

Computer Vision Challenge

Computer Vision

im Studiengang Master Elektro- und Informationstechnik

Sommersemester 2018

Gruppe 11:

Tim Janßen

Andre Thommessen

Oliver Doege

Julia Ströbel

Sebastian Hügler

Abgabe: 12.09.2018

Prüfer: [Prof. Dr.-Ing. Klaus Diepold](http://www.ldv.ei.tum.de/team/ordinarius/klaus-diepold/) / M.Sc. Stefan Röhrl

Inhaltsverzeichnis

[Einleitung 3](#_Toc524174775)

[Aufgabenstellung 3](#_Toc524174776)

[Idee und Umsetzung 5](#_Toc524174777)

[Umwandlung der Bilder in Graubilder 6](#_Toc524174778)

[Intensitäts- und Beleuchtungskorrektur 6](#_Toc524174779)

[Bilaterale Filterung 6](#_Toc524174780)

[Harris-Merkmale berechnen 7](#_Toc524174781)

[Korrespondenzschätzung 8](#_Toc524174782)

[Berechne die Essentielle Matrix 8](#_Toc524174783)

[Bildrektifizierungsalgorithmus 8](#_Toc524174784)

[Disparitätsermittlung 11](#_Toc524174785)

[Projection 11](#_Toc524174786)

[Ausgabe des Free-Viewpoint Bildes 11](#_Toc524174787)

[Graphische Benutzeroberfläche (GUI) 12](#_Toc524174788)

[Abbildungsverzeichnis 14](#_Toc524174789)

[Literaturverzeichnis 15](#_Toc524174790)

# Einleitung

Die Computer Vision Challenge ist ein Bestandteil zur Bewertung der Leistungen der Studenten über ein Semester im Fach Computer Vision. Dieses Jahr besteht die Challenge darin, aus einem Stereo-Bild-Paar eine dritte virtuelle Ansicht zu generieren. Die Computer Vision Challenge ist eine Programmierarbeit, die in Gruppen bestehend aus drei bis fünf Personen durchgeführt wird. Es wird vom Lehrstuhl ein Skript vorgegeben, dessen Vorgaben zu erweitern sind. Einzelne Schritte sind zu kommentieren und Quellen auszuweisen.

Zusatzpunkte sind über eine optimierte Laufzeit und über eine GUI zu erreichen. Ein optimierter Code mit einer relativ kurzen Ausführungszeit wird positiv angerechnet. Eine graphische Benutzeroberfläche(GUI) wirkt sich ebenfalls positiv aus. Diese sollte zum Laden von Stereobildern, zur Einstellung des Blickwinkels der virtuellen Ansicht und zur Ausgabe der virtuellen Ansicht genutzt werden können.

# Aufgabenstellung

Die Challenge besteht darin, aus einem Stereo-Bild-Paar eine dritte virtuelle Ansicht zu generieren. Dabei soll der Blickwinkel der virtuellen Ansicht zwischen den beiden realen Ansichten liegen und durch einen Prozentwert frei bestimmbar sein. Das Programm soll in Matlab ohne die Hilfe spezialisierter Toolboxen erstellt werden. Als Grundlage liegen diverse Papers vor, die übers Semester verteilt wurden, dazu die Folien aus den Vorlesungen sowie die eigene Internetrecherche.

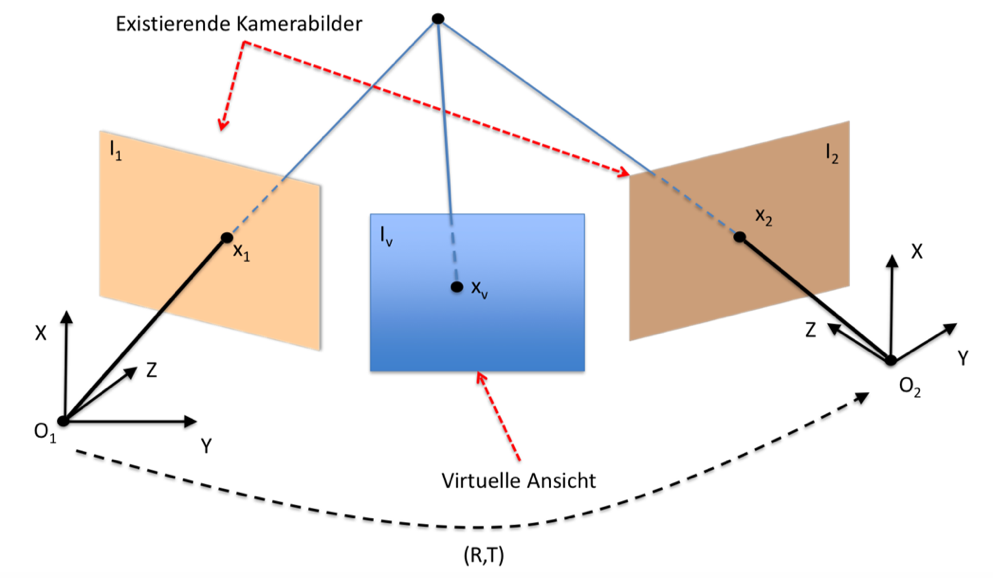


Abbildung 1: Geometrischer Zusammenhang zwischen den realen Bildern und einer künstlich generierten Ansicht

Abbildung 1 zeigt das zu untersuchende Szenario. Es werden zwei Bilder vom selben Motiv vorgegeben und mittels des zu erstellenden Programm übereinandergelegt, sodass ein drittes Bild berechnet wird.

Das vorgegebene Skript *challange.m* soll so erweitert werden, dass die Farbbilder *img/L1.jp*g und *img/R1.jpg* geladen und der Funktion *free\_viewpoint* übergeben werden. Die Funktion *free\_viewpoint* soll dann anhand des Parameters *p* eine virtuelle Ansicht zwischen den beiden realen Ansichten generieren.

Der Parameter *p* soll standardmäßig auf 50% stehen. Die Rechenzeit, die die Funktion *free\_viewpoint* benötigt hat, soll in der Variable *elapsed\_time* abgespeichert werden.



Abbildung 2: Linke Ansicht (L1.jpg)



Abbildung 3: Rechte Ansicht (R1.jpg)

# Idee und Umsetzung

In diesem Kapitel werden zunächst die Idee und der Ablauf des Programms skizziert. Danach werden die Aufgaben der einzelnen Programmbestandteile erläutert.

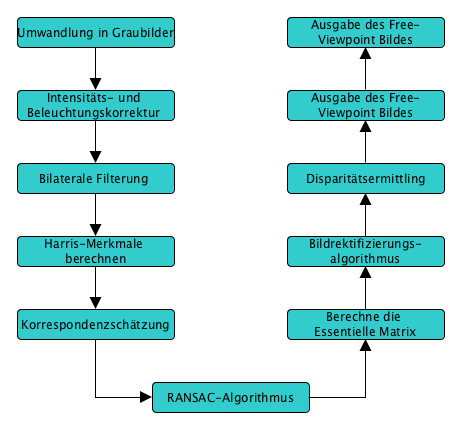




Abbildung 4: Ablauf des Programmes

Die Abbildung 4 zeigt den Aufbau und den Ablauf unseres Programms. Zuerst werden die beiden eingelesenen Bilder in Graubilder umgewandelt, um einen geringeren Aufwand bei den späteren Berechnungen zu erzielen. Danach wird eine Intensitäts- und Beleuchtungskorrektur bei beiden Bildern durchgeführt, um den vollen Intensitätsbereich ausnutzen und damit spätere Eckendetektionen zu erleichtern. Dieselbe Aufgabe verfolgt die darauffolgende bilaterale Filterung, bei der das Bild zwar weichgezeichnet wird, jedoch Kanten und Ecken erhalten bleiben. Im Anschluss erfolgt die Kanten- und Eckendetektion mit Hilfe des Harris-Detektors. Mit den detektierten Ecken aus den beiden Bildern kann eine Korrespondenzschätzung durchgeführt werden, um die Eckpunkte aus dem einen Bild den jeweiligen Eckpunkten aus dem anderen Bild zuzuordnen. Mit dem RANSAC-Algorithmus werden aus den geschätzten Korrespondenzen die robusten Korrespondenzpunktpaare bestimmt, um höhere Genauigkeit bei der Bestimmung der essentiellen Matrix zu erzielen. Diese erfolgt direkt im Anschluss mit Hilfe des Achtpunktalgorithmus. Durch die essentielle Matrix können die Rotation und Translation zwischen den beiden Ansichten berechnet werden. Diese werden zur Berechnung der dritten Ansicht benötigt. Denn mit der Rotation und Translation können die beiden Bilder rektifiziert werden, also eine Transformation durchgeführt werden, bei der die Epipolarlinien kollinear und parallel zur horizontalen Achse positioniert werden (Roth 2008). Somit wurden die Kamerabewegungen zwischen den beiden Bildaufnahmen ausgeglichen und die beiden Ansichten zeigen eine frontale Aufnahme nur um einen gewissen Abstand versetzt. Mit diesen rektifizierten Bildern können die Disparitäten bzw. eine Depth-Map berechnet werden. Aus dieser kann dann eine dritte rektifizierte Ansicht berechnet werden, welche im Anschluss wieder derektifiziert werden muss. Ansonsten passt die Ansicht nicht zu den beiden zu Beginn übergegebenen Bildern. Zuletzt wird das berechnete virtuelle Bild dem Benutzer ausgegeben.

Im Folgenden werden die hier kurz beschriebenen Schritte ausführlich beschrieben und erläutert.

## Umwandlung der Bilder in Graubilder

Nach der Übergabe der beiden Bilder, werden diese zunächst in Graubilder umgewandelt. Dies erlaubt in den späteren Schritten einfachere Berechnungen und somit auch einer kürzeren Berechnungszeit. Dazu wird die Funktion *rgb\_to\_gray* aufgerufen, die überprüft, ob es sich bei den beiden übergegebenen Bildern schon um Graubilder oder Farbbilder handelt. Bei Graubildern wird die Umwandlung übersprungen, bei Farbbildern werden die Intensitätswerte der drei Farben mit den in der Vorlesung vorgestellten Faktoren multipliziert und im Anschluss zusammenaddiert. Daraus entstehen dann die jeweiligen Intensitäten des berechneten Graubildes, welches von der Funktion zurückgegeben wird.

## Intensitäts- und Beleuchtungskorrektur

Die spätere Eckendetektion mit Hilfe des Harris-Algorithmus basiert auf lokale Intensitätsunterschiede innerhalb der Bilder. Um bei dem Algorithmus bessere Ergebnisse zu erzielen, soll der komplette Intensitätsbereich in den Bildern genutzt werden. Deshalb wird nach Graubildumwandlung die Funktion *gain\_offset\_correction\_cdf* aufgerufen, die die Skalierung der Intensitätswerte mit Hilfe der Histogramme der beiden Bilder vornimmt. Dazu werden die Histogramme der beiden Bilder ausgelesen und im Anschluss über den gesamten Intensitätsbereich gestreckt. Dadurch werden auch die einzelnen Intensitäten in den Bildern verändert bzw. auf den neuen skalierten Wert aus dem Histogramm gesetzt.

## Bilaterale Filterung

Um die Ergebnisse der anschließenden Eckendetektion mit dem Harris-Algorithmus weiter zu verbessern bzw. die Voraussetzungen zu optimieren, werden die beiden Bilder einer bilateralen Filterung unterzogen. Diese zeichnet die Bilder weich, wobei Kanten und Ecken erhalten bleiben, aber Rauschen reduziert wird. Somit treten diese noch mehr hervor und sind besser vom restlichen Bild zu erkennen. Dazu wird die Funktion *bfltGray* aufgerufen, welche die bilaterale Filterung auf einem durchführt. Für die Filterung wird ein Gaußfilter verwendet, der auf einzelne Bildfenster angewendet wird. Dabei wird der Kontrast innerhalb der einzelnen Bildfenster berücksichtigt, damit hohe Kontrastunterschiede wie Ecken und Kanten erhalten bleiben. Da die Filterung einen großen Teil der gesamten Programmlaufzeit (leider nicht der Fall) einnimmt, in der dem Benutzer keine neuen Zwischenergebnisse angezeigt werden können, wird diesem ein Fortschrittsbalken angezeigt. Dieser zeigt dem Benutzer den prozentualen Fortschritt bei den Berechnungen an, so dass nicht der Anschein aufkommen kann, das Programm hätte sich eventuell auf gehangen.

Wie oben beschrieben, stellt die bilaterale Filterung einen großen Teil der Programmlaufzeit dar(Nicht mehr der Fall) . Deshalb werden vor der Filterung die beiden Bilder durch die Funktion *downsample* auf eine geringere Auflösung bzw. Pixelanzahl heruntergerechnet und im Anschluss der Filterung durch die Funktion *upsample* wieder hochgerechnet. Durch die geringere Anzahl an Pixeln müssen weniger Berechnungen bei der bilateralen Filterung gemacht werden, weshalb die benötigte Berechnungszeit sinkt. In unserem Fall hat sich die Reduzierung der Auflösung um einen Viertel als optimal herausgestellt, wobei ein Mittelweg zwischen Berechnungsreduzierung und Qualitätsverlust bei den Ergebnissen betrachtet werden musste.

Das Herunterrechnen innerhalb der Funktion *downsample* erfolgt durch das Zusammenfassen von Pixeln und der Berechnungen eines neuen Intensitätswert, der die vorherigen Werte abbildet. Dabei liegt die Gewichtung auf den zentralen Pixeln innerhalb des jeweiligen betrachteten Fensters, welche durch einen Gaußfilter vorgenommen wird. Bei der Funktion *upsample* erfolgt das Gegenteil, es werden Zwischenwerte zwischen den einzelnen Intensitätswerten bestimmt. Dies erfolgt durch eine Interpolation.

## Harris-Merkmale berechnen

Nach den Vorbereitungen und Optimierungen für eine gute Eckendetektion innerhalb der Bilder, erfolgt diese durch den Harris-Detektor, der in der Funktion *harris\_detektor* implementiert wurde. Dieser wurde anhand der Vorlesung und den Hausaufgaben umgesetzt. Zunächst werden die Bildgradienten in x- und y-Richtung bestimmt, worauf eine Gewichtung der zentralen Pixel durch einen Gaußfilter und die Bestimmung der Harris-Matrix folgt. Mit dieser Matrix kann die Merkmalsdetektion durchgeführt werden, in dem nur Pixel als Merkmale klassifiziert werden, die nach der Harrismessung unter einem Schwellwert tau liegen. Danach erfolgt eine weitere Aussortierung bei den Merkmalen. Es werden nur Merkmale gespeichert, die innerhalb eines Fensters den größten Wert haben und somit die höchste Wahrscheinlichkeit für eine klare Ecke besitzen. Die anderen Merkmale innerhalb des jeweiligen betrachten Fensters werden nicht gespeichert. Im Anschluss gibt es eine Liste mit allen gefundenen Merkmalen in den beiden Bildern, welche in die Bilder eingezeichnet und dem Benutzer angezeigt werden.

## Korrespondenzschätzung

Für die Korrespondenzschätzung wird der aus der Vorlesung bekannte RANSAC-Algorithmus verwendet. Die Funktion *F\_ransac.m* implementiert den RANSAC-Algorithmus ins Programm, um die robusten Korrespondenzpunktpaare zu bestimmen. Nach der Definition des Input\_Parsers wird der RANSAC-Algorithmus vorbereitet und die Parse-Funktion ausgeführt.

Der RANSAC-Algorithmus beinhaltet die zufällige Auswahl an acht Korrespondenzpunktpaaren, die Schätzung der Fundamentalmatrix und die Berechnung der Sampson-Distanz. Nach der Bestimmung der im Consensus-Set erhaltenen Paare und der aktuellen Set-Distanz, werden diese mit der bisher größten Set-Größe in *largest\_set\_size* gespeichert. Ist das aktuelle Set größer, wird dieser als des neuen größten übernommen. Sind beide gleich groß, wird die Set-Distanz mit der in *largest\_set\_size* bisher größten Sets gespeicherten verglichen. Das Set mit der kleineren absoluten Samspon-Distanz soll somit das neue größte Set werden.

Die Fundamentalmatrix F, mit der das neue größte Set generiert wurde, wind in *largest\_set\_F* gespeichert. Mit der Funktion *sampson\_dist* wird die Sampson-Distanz basierend auf der Fundamentalmatrix F berechnet. Dazu wird ein vorher festgelegter Dach-Operator verwendet. Dieser Dach-Operator wandelt einen 3-Komponenten Vektor in eine schiefsymmetrische Matrix. Im Anschluss werden die robusten Korrespondenzpunktpaare ausgegeben.

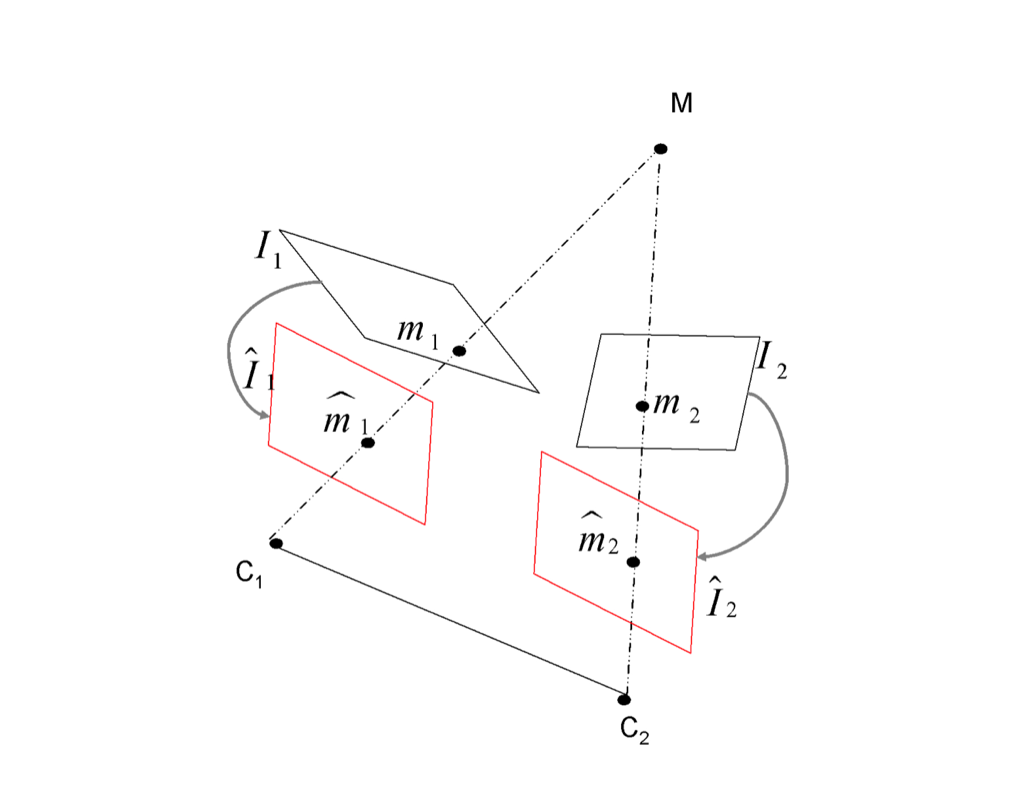
## Berechne die Essentielle Matrix

Die Kamerakalibrierungsmatrix ist in der Datei calib\_k.m abgespeichert und wurde mit der in der Video-Vorlesung angegebenen Toolbox bestimmt. Welche Toolbox!? =>Seb und Julia fragen

## Bildrektifizierungsalgorithmus

Ziel ist es die Epipole beider gegebenen Punkte gegen unendlich laufen zu lassen. Hier werden Bildpunkte der einzelnen Objekte jeweils der verschiedenen Bilder zugeordnet.

Bilder werden so gedreht, dass sie Parallel zueinander sind. (Paper Overview (erstes))

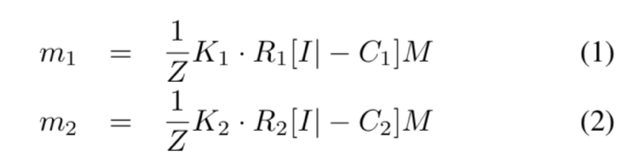


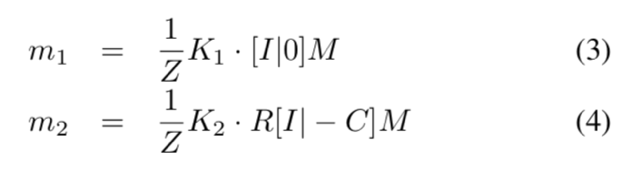
Parallele Epipolarlinien. 🡪 Suche auf den Linien nach gleichen Merkmalen (NNC-SAD)

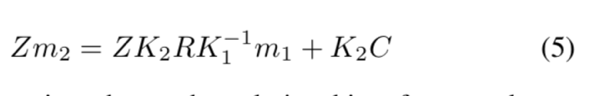
Sad\_scanline nicht mehr vorhanden => Ersetzt durch disparity\_estimation.m

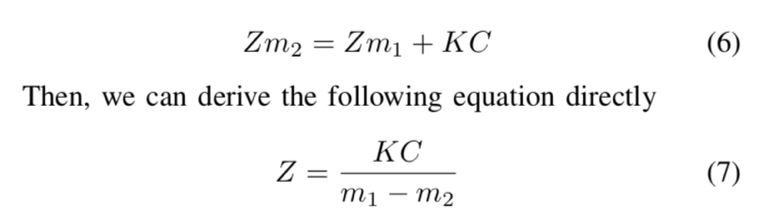
* SAD verwendet da schneller zu berechnen als NNC und Bias-Gain-Unterschiede der Bilder bereits vorher durch gain\_offset\_correction\_cdf behandelt wurden

Die Funktion sad\_scanline.m nnc\_scanline.m sorgt dafür, dass um ein Pixel ein gewisser Bereich (Window) gelegt wird, welches mit einem Bereich des anderem Bildes verglichen wird. Den Bereichen, die die geringste Abweichung aufweisen wird die gleiche Zugehörigkeit zugeordnet. Formel 1-7 Paper s.1025 , Berechnung Abstand, ... ,









SAD wird in downgesampelten Bildern berechnet da sonst Berechnung zu lange dauert

Tiefenkarte Resultat, dadurch sind die Punkte im Raum bekannt und diese können auf die dritte, neue Fläche projiziert werden.

Der Bildrektifizierungsalgorithmus besteht Hauptsächlich aus der *rectification.m* Funktion. Diese Funktion berichtigt die Graubilder img1 und img2. Die Parameter, welche vorausgesetzt werden, sind eine Essentielle- oder Fundamentalmatrix. Falls eine Essentielle Matrix übergeben wird, wird zusätzlich die Kalibrierungsmatrix K benötigt, sonst die Korrespondenzpunktpaare.[3] In unserem Fall übermitteln wir der Funktion die Essentielle Matrix mit einer Kalibrierungsmatrix, bekannt aus den vorherigen Schritten.

Der Algorithmus für die Rectification wird über die Funktion *rectiy\_fusiello.m* aufgerufen. [4]

Nach der Festlegung der Epipole und der Ausrichtung der X-, Y-, Z-Achsen werden die Weltkoordinaten in Pixelkoordinaten umgewandelt. Anschließend wird eine Rotation der Matrix jeweils für das linke und Rechte Bild durchgeführt. Aus der Funktion resultieren Homographien der einzelnen Bilder.

Im Anschluss werden die Schnittpunkte der optischen Achse und der Bildebene aufeinander ausgerichtet und die Bildgröße beider Bilder angepasst. Durch die Berechnung und Drehung der Bildmitte, liegen die Bildebenen parallel zum Translationsvektor. Der vertikale Versatz der begradigten Bilder (Y-Achse) wird so angepasst, dass die Bilder vertikal auf der gleichen Höhe sind. Zur Interpolation der Bilder werden zuerst die Bildausschnitte in das neue Koordinatensystem verschoben und daraufhin zurücktransformiert. Hierfür müssen die Bildecken bekannt sein. Diese werden, da die Bilder aktuell gleich groß sind, berechnet. Anschließend wird der Bildausschnitt, der aktuell beide Bilder enthält, ermittelt. Im Anschluss müssen die Corner zusammengefügt werden. Die Grenzen werden so festgelegt, dass die schwarzen Ränder abgeschnitten werden. Die Differenz wird in *offset\_x* zwischengespeichert, um zu einem späteren Zeitpunkt damit die Disparität zu berechnen. Für die Derektifizierung wird eine Ansicht bestimmt, die das gesamte rektifizierte Bild zeigt, ohne Abgrenzungen in y-Richtung. Für die Depth-Map wird der Abstand benötigt, um den die Derektifizierung größer ist. Zur Interpolation wird ein Meshgrid benötigt, welches zurücktransformiert werden kann. Bei der Interpolation werden die Stützstellen, an denen Werte bekannt sind, dazu benutzt, um unbekannte Werte an gewissen Stellen der Bilder zu bestimmen.

Depth\_estimation.m

% DEPTH\_ESTIMATION Die Tiefen werden mit Hilfe den downgesampelten Bilder

% berechnet. Zunächst wird die geringste Auflösung (Layer 3, 2x

% Downsampling) verwendet. Mit Hilfe von SAD werden zunächst grob die

% Disparitäten bestimmt. Gleichzeitig wird die Varianz jedes

% Fensterausschnitts verwendet, um homogene Flächen zu bestimmen. Die

% Disparitäten der homogenen Flächen werden nicht genauer bestimmt, da dies

% auch nur schwer möglich ist. Die restlichen Stellen werden hingegen in

% der nächst höheren Auflösung (Layer 2) genauer definiert. Um Rechenzeit zu sparen

% werden die grob bestimmten Disparitäten der niedrigeren Auflösung

% verwendet, um für jedes Pixel ein (kleineres) Liniensuchintervall zu

% erhalten (=> Erneute Suche mit refine\_disp.m).

% Für kleine Disparitäten kann auch ein gröbere Depth Map ausreichen und

% Rechenzeit gespart werden. (Fall mean\_disparity <= 1500)

% Quelle:

% Fast Computation of Dense and Reliable Depth Maps from Stereo Images

% M. Tornow, M. Grasshoff, N. Nguyen, A. Al-Hamadi and B. Michaelis

Um die Tiefen zu schätzen, wird linienweise zu jedem Pixel die Disparität bestimmt. Minimaler und maximaler Liniensuchbereich wird grob über die robusten Korrespondenzpunktpaare geschätzt (get\_stat\_disparity.m) => Gibt auch den Mittelwert der Disparitäten zurück. Bei kleinem Mittelwert ist es wahrscheinlicher dass eine niedrige Auflösung zur Bestimmung der Disparitäten ausreicht als bei großem Mittelwert:

Mean\_disparity <= 1500: Verwende ein Viertel der Auflösung zur Disparitätsermittlung

Mean\_disparity > 1500: Verwende zunächst ebenfalls ein Viertel der Auflösung (Layer 3), berechne während der SAD Liniensuche ebenfalls die Standardabweichung jedes Fensters: std < 3 => homogene Fläche wurde detektiert…führe erneut SAD in der nächst höheren Auflösung durch (Layer 2) allerdings nur an den Stellen, an denen keine homogene Fläche detektiert wurde. Disparität ist durch Layer 3 bereits grob festgelegt. Nutze dies bei der Disparitätssuche bei Layer 2, indem Layer 3 dazu verwendet wird, das Liniensuchintervall zu verkleinern (Minimum/Maximum innerhalb eines Fenster bestimmen die neuen Suchgrenzen für das jeweilige Pixel) => Anschließend wird Disparity Map zusammengesetzt. Für alle Pixel, deren Disparitäten nicht im Layer 2 genauer bestimmt wurden (refined wurden) (homogene Flächen), werden die Disparitäten von Layer 3 übernommen. (siehe InTech-Paper: **4.7 Multi-layer correspondence search) …continuity\_filter.m (siehe 4.6 Applying the continuity constraint)**

Projection.m => Rektifiziere die bekannten Ecken (Corners) der virtuellen Ansicht, dadurch ist festgelegt welcher Bildauschnitt später derektifiziert werden muss.

Zunächst neue Homographie für Rektifizierung bestimmen. Neue Rotationsmatrix wird anteilig mit Hilfe von p und den maximalen Winkeln, die bei der Rotation von linker zu rechter Ansicht auftreten, bestimmt. (euler\_rotation.m,euler\_angles.m)

Folgende Transforamtion wird auf Bildecken der virtuellen Ansicht angwendent, um deren Position in rektifizierten Pixelkoordinaten zu erhalten.

% Umwandlung der derektifizierten Pixelkoordinaten x in rektifizierte

% Pixelkoordinaten x':

% x' = K\*R\_rect'\*R\_v\*inv(K)\*x

% ("Derektifizierte Ansicht zunächst um R\_rect nach rechts drehen und

% anschließend mit R\_v um Winkel alpha nach links drehen")

% Bzw. inverse Transformation:

% Umwandlung der rektifizierten Pixelkoordinaten x' in derektifizierte

% Pixelkoordinaten x:

% x = K\*R\_rect\*R\_v'\*inv(K)\*x'

% ("Rektifizierte Ansicht zunächst um R\_rect nach links drehen und

% anschließend mit R\_v um Winkel alpha nach rechts drehen")

Linke und rechte rektifizierte Ansicht werden nun in die neue rektifizierte Ansicht verschoben. (new\_rectified\_view.m)

Pixelkoordinaten der linken/rechten Ansicht werden folgendermaßen in neue rektifizierte Ansicht umgerechnet (nur x-Pixelkoordinate wird verschoben, da rektifiziert):

% Tiefe wurde bisher bestimmt als: depth = 1/(x1\_pixel-x2\_pixel). Der erste

% Eintrag der Kalibrierungsmatrix K(1,1) entspricht dem Umrechnungsfaktor

% zwischen Abständen der x-Weltkoordinaten und x-Pixelkoordinaten:

% d\_x\_pixel = K(1,1)\*d\_x\_welt

% Daraus folgt für die "echte" Tiefe: depth = K(1,1)/(x1\_pixel-x2\_pixel)

% Daraus folgt:

% lambda1 = lambda2 = lambda = z-Komponente der Punkte im Raum

% = K(1,1)\*depth\_map'

% Umformen von Gleichung(1) ergibt die neuen Pixelkoordinaten:

% x2\_pixel = x1\_pixel + T\_pixel/lambda

% => x2\_pixel = x1\_pixel + (K(1,1)\*T\_x) / ((K(1,1))\*depth\_map)

% = x1\_pixel + T\_x / depth\_map

Ausgehend von einer Ansicht (links oder rechts):

Da die Tiefen nicht exakt sind ist nicht jeder Pixelkoordinate in der neuen rektifizierten Ansicht unbedingt eindeutig eine Intensität zugeordnet. Mehrere Intensitäten können auf die selbe neue Pixelposition verschoben worden sein, während an manchen neuen Pixelkoordinaten sich gar keine Intensität befindet. Mehrere Intensitäten mit der selben neuen Pixelkoordinate werden deshalb gemittelt. (mit accumarray)

Dies wird für beide Fällen (ausgehend von linker und rechter Ansicht) durchgeführt.

An Stellen, die von beiden Ansichten aus zu sehen sind, sind die Intensitätinformationen nun aus beiden Ansichten vorhanden. Die Intensitäten werden je nach Position mit p und (1-p) gewichtet (Vertraue linker Ansich mehr, falls p < 0.5, sonst vertraue mehr der rechten Ansicht). Stellen, die jeweils nur aus einer der Ansichten zu sehen sind werden unverändert mit zur Interpolation verwendet, sodass keine größere Löcher enstehen, wie es bei der Berechnung von nur einer Ansicht aus der Fall wäre.

* Evt Plots von Depth Map hinzufügen: Entsprechende Zeilen für Plot auskommentieren

## Disparitätsermittlung

Hier wird die Funktion *get\_min\_max\_disparity.m* aufgerufen. Diese berechnet die grob minimale und maximale Disparity um den Liniensuchbereich auszugeben. Als erstes werden die Koordinaten in homogene Pixelkoordinaten umgewandelt. Auf die umgewandelten Korrespondenzpunktpaare wird die Rectification-Transformation angewendet. Im Anschluss liegen die Korrespondenzpunkte des linken Bildes jeweils weiter rechts als der des rechten Bildes.

## Projection

Die Projection.m berechnet neue rektifizierte Zwischenansichten und führt anschließend eine Derektifizierung durch.

## Ausgabe des Free-Viewpoint Bildes

Die Bildgröße des dritten, berechneten Bildes wird auf die Bildgröße der gegebenen Bilder angepasst.

>>>>>>>>Hier werden die Bilder eingefügt aus den verschiedenen Blickwinkeln!!<<<<<<<<<

# Graphische Benutzeroberfläche (GUI)

Die graphische Benutzeroberfläche dient dem Bediener zur Steuerung des Programmes. Für ein schnelles zurechtfinden befinden sich auf der Oberfläche nur die wichtigsten Operatoren.



Abbildung 5: Benutzeroberfläche vor der Berechnung

Nach einer kurzen Erklärung, was dieses Programm leistet und wie es zu bedienen ist, kann der Blickwinkel, Parameter *p*, von 0 bis 1 eingestellt werden. Die richtige Eingabe wird durch eine minimale und maximale Grenze in dem Eingabefeld sowie einem Hinweis hinter dem Feld gewährleistet. Die gewünschte linke und rechte Ansicht wird jeweils über die Buttons *Linke Ansicht laden* und *Rechte Ansicht laden* ausgewählt. Dazu wird eine Explorer-Ansicht geöffnet, um die Bildauswahl für den Benutzer zu vereinfachen. Nach der Auswahl werden die Bilder in das Programm geladen und dem Benutzer zur Bestätigung jeweils unter den Buttons angezeigt. Zusätzlich ist der Pfad der Bilder direkt darunter einsehbar, damit der Benutzer kontrollieren kann, ob die richtigen Bilder geladen wurden und er sich nicht verklickt hat. Das dritte Bild kann dann über den Button *3. Ansicht berechnen* über das dahinter entworfene Programm berechnet werden. Nach der Berechnung wird das erzeugte Bild zwischen beiden zuvor geladenen Bildern angezeigt, um einen guten Vergleich zu den äußeren Ansichten zu geben. Die Berechnungszeit, die das Programm für die Berechnung der mittleren Ansicht benötigt, wird unterhalb des Bildes in Minuten angezeigt.

Die GUI wurde mit Hilfe des von Matlab zur Verfügung gestellten *App Designer* erstellt. Dieser bietet ebenfalls ein graphisches Interface im Sinne von *what you see, is what you get*. Dies erleichtert das Positionieren der einzelnen Elemente in dem Fenster. Zudem erzeugt es für die einzelnen Elemente den entsprechenden Code für deren Erzeugung. Danach konnten die Funktionen der Button und Eingabeoberflächen an den entsprechenden Stellen im Code implementiert werden. Das Einarbeiten in den App Designer erfolgte über die bereitgestellte Hilfe von Matlab.

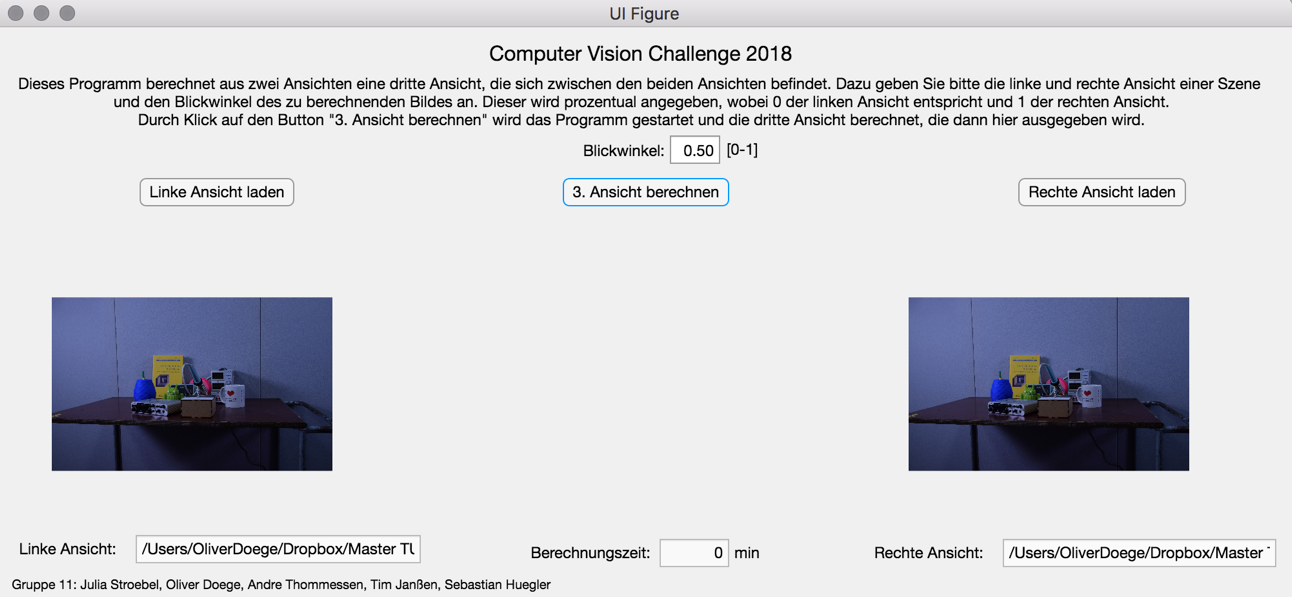


Abbildung 6: Benutzeroberfläche mit geladenen Bildern

# Abbildungsverzeichnis

[Abbildung 1: Geometrischer Zusammenhang zwischen den realen Bildern und einer künstlich generierten Ansicht 3](file:///C:\Users\OliverDoege\Desktop\Doku_Gr11_Challenge.docx#_Toc522452448)

[Abbildung 2: img/L1.jpg 4](file:///C:\Users\OliverDoege\Desktop\Doku_Gr11_Challenge.docx#_Toc522452449)

[Abbildung 2: img/R1.jpg 4](file:///C:\Users\OliverDoege\Desktop\Doku_Gr11_Challenge.docx#_Toc522452450)

[Abbildung 4: Projektverlauf 5](file:///C:\Users\OliverDoege\Desktop\Doku_Gr11_Challenge.docx#_Toc522452451)

# Literaturverzeichnis

[1] Roth, Dr. Gerhard. 2008. *Rectification*. Zugriff: 02. September 2018.  
 http://people.scs.carleton.ca/~c\_shu/Courses/comp4900d/notes/rectification.pdf

[3] http://www.sci.utah.edu/~gerig/CS6320-S2013/Materials/CS6320-CV-F2012-Rectification.pdf

[4] http://people.scs.carleton.ca/~c\_shu/Courses/comp4900d/notes/rectification.pdf