Motion Sensor Introduction; 2011; http://invensense.com/mems/gyro/documents/whitepapers/1Sensor%20Introduction.pdf

[8] PRIME Faraday Partnership: An introduction to MEMS; Loughborough University; 2002; ISBN 1-84402020-7

[9] Oliver Nehrig: Entwurf und Realisierung eines Beschleunigungssensorsystem auf der Basis von in Silizium integrierter Mikromechanik für die besonderen Anforderungen bei Schwerlasthandhabungssystemen; Gerhard-Mercator-Universität – Gesamthochschule Duisburg; 2003

[10] Andrea Candini und Marco Affronte: Magnetometry by means of Hall micro-probes in the Quantum Design PPMS; S3 National Center on Nanostructures and Biosystem; 27. Febuar 2008

[11] Vibrating structure gyroscope; Wikipedia; http://en.wikipedia.org/wiki/Vibrating_structure_gyroscope

[12] Claudia C. Meruane Naranjo: Analysis and Modeling of MEMS based Intertial Sensors; School of Eletrical Engineering Kungliga Tekniska Hgskolan; Stockholm, Schweden, 2008

[13] Joe Seer, Martin Lim und Steve Nasiri: Development of High-Performance, High-Volume Consumer MEMS Gyroscopes; http://invensense.com/mems/gyro/documents/whitepapers/Development-of-High-Performance-High-Volume-Consumer-MEMS-Gyroscopes.pdf

[14] David Fleet and Aaron Hertzmann: Camera Models; 2005; http://cs.brown.edu/~ls/teaching08/LN06_Camera.pdf

[15] Alvin Quach: Complementary Metal-Oxide Semiconductor Sensors; University of California Santa Barbara; 11. März 2010; http://www.writing.ucsb.edu/faculty/holms/technology_report_Alvin%20Quach2.pdf

[16] Peter Eisert: Model-based Camera Calibration Using Analysis by Synthesis Techniques; Proc. International Workshop on Vision, Modeling, and Visualization; Erlangen, Deutschland; S. 307-314; November 2002

[17] Sebastian O.H. Madgwick: An Efficient Orientation Filter for inertial and intertial/magnetic sensor arrays; 30. April 2010; http://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf

[18] Ethan Rublee, Vincent Rabaud, Kurt Konolige und Gary Bradski: Orb: an efficient alternative to SIFT and SURF; Computer Vision (ICCV), 2011 IEEE International Conference; S. 2564-2571; Barcelona, Spanie; 6.-13. November 2011

[19] Jay A. Farrell: Computer of the Quaternion from a Rotation Matrix; University of California; 20 April 2008

[20] Matthew Brown, Richart I. Hartley und David Nistér: Minimal Solutions for Panoramic Stitching; International Conference on Computer Vision and Pattern Recognition (CVPR2007); Minneapolis, USA; Juni 2007

[21] K. S. Arun, T. S. Huang und S. D. Blostein: Least-Squares Fitting of Two 3-D Point Sets; IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. Pami-9, No. 5; S. 698-700; September 1987

[22] Richard Szeliski: Image Alignment and Stitching: A Tutorial; Foundations and Trends in Computer Graphics and Vision, Vol. 2, No 1; S. 1-104; 2006

[23] Johannes Furch: System zur Erstellung von Panoramafotografien; Eberhard Karls Universität Tübingen; 6. Mai 2010

[24] Greg Welch und Gary Bishop: The Kalman Filter; http://www.cs.unc.edu/~welch/kalman/

[25] Daniele Comotti und Michele Ermidoro: Progetto di Microelettronica Sviluppo di algoritmi per la stima dell'orientamento di un sensore inerziale; Università degli studi di Bergamo; 29. März 2011
http://9dof-orientation-estimation.googlecode.com/files/ProgettoMicroelettronica_Relazione_V16.pdf

[26] Daniele Comotti und Michele Ermidoro: Report of the Course „Progetto di Microelettronica"; Università degli studi di Bergamo; 16. Mai 2011
https://9dof-orientation-estimation.googlecode.com/files/MicroelectronicsProject_Report_V11.pdf

[27] Managing the Activity Lifecycle:
http://developer.android.com/training/basics/activity-lifecycle/index.html

[28] Android SDK: http://developer.android.com/sdk/index.html

[29] Android NDK: https://developer.android.com/tools/sdk/ndk/index.html

[30] Apache ANT: http://ant.apache.org/bindownload.cgi

[31] Qt 5.1.1 Download: http://download.qt-project.org/official_releases/qt/5.1/5.1.1/

[32] OpenCV API Reference: http://docs.opencv.org/modules/refman.html

[33] Android Koordinatensystem – Abbildung 1:
http://developer.android.com/images/axis_device.png

[34] Beschleunigungssensor: Abbildung 2:
http://www.daviddarling.info/encyclopedia/A/accelerometer.html

[35] Hall-Effekt des Magnetometers - Abbildung 4:
http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/hall.html

[36] Tuning-Fork-Vibrationskreisel – Abbildung 5:

http://electroiq.com/blog/2010/11/introduction-to-mems-gyroscopes/

[37] Crosscheck-Veranschaulichung – Abbildung 9:
http://stackoverflow.com/questions/11181823/why-we-need-crosscheckmatching-for-feature/11484935#11484935