Egomotion from Optical Flow

Manuel Bühler

7. Juni 2016

Bericht zum Lab Course am Lehrstuhl für Mustererkennung und Bildverarbeitung
der Albert Ludwigs Universität Freiburg.
Betreuer: Benjamin Ummenhofer

Inhaltsverzeichnis

1 Einführung

Die Aufgabe in diesem Lab Course bestand darin, aus dem gegebenen optischen Flussfeld einer Bildsequenz einer statischen Szene die Eigenbewegung der Kamera, die diese Bildsequenz aufgenommen hat, zu schätzen. Dazu sollten verschiedene Algorithmen näher betrachtet werden.

Im ersten Teil der Arbeit sollte der robuste 8-Punkte Algorithmus aus [8] implementiert und getestet werden. Dazu wurde zuerst der Standard 8-Punkte Algorithmus implementiert und anschließend um Gewichtung, Data Normalization und Rank Enforcement erweitert, so dass er robust gegenüber Ausreißern ist. Damit kann er nicht nur auf ideale, synthetische Bildsequenzen angewendet werden, sondern auch auf Bilder aus der Realität, die immer mit Ausreißern behaftet sind.

Die zweite Aufgabe bestand darin, die Performance des implementierten robusten 8-Punkte Algorithmus mit der „Direct Optimization of Frame-to-Frame Rotation“ aus [5] zu vergleichen. Dazu wurden zwei verschiedene Szenen benutzt, eine mit synthetischem Bildmaterial und eine mit realem Bildmaterial.

1.1 Optischer Fluss

Als Ausgangsmaterial standen neben den einzelnen Bildern auch die optischen Flussfelder zwischen aufeinander folgenden Bildern zur Verfügung. Ein optisches Flussfeld besteht aus einer Menge von Flussvektoren. Jeder dieser Flussvektoren gibt an, welche relative Bewegung ein bestimmter Punkt vom ersten zum zweiten Bild vollzogen hat („Wie hat sich aus Sicht der Kamera ein Punkt in der Zeit zwischen zwei Bildaufnahmen bewegt?“).

Ausgangspunkt für die Algorithmen zur Berechnung der Eigenbewegung der Kamera sind Punktkorrespondenzen von aufeinander folgenden Bildern. D.h. für jedes Paar von aufeinander folgenden Bildern muss zu jedem Punkt p im ersten Bild der korrespondierende Punkt p' im zweiten Bild bekannt sein. Diese Punktkorrespondenzen können aus dem gegebenen optischen Fluss durch einfache Addition berechnet werden:

$$p' = p + v$$

Dabei ist v der Flussvektor vom ersten zum zweiten Frame im Punkt p.

Die Berechnung der optischen Flussvektoren war nicht Teil dieser Arbeit; die Vektoren standen bereits für jeden Bildpunkt in allen Frames zur Verfügung. Die Berechnung der Flussvektoren wird in [6] und [1] behandelt.

1.2 Epipolargeometrie

Die Epipolargeometrie beschreibt die relative Pose, d.h. Lage und Ausrichtung, von zwei Kameras zueinander sowie deren interne Parameter. Ist die relative Pose bekannt, so erhält man durch die Projektion einer Projektionslinie eines Punktes x von Kamera 1 in die Bildebene von Kamera 2 eine sogenannte Epipolarlinie l' (siehe Abb. 1). Die Epipolarlinie ist die Linie, auf der der korrespondierende Punkt x' liegen muss. Diese Abbildung eines Bildpunktes x in Kamera 1 auf die zugehörige Epipolarlinie l' in Kamera 2 wird durch die Fundamentalmatrix ausgedrückt:

$$l' = Fx$$

Mit Hilfe der Fundamentalmatrix wird die Epipolargeometrie vollständig beschrieben. [3]

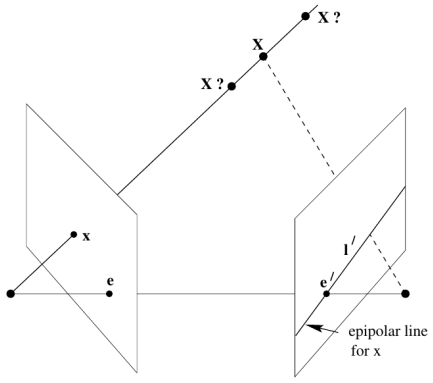

Abbildung 1: Projiziert man eine Projektionslinie in die Bildebene der anderen Kamera, so erhält man die Epipolarlinie. Quelle: R. Hartley, A. Zisserman [3]

2 Verwendete Methoden

Im Folgenden wird die Funktionsweise der verwendeten Methoden zur Eigenbewegungsschätzung erläutert. Die Performances der einzelnen Algorithmen werden in Abschnitt 6 verglichen.

Das Ziel der Methoden ist es jeweils, mit Hilfe bekannter Punktkorrespondenzen zwischen zwei aufeinanderfolgenden Frames die Eigenbewegung der Kamera zu berechnen. Da die Punktkorrespondenzen einer realen Szene fehlerbehaftet sein können, wird eine Annäherung berechnet, so dass der Fehler möglichst klein wird. Die Eigenbewegung wird schließlich durch eine Kombination aus Rotation und Translation ausgedrückt.

2.1 8-Punkte Algorithmus

Der 8-Punkte Algorithmus wird in [8] genau beschrieben. Seine Funktionsweise basiert im Wesentlichen auf der Epipolargeometrie (siehe Abschnitt 1.2). Ist ein Punkt $\tilde{\mathbf{x}} = (x, y, 1)^T$ in einem Frame und der dazu korrespondierende Punkt $\tilde{\mathbf{x}}' = (x', y', 1)^T$ (Punkte in homogenen Koordinaten) im darauf folgenden Frame bekannt, so kann mit Hilfe der Epipolargeometrie die Formel [2]

$$0 = \tilde{\mathbf{x}}'^T F \tilde{\mathbf{x}} = \mathbf{s}^T \mathbf{f} \tag{1}$$

aufgestellt werden mit dem Vektor

$$\mathbf{s} = (xx', yx', x', xy', yy', y', x, y, 1)^T$$

und der Fundamentalmatrix in Vektorform

$$\mathbf{f} = (f_{1,1}, f_{1,2}, f_{1,3}, f_{2,1}, f_{2,2}, f_{2,3}, f_{3,1}, f_{3,2}, f_{3,3})^T.$$

Die Fundamentalmatrix F ist eine 3×3 Matrix. Ihre 9 Werte sind bis auf einen Skalierungsfaktor definiert. Somit sind 8 Punktkorrespondenzen ausreichend, um die Gleichung (1) lösen zu können. [8]

Da die Punktkorrespondenzen nicht exakt sind, wird das Ergebnis genauer, wenn mehr als 8 Korrespondenzen zur Schätzung der Fundamentalmatrix verwendet werden. Dazu wird nun die folgende Energie minimiert [8]:

$$E(\mathbf{f}) = \sum_{i=1}^{n} (\mathbf{s}_i^T \mathbf{f})^2 = \|S\mathbf{f}\|^2 \tag{2}$$

Dabei ist S eine $n \times 9$ Matrix, die aus n Reihenvektoren $\mathbf{s}_i^T$ mit $1 \leq i \leq n$ besteht.

Die Energie (2) kann nun minimiert werden indem eine Lösung mit der Methode der kleinsten Fehlerquadrate für die Gleichung $S\mathbf{f} = \mathbf{0}$ gesucht wird. [8]

Dieses Problem wurde mit Hilfe der Singulärwertzerlegung der Matrix S gelöst. Dabei ist derjenige Singulärvekor mit dem kleinsten Singulärwert die Lösung für $\mathbf{f}$. [3]

2.2 Robuster 8-Punkte Algorithmus

Da die vom Algorithmus benötigten Punktkorrespondenzen in der Praxis nicht perfekt sind, werden verschiedene Techniken angewendet, um den Algorithmus möglichst robust gegenüber Ausreißern zu machen. Ausreißer können insbesondere durch Rauschen, Verdeckungen und Beleuchtungsänderungen entstehen [8].

2.2.1 Gewichte

Um den Einfluss, den Ausreißer auf die Berechnungen haben, zu verringern, werden Gewichte eingeführt. Diese Gewichte werden für Ausreißer sehr klein und für Nicht-Ausreißer groß. Dadurch werden Ausreißer weniger bzw. Nicht-Ausreißer mehr berücksichtigt.

Die Energiefunktion lautet nun [8]

$$E(\mathbf{f}) = \sum_{i=1}^{n} \Psi((\mathbf{s}_i^T \mathbf{f})^2). \tag{3}$$

Dabei ist $\Psi(s^2)$ die L_1-Norm

$$\Psi(s^2) = \sqrt{s^2 + \varepsilon^2} \tag{4}$$

mit der Konstanten $\varepsilon = 10^{-3}$.

Dies führt schließlich zum nichtlinearen Problem [8]

$$\mathbf{0} = (S^T W(\mathbf{f}) S - \lambda I)\mathbf{f}, \tag{5}$$

wobei W eine $n \times n$ Diagonalmatrix ist mit den positiven Gewichten $w_{i,i} = \Psi'((s_i^T \mathbf{f})^2)$. Dieses Problem wird gelöst indem zuerst mit dem 8-Punkte Algorithmus ein Startwert für $\mathbf{f}$ berechnet wird (siehe Abschnitt 2.1). Dann werden die Gewichte berechnet mit

$$w_{i,i} = \Psi'((s_i^T \mathbf{f})^2) = \frac{0,5}{\sqrt{s^2 + \varepsilon^2}}. \tag{6}$$

Anschließend wird $\mathbf{f}$ neu berechnet mit der neuen Systemmatrix $S^T W S$. Durch Iterieren der letzten beiden Schritte nähert man sich sukzessive an die Lösung an. [8]

2.2.2 Data Normalization

Um den Algorithmus robust gegenüber Ähnlichkeitstransformationen (Translation, Rotation und Skalierung) zu machen, wird eine Normalisierung der Daten ausgeführt und der Algorithmus dann auf diese normalisierten Daten angewendet. Die Normalisierung beinhaltet eine Translation und Skalierung aller Punkte $\tilde{\mathbf{x}}_i$ und $\tilde{\mathbf{x}}_i{}'$, $1 \leq i \leq n$, mit Hilfe der affinen Transformationen T und T', so dass $T\tilde{\mathbf{x}}_i$ und $T'\tilde{\mathbf{x}}_i{}'$ im Durchschnitt die Koordinaten $(1, 1, 1)^T$ haben. Mit den normalisierten Punkten sieht die Formel der Epipolargeometrie (siehe auch Gleichung (1)) folgendermaßen aus:

$$\tilde{\mathbf{x}}'^T T'^T \hat{F} T \tilde{\mathbf{x}} = \hat{\mathbf{s}}^T \hat{\mathbf{f}} \tag{7}$$

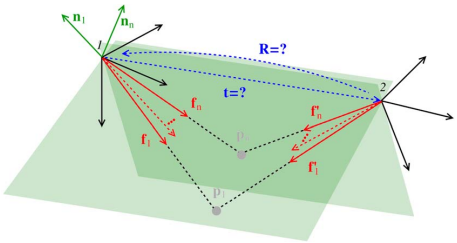

Abbildung 2: Der Eigensolver berechnet die relative Pose (blau) bestehend aus R und t. Gegeben sind die Richtungsvektoren (rot). Quelle: L. Kneip, S. Lynen [5]

Dabei ist $\hat{F}$ die Fundamentalmatrix der normalisierten Daten und $\hat{\mathbf{f}}$ die Fundamentalmatrix der normalisierten Daten in Vektorform. [8]

Schließlich kann aus $\hat{F}$ die Fundamentalmatrix F für die originalen Punkte berechnet werden: $F = T'^T \hat{F} T$. [8]

2.2.3 Rank Enforcement

Im Allgemeinen wird die Lösung des 8-Punkte-Algorithmus (d.h. die Fundamentalmatrix F) keine Singuläre Matrix sein. Deshalb wird ein Rank Enforcement gemacht [8]. Dazu wird F, z.B. mit Hilfe der Singulärwertzerlegung (singular value decomposition, SVD), durch die nächste Matrix vom Rang 2 ersetzt [7]. Die SVD zerlegt die Matrix F in die drei Matrizen $F = UDV^T$. Anschließend wird in der Diagonalmatrix D, welche die Singulärwerte enthält, der kleinste Singulärwert zu 0 gesetzt. Schließlich kann die neue Lösung durch Multiplikation der drei Matrizen berechnet werden. [3]

2.3 Direct Optimization of Frame-to-Frame Rotation

Ein anderer Ansatz zur Lösung des Problems der Schätzung der Eigenbewegung ist die *Direct Optimization of Frame-to-Frame Rotation* aus [5]. Da diese Optimierung auf einem Eigenwert-Minimierungsproblem basiert, wird dieser Ansatz im Folgenden auch als *Eigensolver* bezeichnet.

2.3.1 Funktionsweise

Der Algorithmus berechnet direkt (d.h. ohne den Umweg über die Fundamentalmatrix) die relative Pose zweier aufeinander folgender Kamerapositionen (siehe Abb. 2). Dazu wird für jede Punktkorrespondenz aus p und p' jeweils ein 3D-Richtungsvektor $\mathbf{f}_i$ bzw. $\mathbf{f}'_i$ benötigt, der von dem jeweiligen Kamerazentrum in Richtung des Bildpunktes zeigt. Diese Richtungsvektoren erhält man, indem man an die 2D-Koordinate eine dritte Koordinate mit dem Wert „1“ anhängt, d.h. aus dem Punkt $p = (x, y)$ wird der Richtungsvektor $\mathbf{f} = (x, y, 1)$. Aus jeweils korrespondierenden Richtungsvektoren in zwei aufeinander folgenden Frames berechnet der Eigensolver direkt die Rotationsmatrix R und den Translationsvektor t. Die Matrix R transformiert die Vektoren des zweiten Frames in den ersten Frame, der Vektor t gibt die Position des zweiten Kamerazentrums im Koordinatensystem der ersten Kamera an. [5]

Die Berechnung von R und t wird in [5] ausführlich erläutert. Die dort erwähnte Implementierung des Eigensolvers aus der OpenGV Bibliothek[1] wurde auch in dieser Arbeit verwendet.

[1] Online verfügbar unter: http://laurentkneip.github.io/opengv

2.3.2 Gewichte

Da auch dieser Algorithmus mit Ausreißern umgehen muss, werden wie schon beim robusten 8-Punkte Algorithmus (siehe Abschnitt 2.2) Gewichte benutzt. Diese werden wie folgt berechnet:

Für jede Punktkorrespondenz wird eine epipolare Ebene definiert. Diese wird aus den beiden Kamerazentren und dem 3D Punkt aufgespannt (siehe grüne Flächen in Abbildung 2). Für diese Ebene lässt sich der Normalenvektor berechnen:

$$\mathbf{n}_i = \mathbf{f}_i \times \mathbf{R}\mathbf{f}'_i. \tag{8}$$

Jeder der Normalenvektoren $\mathbf{n}_i$ sollte idealerweise senkrecht zum Translationsvektor t stehen (da t in jeder der epipolaren Ebenen liegt). D.h. das Skalarprodukt aus $\mathbf{n}_i$ und t sollte 0 ergeben:

$$\mathbf{n}_i \cdot t = 0. \tag{9}$$

Da die Punktkorrespondenzen jedoch nicht ideal sind, d.h. sie enthalten Ausreißer, wird das Skalarprodukt $\mathbf{n}_i \cdot t$ umso größer, je schlechter eine Punktkorrespondenz ist. D.h. der Betrag des Skalarproduktes ist ein Maß für den Fehler einer Punktkorrespondenz:

$$err_i = |\mathbf{n}_i \cdot t| \tag{10}$$

Nun können die Gewichte durch

$$w_i = \frac{1}{max(\epsilon, err)} \tag{11}$$

berechnet werden, wobei die Konstante $\epsilon > 0$ verhindert, dass durch 0 geteilt wird.

Nun werden R und t unter Zuhilfenahme der Gewichte mit Hilfe des Eigensolver Algortihmus erneut berechnet. Das Berechnen der Gewichte und das Berechnen von R und t werden solange iteriert, bis man sich nahe genug an die Lösung angenähert hat.

3 Implementierung

Das gesamte Projekt ist in C++ implementiert unter Zuhilfenahme der im folgenden Abschnitt beschriebenen Bibliotheken. Im Folgenden werden auch diverse Problematiken erläutert, die bei der Implementierung auftraten, die verwendeten Ein- und Ausgabeformate, sowie einige Routinen, die zur weiteren Vor- und Nachbearbeitung der Daten notwendig sind, damit die Ergebnisse der Algorithmen schließlich verglichen werden können.

3.1 Verwendete Bibliotheken

3.1.1 cmatrixlib

Die cmatrix Bibliothek enthält zahlreiche hilfreiche Funktionen zum Umgang mit Matrizen bzw. Bilddaten. Unerlässlich für diese Arbeit war die Funktion "readMiddlebury", die das Einlesen der optischen Flussfelder aus den zur Verfügung gestellten Dateien ermöglicht.

3.1.2 OpenGV

OpenGV steht für Open Geometric Vision. Die Bibliothek wird in [4] beschrieben und die Autoren von [5] stellen darin alle erwähnten Algorithmen zur Verfügung. Die Bibliothek beinhaltet u.a. Algorithmen zur Berechnungen von Position und Orientierung von kalibrierten Kameras. Eine Implementierung des Eigensolvers aus [5] ist in der OpenGV Bibliothek enthalten und wurde für diese Arbeit genutzt.

3.2 Berechnung der Punktkorrespondenzen aus dem optischen Fluss

Wie in Abschnitt 1.1 beschrieben, sind die optischen Flussfelder zwischen zwei aufeinander folgenden Frames gegeben. Diese Daten sind das Ergebnis des Dense Point Trackings [6] und der Schätzung des optischen Flusses [1] und werden mit der Funktion *readMiddlebury* aus der cmatrixlib (siehe Abschnitt 3.1.1) eingelesen. Um die Punktkorrespondenzen zu erhalten, werden die Flussvektoren zu den jeweiligen Punktkoordinaten addiert. Werden die Punktkorrespondenzen für weiter auseinander liegende Frames benötigt, so werden entsprechend die Flussvektoren über mehrere Frames hinweg addiert. Nach jedem Frame müssen die Punkte, deren Korrespondenzen außerhalb des Bildbereiches liegen, aussortiert werden, da für diese kein optischer Fluss bekannt ist.

3.3 Effiziente Berechnung der Systemmatrix S^TWS

Wie in Abschnitt 2.2.1 erläutert, muss beim robusten 8-Punkte Algorithmus in jeder Iteration die Systemmatrix S^TWS neu berechnet werden. Diese Matrixmultiplikation führt zu sehr vielen Rechenschritten und damit zu einer sehr langen Laufzeit: S^T ist eine $9 \times n$ Matrix, W eine $n \times n$ Matrix und S eine $n \times 9$ Matrix. D.h. zur Berechnung sind $9 \cdot n \cdot n + 9 \cdot 9 \cdot n$ Multiplikationen und $9 \cdot n \cdot (n-1) + 9 \cdot 9 \cdot (n-1)$ Additionen notwendig, wobei n die Anzahl der Punktkorrespondenzen ist. D.h. in diesem Schritt wächst die Laufzeit *quadratisch* zu n. Da W jedoch eine Diagonalmatrix ist, enthält sie nur n Werte, die von 0 verschieden sein können, d.h. zur Berechnung von S^TW sind nur n Multiplikationen und keine Additionen notwendig; zur Multiplikation dieser Matrix mit S kommen noch einmal $9 \cdot 9 \cdot n$ Multiplikationen und $9 \cdot 9 \cdot (n-1)$ Additionen hinzu, was insgesamt zu einer *linearen* Laufzeit von n führt.

3.4 Höhere Genauigkeit beim 8-Punkte Algorithmus

Bereits bei den ersten Tests des Programmcodes wurden Probleme bei der Berechnung der Gewichte festgestellt: Die Gewichte der einzelnen Punktkorrespondenzen konvergierten nicht, d.h. bei „schlechten“ Korrespondenzen wurden die Gewichte nicht immer kleiner und bei „guten“ nicht immer größer, sondern es wurde sprunghaftes Verhalten festgestellt. Das Problem lag schließlich in der Verwendung des Datentyps *float* an einigen Stellen. Dessen Genauigkeit ist zu klein. Stattdessen wird jetzt der Datentyp *double* verwendet.

3.5 Berechnung der Rotation und Translation

Mit dem 8-Punkte Algorithmus wird lediglich die Fundamentalmatrix berechnet, von Interesse ist jedoch die Eigenbewegung der Kamera, d.h. aus der Fundamentalmatrix F müssen die Rotationsmatrix R und der Translationsvektor $\mathbf{t}$ berechnet werden.

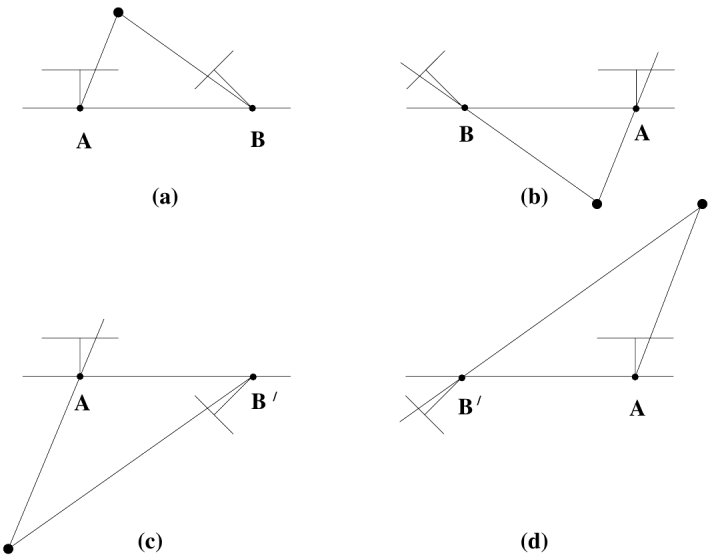

Abbildung 3: Die vier möglichen Lösungen bei der Rekonstruktion der Kameratranslation und -rotation. Nur bei Lösung (a) befindet sich der Punkt vor beiden Kameras. Quelle: R. Hartley, A. Zisserman [3]

Dazu wird als erstes die *Essentialmatrix* E benötigt. Als Essentialmatrix wird eine Fundamentalmatrix dann bezeichnet, wenn diese zu einem Paar von normalisierten Kameras gehört, d.h. Kameras, die jeweils die Einheitsmatrix I als Kalibrationsmatrix haben. Demzufolge erhält man die Essentialmatrix E mit Hilfe der Fundamentalmatrix F und den Kalibrationsmatrizen K und K' folgendermaßen [3]:

$$E = K'^T F K. \tag{12}$$

Anschließend wird E mit Hilfe der SVD in die drei Matrizen UDV^T zerlegt. Nun erhält man für R zwei mögliche Lösungen [3]:

$$R = UWV^T \text{ oder } R = UW^TV^T, \tag{13}$$

wobei W die Orthogonalmatrix

$$W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

ist. [3]

Die Translation $\mathbf{t}$ steht in der letzten Spalte von U. Da jedoch das Vorzeichen von E und damit auch von $\mathbf{t}$ unbekannt ist, gibt es auch für $\mathbf{t}$ zwei mögliche Lösungen [3]:

$$\mathbf{t} = +u_3 \text{ oder } \mathbf{t} = -u_3. \tag{15}$$

Für eine gegebene Kameramatrix der ersten Kamera $P = [I|0]$ existieren für die Kameramatrix der zweiten Kamera P' die vier Lösungen [3]

$$P' = [UWV^T| + u_3] \text{ oder } [UWV^T| - u_3] \text{ oder } [UW^TV^T| + u_3] \text{ oder } [UW^TV^T| - u_3]. \tag{16}$$

Abbildung 4: Links ein Bild der *town*-Szene, rechts ein Bild der *Straßenschildszene*.

Welche der vier Lösungen die richtige ist, kann herausgefunden werden, indem ein Punkt rekonstruiert wird (auch Triangulation genannt) und geschaut wird, bei welcher der vier Lösungen sich der Punkt vor beiden Kameras befindet (siehe Abbildung 3). [3]

Dazu wurde immer der Punkt mit der größten Gewichtung herangezogen, da dieser mit großer Wahrscheinlichkeit kein Ausreißer ist.

Beim Eigensolver Algorithmus wurde ebenfalls eine Triangulation gemacht, da die Richtung des Translationsvektors insbesondere bei sehr kleinen Rotationen nicht immer korrekt berechnet wurde.

3.6 Abbruchkriterien

Als Abbruchkriterium wird beim robusten 8-Punkte Algorithmus bei jeder Iteration die Energie $E(\mathbf{f})$ (siehe Gleichung 3) berechnet. Diese sollte bei jeder Iteration kleiner werden und schließlich konvergieren. Implementiert wurde das Abbruchkriterium so, dass nach Iteration k abgebrochen wird, wenn gilt:

$$E_{k-1}(\mathbf{f}) < (1 + 10^{-8})E_k(\mathbf{f}). \tag{17}$$

Analog dazu wird beim Eigensolver Algorithmus in jeder Iteration die Summe aller Fehler err_{ges} berechnet (siehe dazu Gleichung 10):

$$err_{ges} = \sum_{i=1}^{n} err_i = \sum_{i=1}^{n} |\mathbf{n}_i \cdot t|. \tag{18}$$

Entsprechend wird nach Iteration k abgebrochen wenn

$$err_{ges,k-1}(\mathbf{f}) < (1 + 10^{-8})err_{ges,k}(\mathbf{f}). \tag{19}$$

4 Experimente

4.1 Benutzte Szenen

Zum Testen der implementierten Algorithmen wurden verschiedene Szenen verwendet. Bei den Szenen *town* und *town2* wurden Bilder von einem am Rechner erzeugten Objekt (siehe Abbildung 4) aus verschiedenen Kamerapositionen erstellt: Bei der *town*-Szene dreht sich die Kamera in ein-Grad-Schritten einmal um das Objekt. Bei

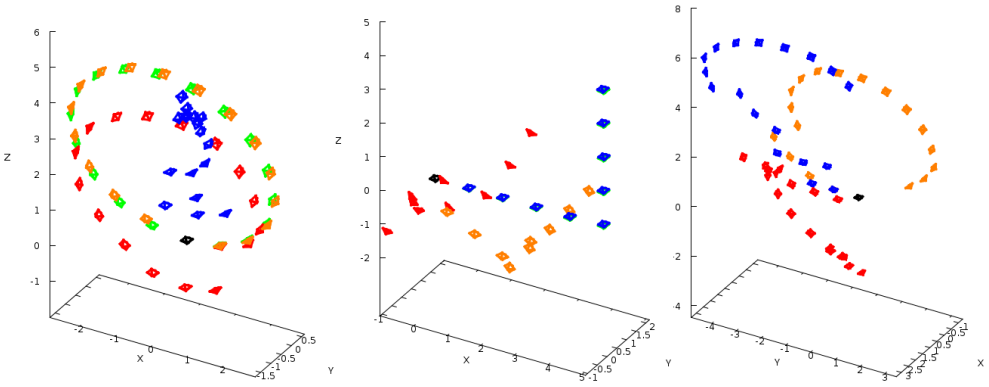

Abbildung 5: Mit den Standardalgorithmen berechnete Kamerapositionen der drei Szenen *town* (links), *town2* (mitte) und *Straßenschild* (rechts). *Schwarz*: Position der Kamera beim 1. Frame. *Grün*: Exakte Kamerapositionen. *Rot*: Positionen berechnet vom 8-Punkte Algorithmus. *Blau*: Berechnete Positionen des Eigensolver Algorithmus mit Einheitsmatrix als Initialisierung für *R*. *Orange*: Berechnete Positionen des Eigensolver Algorithmus, wobei mit den Rotationsmatrizen des 8-Punkte Algorithmus initialisiert wurde.

der *town2*-Szene bewegt sich die Kamera zunächst in 100 gleichgroßen Schritten nach rechts und anschließend in 99 ebenfalls gleichgroßen Schritten nach vorne, wobei die Blickrichtung der Kamera immer exakt gleich bleibt.

Als dritte Szene wurde eine reale Szene verwendet. In dieser läuft jemand mit der Kamera einmal um ein Straßenschild (siehe Abbildung 4). Da die Kamera von Hand geführt wurde, sind die exakten Bewegungen der Kamera nicht bekannt, so dass es nicht möglich ist, für diese Szene die Abweichungen der Algorithmen zur wahren Kamerabewegung zu bestimmen.

4.2 Visualisierung

Um die Ergebnisse visuell darstellen zu können werden die errechneten Kamerapositionen mit Hilfe von Gnuplot in ein dreidimensionales Koordinatensystem gezeichnet (Abbildungen 5 bis 7).

4.3 Frameabstände

Um das Verhalten bei unterschiedlich großen Kamerabewegungen zu untersuchen, wurden die Algorithmen mit verschiedenen Frameabständen getestet. Für den Frameabstand n wurden zuerst Flussfelder für jeweils n aufeinanderfolgende Frames berechnet, indem jeweils die einzelnen Vektoren der einzelnen Flussfelder addiert wurden. Anschließend wurden die Algorithmen mit den neu berechneten Flussfeldern getestet.

4.4 Standardversion

Als erstes Experiment wurden die Standardversionen des 8-Punkte Algorithmus und des Eigensolvers ausgeführt, d.h. es wurden die Algorithmen verwendet, wie sie in Abschnitt 2.1 und Abschnitt 2.3.1 erläutert wurden, also ohne Gewichtungen der Punktkorrespondenzen und mit nur einer Iteration. Die berechneten Kamerapositionen für

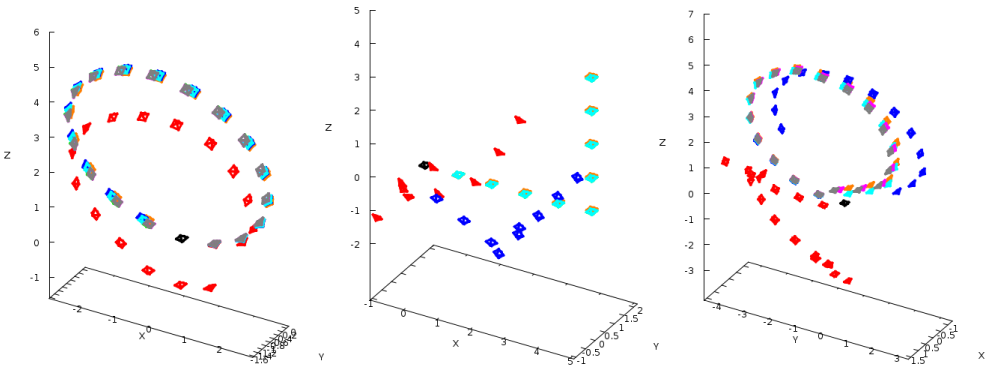

Abbildung 6: Mit den robusten Algorithmen berechnete Kamerapositionen der drei Szenen *town* (links), *town2* (mitte) und *Straßenschild* (rechts). *Schwarz*: Position der Kamera beim 1. Frame. *Grün*: Exakte Kamerapositionen. *Rot*: Berechnete Positionen des Standard 8-Punkte Algorithmus. *Blau*: Berechnete Positionen des Standard Eigensolver Algorithmus, Initialisierung mit Standard 8-Punkte Algorithmus. *Orange*: Berechnete Positionen des robusten 8-Punkte Algorithmus. *Cyan*: Berechnete Positionen des robusten Eigensolver Algorithmus, Initialisierung mit robustem 8-Punkte Algorithmus. *Magenta*: Berechnete Positionen des robusten 8-Punkte Algorithmus mit Tracks. *Grau*: Berechnete Positionen des robusten Eigensolver Algorithmus mit Tracks, Initialisierung mit robustem 8-Punkte Algorithmus.

einen Frameabstand von 20 sind in Abbildung 5 zu sehen. Zum Vergleich wurden auch die exakten Kamerapositionen in grün geplottet. Zur Initialisierung des Eigensolveralgorithmus (die verwendeten Eigensolverfunktionen aus der OpenGV-Bibliothek benötigen immer eine Initialisierung für die Rotation) wurde zuerst die Einheitsmatrix für R verwendet (im Plot blau dargestellt), in einem weiteren Durchlauf wurde mit den von dem 8-Punkte Algorithmus berechneten Rotationsmatrizen initialisiert (orange). Es ist zu sehen, dass das Ergebnis des Eigensolvers sehr stark von der Initialisierung der Rotationsmatrix abhängt. Dieses Verhalten wurde bei allen Szenen und mit allen getesteten Frameabständen beobachtet.

Es ist ebenfalls erkennbar, dass bei der *town2*-Szene die berechneten Positionen des Eigensolvers mit der Einheitsmatrix als Initialisierung (Abbildung 5, mittlerer Plot, Blau) sehr nahe an die tatsächlichen Kamerapositionen herankommen (grün, verdeckt durch Blau). Der Grund dafür liegt darin, dass die Einheitsmatrix, mit der initialisiert wurde, bereits der tatsächlichen Rotation entspricht. Damit musste der Algorithmus selbst nichts mehr leisten; die Rotation ist bereits nach der Initialisierung korrekt.

4.5 Robuste Version

Als nächstes wurden die robusten Versionen der Algorithmen verglichen, d.h. es wurden Gewichtungswerte für die einzelnen Punktkorrespondenzen berechnet und solange iteriert, bis das Abbruchkriterium (siehe Abschnitt 3.6) erfüllt war.

Für die *town*-Szene und die *Straßenschildszene* wurden diese Experimente zweimal ausgeführt: Einmal wurden die Punktkorrespondenzen mit Hilfe der kompletten Flussfelder (siehe Abschnitt 1.1) berechnet, und einmal wurden nur Punkte mit guter Struktur (Tracks) verwendet. Diese standen wie die Flussfelder ebenfalls zur Verfügung. Die Punkte mit guter Struktur beinhalten nur sogenannte Interest Points,

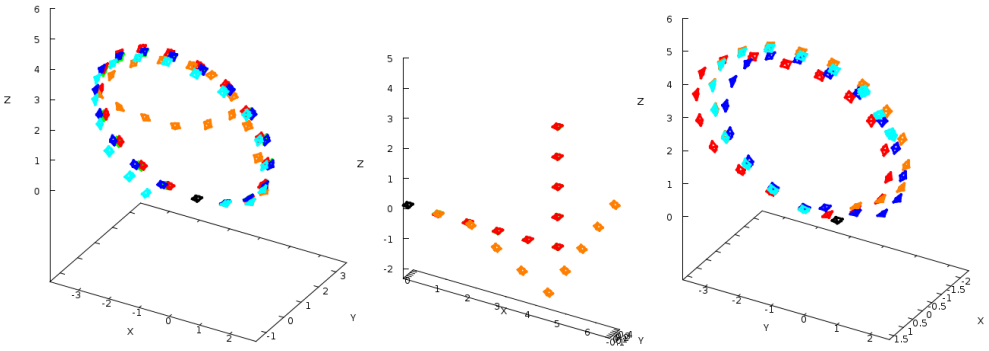

Abbildung 7: Mit RANSAC und den Standardalgorithmen berechnete Kamerapositionen der drei Szenen *town* (links), *town2* (mitte) und *Straßenschild* (rechts). *Schwarz*: Position der Kamera beim 1. Frame. *Grün*: Exakte Kamerapositionen. *Rot*: Berechnete Positionen des 8-Punkte Algorithmus, alle Punkte. *Blau*: Berechnete Positionen des 8-Punkte Algorithmus, Tracks. *Orange*: Berechnete Positionen des Eigensolver Algorithmus, Initialisierung mit robustem 8-Punkte Algorithmus, alle Punkte. *Cyan*: Berechnete Positionen des Eigensolver Algorithmus, Initialisierung mit robustem 8-Punkte Algorithmus, Tracks.

so dass man weniger Punktkorrespondenzen hat, die dafür jedoch besser sind, d.h. die Flussvektoren sind weniger fehlerbehaftet und enthalten weniger Ausreißer.

Abbildung 6 zeigt Kamerapositionen der robusten Algorithmen mit dem Frameabstand 20. Zum Vergleich sind nochmal die Kamerapositionen der Standardalgorithmen geplottet (8-Punkte: rot; Eigensolver: blau). Es ist zu sehen, dass die robusten Versionen der Algorithmen (andere Farben außer blau und rot) im Gegensatz zu den Standardversionen sehr nahe an die wahren Kamerapositionen herankommen. Das gilt sowohl bei der Verwendung der kompletten Flussfelder als auch bei den Punkten mit guter Struktur.

4.6 Standardversion mit RANSAC

Eine Alternative zu den robusten Algorithmen besteht in der Kombination von RANSAC (Random Sample Consensus) mit den Standardalgorithmen. Zuerst wird durch die Verwendung von RANSAC ein sogenanntes *Consensus Set* berechnet. Dieses besteht aus einer Teilmenge der Punktkorrespondenzen, die im Optimalfall keine Ausreißer mehr enthält. Anschließend wird auf dieser (ausreißerfreien) Teilmenge die Standardversion des Eigensolvers bzw. 8-Punkte Algorithmus angewendet.

Für RANSAC wurden die in der OpenGV Bibliothek vorhandenen Funktionen genutzt. Als Parameter wurden jeweils die Werte aus den mit OpenGV mitgelieferten Beispielen verwendet. Diese sehen wie folgt aus: Für den 8-Punkte Algorithmus: *threshold*: 2.0*(1.0-cos(atan(sqrt(2.0)*0.5/800.0))); *max_iterations*: 50. Für den Eigensolver Algorithmus: *threshold*: 1.0; *max_iterations*: 100; *SampleSize*: 10

In Abbildung 7 sind die Kamerapositionen der Standardversionen mit vorgeschaltetem RANSAC mit einem Frameabstand von 20 zu sehen. Alle bis auf den auf allen Punkten der *town*- und *town2*-Szene berechneten Eigensolver Algorithmus kommen recht nahe an die exakten Kamerapositionen heran.

5 Berechnung der Performance der Algorithmen

Um die Algorithmen vergleichen zu können, wurden für die Szenen *town* und *town2* jeweils die Abweichungen der Algorithmen zu der realen Rotation und Translation berechnet.

Für die Bestimmung der Abweichung der Rotation wurde der Winkel der Rotationsmatrix, die die reale Rotation in die vom Algorithmus berechnete Rotation transformiert, berechnet. Damit liegt die Abweichung der Rotation immer zwischen 0 und π.

Die Abweichung der Translation wurde bestimmt, indem der Betrag des Vektors berechnet wurde, der sich ergibt, wenn man den Translationsvektor der realen Translation von dem vom Algorithmus berechneten Translationsvektor abzieht. Daraus folgt, dass die so berechnete Abweichung der Translation immer einen Wert zwischen 0 und 2 ergibt, da alle Translationsvektoren zuvor auf eine Länge von 1 normiert wurden. Diese Normierung wurde durchgeführt, da in den Szenen keine einzige Längenangabe bekannt ist, so dass mit dem Translationsvektor in diesem Falle lediglich eine Richtung, jedoch keine Länge angegeben wird.

6 Ergebnisse

Die durchschnittlichen Abweichungen und Laufzeiten der Algorithmen bei verschiedenen Frameabständen sind in Tabelle 1 für die *town*-Szene und in Tabelle 2 für die *town2*-Szene angegeben.

Unter den 8-Punkte Algorithmen hat die robuste Version die beste Performance; nur mit den Tracks berechnet eine noch bessere als mit allen Punkten berechnet. Die benötigte Zeit zur Berechnung ist bei der robusten Version mit allen Punkten sehr groß.

Bei allen Algorithmen sind die Zeiten sehr viel kürzer, wenn nur die Tracks zur Berechnung verwendet werden anstatt aller Punktkorrespondenzen. Dies liegt an der sehr viel geringeren Anzahl der Tracks gegenüber allen Punkten: Für die *town*-Szene gibt es insgesamt 86.411 Trajektorien; die Anzahl der Trajektorien, die über zwei bestimmte Frames gehen, ist geringer. Im Gegensatz dazu bestehen die Bilder aus 409.920 ($854 \cdot 480$) Bildpunkten, abzüglich einiger Punkte in den Randbereichen, die keinen korrespondierenden Punkt im 2. Frame besitzen. Tatsächlich sind es zwischen 402.000 und 408.000 Punktkorrespondenzen pro Frameübergang.

Bei den Eigensolver Algorithmen hängt die Performance stark von der Initialisierung der Rotationsmatrix ab, je nach dem wie stark diese von der wahren Rotation abweicht. Es wurde versucht, mit der Einheitsmatrix zu initialisieren. Jedoch waren die Ergebnisse so schlecht, dass diese Möglichkeit nicht weiter betrachtet wurde (siehe Abbildung 5,linker Plot, blau; bei der *town2*-Szene (mittlerer Plot) ist die Performance in diesem Fall nur gut, weil die Einheitsmatrix in diesem Fall gerade der wahren Rotation entspricht).

Für die Initialisierung der Eigensolver Algorithmen wurden schließlich der robuste 8-Punkte Algorithmus und zum Vergleich der Standard 8-Punkte Algorithmus gewählt. Der Robuste, weil er eine sehr gute Performance hat und damit der wahren Rotation schon sehr nahe kommt, und der Standardalgorithmus, weil dieser schnell und einfach zu berechnen ist.

Die Abweichungen der Standardalgorithmen sind groß. Bei der *town*-Szene hat der Standard 8-Punkte Algorithmus die größten Abweichungen von allen 8-Punkte Algorithmen, bei der *town2*-Szene sogar die größten Abweichungen von allen Algorithmen. Beim Standard Eigensolver sind die Abweichungen geringer als beim Stan-

Tabelle 1: Performances der Algorithmen bei der *town*-Szene im Vergleich. Die Szene, bestehend aus insgesamt 357 Frames, wurde für jeden Frameabstand n komplett berechnet, so als bestünde die Szene nur aus jedem n-ten Frame. Damit ergeben sich 357/n Frameübergänge (es muss abgerundet werden). Für jeden Frameübergang wurden t und R berechnet; anschließend deren Abweichungen zu dem wahren t bzw. R. Δt_n und ΔR_n geben den Mittelwert dieser Abweichungen von t bzw. R für einen Frameabstand von n an. *Zeit* gibt die durchschnittliche Laufzeit des Algorithmus in Sekunden für die Berechnung eines Frameüberganges an (Laufzeiten gemessen unter Ubuntu 14.04 Kernel 3.13.0 x86_64, Intel Core i5-4200M CPU @ 2.50GHz). Der Durchschnitt wurde über die Einzelzeiten bei einem Frameabstand von 20 gebildet. Die besten Werte sind jeweils Fett dargestellt.

Algorithmus	Δt_{20} ΔR_{20}	Δt_4 ΔR_4	Δt_3 ΔR_3	Δt_2 ΔR_2	Δt_1 ΔR_1	*Zeit*
8-Punkte Standard	0,0885 0,0590	0,1402 0,0290	0,1502 0,0248	0,1440 0,0163	0,1681 0,0075	0,9996
8-Punkte Standard, Tracks	0,0263 0,0121	0,0239 0,0041	0,0237 0,0031	0,0294 0,0021	0,0475 0,0012	0,0037
8-Punkte Robust	0,0178 0,0035	0,0140 0,0006	0,0130 0,0005	0,0113 0,0004	0,0106 0,0003	8,4535
8-Punkte Robust, Tracks	0,0021 0,0012	0,0030 **0,0005**	0,0034 0,0006	0,0040 0,0004	0,0063 0,0004	0,0342
8-Punkte, RANSAC	0,0310 0,0167	0,0245 0,0031	0,0305 0,0027	0,0379 0,0022	0,0644 0,0017	1,7265
8-Punkte, RANSAC, Tracks	0,0198 0,0104	0,0273 0,0035	0,0317 0,0031	0,0411 0,0026	0,0762 0,0023	0,0283
Eigensolver Standard, Init.: 8-Punkte Standard	0,0094 0,0069	0,0894 0,0111	0,0698 0,0083	0,0907 0,0060	0,1279 0,0038	0,0378
Eigensolver Standard, Init.: 8-Punkte Robust	0,0094 0,0069	0,0070 0,0014	0,0063 0,0011	0,0055 0,0009	0,0070 0,0006	0,0380
Eigensolver Standard, Tracks, Init.: 8-Punkte Standard	0,0010 0,0013	0,0537 0,0074	0,1012 0,0098	0,1011 0,0066	0,1227 0,0036	**0,0006**
Eigensolver Standard, Tracks, Init.: 8-Punkte Robust	0,0010 0,0013	0,0010 **0,0005**	0,0011 0,0005	0,0012 0,0004	0,0016 0,0004	**0,0006**
Eigensolver Robust, Init.: 8-Punkte Standard	0,0084 0,0048	0,0531 0,0058	0,0691 0,0057	0,0852 0,0042	0,1143 0,0028	0,0993
Eigensolver Robust, Init.: 8-Punkte Robust	0,0084 0,0048	0,0066 0,0006	0,0058 **0,0004**	0,0043 **0,0002**	0,0014 **0,0001**	0,0980
Eigensolver Robust, Tracks, Init.: 8-Punkte Standard	**0,0007** **0,0011**	0,0468 0,0054	0,0905 0,0072	0,0828 0,0046	0,1139 0,0030	0,0017
Eigensolver Robust, Tracks, Init.: 8-Punkte Robust	**0,0007** **0,0011**	**0,0006** 0,0006	**0,0006** 0,0006	**0,0008** 0,0004	**0,0012** 0,0004	0,0016
Eigensolver, RANSAC, Init.: 8-Punkte Standard	0,1928 0,1757	0,1627 0,0720	0,1654 0,0732	0,1525 0,0481	0,1726 0,0254	0,1167
Eigensolver, RANSAC, Init.: 8-Punkte Robust	0,0591 0,0695	0,0709 0,0777	0,0550 0,0610	0,0447 0,0509	0,0243 0,0234	0,1144
Eigensolver, RANSAC, Tracks, Init.: 8-Punkte Standard	0,1163 0,0643	0,1645 0,0901	0,1614 0,0666	0,1540 0,0459	0,1723 0,0248	0,0018
Eigensolver, RANSAC, Tracks, Init.: 8-Punkte Robust	0,0379 0,0321	0,0693 0,0782	0,0591 0,0657	0,0450 0,0510	0,0245 0,0246	0,0019

Tabelle 2: Performances der Algorithmen bei der *town2*-Szene im Vergleich. Berechnung der Werte wie in Tabelle 1.

Algorithmus	Δt_{20} ΔR_{20}	Δt_4 ΔR_4	Δt_3 ΔR_3	Δt_2 ΔR_2	Δt_1 ΔR_1	*Zeit*
8-Punkte Standard	1,0418	0,7539	0,7447	0,6825	0,4645	0,6620
	0,2261	0,0314	0,0171	0,0078	0,0015	
8-Punkte Robust	0,0071	0,0041	0,0344	0,0044	0,0183	5,2035
	0,0010	**0,0003**	**0,0002**	**0,0002**	0,0002	
8-Punkte, RANSAC	0,0096	0,0183	0,0293	0,0395	0,0525	1,0424
	0,0047	0,0010	0,0010	0,0012	0,0008	
Eigensolver Standard,	0,3894	0,5196	0,5429	0,3879	0,2331	0,0456
Init.: 8-Punkte Standard	0,0888	0,0195	0,0136	0,0048	0,0007	
Eigensolver Standard,	0,0061	0,2533	0,2509	0,2330	0,2065	**0,0453**
Init.: 8-Punkte Robust	0,0021	0,0006	0,0004	0,0004	0,0002	
Eigensolver Robust,	0,3821	0,4233	0,3760	0,1873	0,0390	0,1694
Init.: 8-Punkte Standard	0,0830	0,0168	0,0121	0,0039	0,0006	
Eigensolver Robust,	**0,0032**	**0,0032**	**0,0029**	**0,0030**	**0,0067**	0,1699
Init.: 8-Punkte Robust	0,0015	**0,0003**	**0,0002**	**0,0002**	**0,0001**	
Eigensolver, RANSAC,	1,0297	0,7625	0,7477	0,6887	0,4666	0,2835
Init.: 8-Punkte Standard	0,0898	0,0504	0,0329	0,0250	0,0122	
Eigensolver, RANSAC,	0,0713	0,0297	0,0577	0,0168	0,0240	0,1559
Init.: 8-Punkte Robust	0,0661	0,0394	0,0350	0,0209	0,0122	

dard 8-Punkte Algorithmus. Da R und t des 8-Punkte Algorithmus als Initialisierung für den Eigensolver verwendet werden, zeigt dies, dass der Eigensolver das Ergebnis verbessert.

Beim Vergleich der robusten Algorithmen fällt auf, dass die Performance des Eigensolvers wieder stark von der Initialisierung abhängt: Wird mit dem schlechteren R und t des Standard 8-Punkte Algorithmus initialisiert, wird das Ergebnis nicht annähernd so gut wie wenn R und t des robusten 8-Punkte Algorithmus zur Initialisierung verwendet werden. Und das, obwohl der Eigensolver solange iteriert wird, bis das Ergebnis konvergiert. Damit ist das Ergebnis des Standard 8-Punkte Algorithmus nicht so gut zur Initialisierung für den Eigensolver geeignet.

Wird der robuste Eigensolver mit R und t des robusten 8-Punkte Algorithmus initialisiert, so wird dessen Ergebnis durch den Eigensolver noch verbessert. In vielen Fällen wurden durch diese Vorgehensweise die besten Performances überhaupt erreicht. Bessere Performances konnten nur in einigen Fällen durch die Benutzung der Tracks erreicht werden.

Werden die robusten Algorithmen mit Tracks verwendet, so werden fast immer bessere Performances bei den Translationen erreicht, als wenn alle Punktkorrespondenzen zur Berechnung verwendet werden. Die Performances der Rotationen sind meist sehr ähnlich. Beim Vergleich der Standardalgorithmen mit Tracks ist der 8-Punkte Algorithmus meist genauer als der Eigensolver, bei den robusten Algorithmen mit Tracks ist der Eigensolver genauer in der Berechnung der Translation, bei der Berechnung der Rotation sind beide fast gleich gut.

Der RANSAC 8-Punkte Algorithmus erreicht eine bessere Performance als der Standard 8-Punkte Algorithmus, aber eine etwas schlechtere als der robuste 8-Punkte Algorithmus. Gleichzeitig ist seine Laufzeit etwas länger als die des Standard 8-Punkte Algorithmus. Der RANSAC Eigensolver Algorithmus erzielt fast immer eine schlech-

tere Performance als der robuste Eigensolver und der Standard Eigensolver Algorithmus. Außerdem sind bereits t und R, die zur Initialisierung verwendet werden, besser als das Ergebnis des RANSAC Eigensolvers. Das gilt auch in Verbindung mit den Tracks.

Da die Algorithmen in den Standardversionen keine Gewichtungen der Punktkorrespondenzen vornehmen und jeweils nur eine Iteration berechnet wird, sind die Ergebnisse entsprechend nicht sehr gut. Im Falle der *town*-Szene läuft der 8-Punkte Algorithmus in die richtige Richtung, d.h. die Genauigkeit der Rotationen ist zwar gering, sie reicht jedoch schon als Initialisierung für den Eigensolver aus, so dass bereits der Standard-Eigensolver ein gutes Ergebnis erzielt (siehe auch Abbildung 5, linker Plot).

7 Fazit

Dass die robusten Algorithmen bessere Performances erreichen als die Standardalgorithmen, war zu erwarten. Der Nachteil des robusten 8-Punkte Algorithmus ist seine lange Laufzeit, während der Nachteil des robusten Eigensolvers in der benötigten Initialisierung liegt.

Die Initialisierung des Eigensolvers mit dem Standard 8-Punkte Algorithmus könnte in einigen Fällen ausreichend sein, in diesem Fall hätte man mit der Kombination des Standard 8-Punkte Algorithmus mit anschließendem (robusten) Eigensolver ein gutes Ergebnis in einer relativ kurzen Rechenzeit. Liefert dieses Vorgehen kein zufriedenstellendes Ergebnis, so sollte man die lange Laufzeit des robusten 8-Punkte Algorithmus in kauf nehmen, außer, man kann auf anderem Wege eine gute Initialisierung erhalten. Mit einer guten Initialisierung erhält man mit dem robusten Eigensolver in der Regel ein besseres Ergebnis als mit dem 8-Punkte Algorithmus.

Liegen zusätzlich die Punktkorrespondenzen der Punkte mit guter Struktur vor, so bringt deren Einsatz eine Verbesserung der Performance, vor allem was die Translation anbelangt; es konnte nur eine recht geringe Verbesserung der Genauigkeit der berechneten Rotation beobachtet werden.

Die Verwendung von RANSAC mit dem 8-Punkte Algorithmus stellt einen Kompromiss dar zwischen Laufzeit und Genauigkeit: Die Ergebnisse sind schon recht gut, die Laufzeit ist aber noch viel geringer als beim robusten 8-Punkte Algorithmus. Der Einsatz von RANSAC mit dem Eigensolver konnte nicht überzeugen, da dies zu einer Verschlechterung der Performance gegenüber dem 8-Punkte Algorithmus führte, obwohl mit dessen Ergebnissen Initialisiert wurde.

Die genauesten Ergebnisse erhält man mit dem robusten Eigensolver und der Verwendung von Tracks, eine gute Initialisierung vorausgesetzt.

Literatur

[1] T. Brox and J. Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):500–513, 2011.

[2] O. D. Faugeras, Q.-T. Luong, and T. Papadopoulo. *The geometry of multiple images - the laws that govern the formation of multiple images of a scene and some of their applications.* MIT Press, 2001.

[3] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, New York, NY, USA, 2nd edition, 2003.

[4] L. Kneip and P. Furgale. OpenGV: A unified and generalized approach to real-time calibrated geometric vision. In *Proc. of The IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China, May 2014.

[5] L. Kneip and S. Lynen. Direct Optimization of Frame-to-Frame Rotation. In *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, pages 2352–2359, 2013.

[6] N. Sundaram, T. Brox, and K. Keutzer. Dense point trajectories by GPU-accelerated large displacement optical flow. In *European Conference on Computer Vision (ECCV)*, Lecture Notes in Computer Science. Springer, Sept. 2010.

[7] R. Y. Tsai and T. S. Huang. Uniqueness and Estimation of Three-Dimensional Motion Parameters of Rigid Objects with Curved Surfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):13–26, 1984.

[8] L. Valgaerts, A. Bruhn, M. Mainberger, and J. Weickert. Dense versus Sparse Approaches for Estimating the Fundamental Matrix. *International Journal of Computer Vision*, 96(2):212–234, 2012.