Bachelorarbeit

Entwurf und Implementierung eines einfachen, stereobildausschnitt basierten Entfernungsmessverfahrens zum Einsatz auf einer Marsrover-Plattform

Erstprüfer: Prof. Dr. Friedhelm Mündemann
Zweitprüfer: Dipl.-Inform. Frank Trauthan
Fachhochschule Brandenburg
Fachbereich Informatik & Medien

Bearbeitungszeitraum: 11.08.10 – 06.10.10
vorgelegt von:
Yves Schwarz

Eidesstattliche Erklärung

“Ich versichere, daß ich diese Abschlußarbeit ohne fremde Hilfe selbständig verfaßt und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen deutlich kenntlich gemacht.“

(Ort, Datum) (Unterschrift)

Danksagung

Ich möchte meiner Mutter und Oma danken, die mich auf meinem bisherigen Lebensweg stets unterstützt und gefördert haben. Dadurch habe ich meine Fähigkeiten und Interessen entdecken können. Hierzu zählt auch die Informatik. Weiterhin danke ich meinen beiden Betreuern für die weitreichende und unterstützende Begleitung bei der Entstehung dieser Arbeit.

Inhalt

1. Einleitung

„Ein kleiner Schritt für mich, aber ein großer Schritt für die Menschheit.“ Dieser Ausspruch von Neil Armstrong spiegelt sehr gut den Geist der Raumfahrt wieder. So gelang 1969 mit der Apollo 11 Mission die erste bemannte Landung auf dem Mond. Die ExoMars Mission welche 2018 Richtung Mars starten wird, ist zwar eine unbemannte Mission, jedoch wird sie durch ihre wissenschaftlichen Geräte maßgeblich dazu beitragen, dass spätere bemannte Missionen so gut vorbereitet werden können wie möglich.

Die ExoMars Mission ist eine von der ESA geplante Mission, die einen Mars Rover zum 4. Planeten in unserem Sonnensystem bringen soll. Dieser soll sich mittels der Panorama Kamera (PanCam) auf der Planetenoberfläche bewegen. Die Bilder, die mit dieser Kamera vom Mars Rover gemacht werden, nutzt dieser zur Routenplanung, für die Kontrolle der Systeme und Manöver, als auch für wissenschaftliche Untersuchungen der Marsoberfläche. Durch die Anforderung das nahe und auch weiter entfernte Objekte scharf und hochauflösend darzustellen sind, ist es notwendig, dass das Objektiv eine große Brennweite sowie automatische Fokussierungsmechanismen besitzt. Bedingt durch die extremen Einwirkungen während des Raketenstarts, der Flugphase zwischen Erde und Mars, als auch der Planetenoberfläche ist es notwendig, dass die Kamerasysteme an diese Umweltbedingungen angepasst werden müssen und dadurch nur Bauteile verwendet werden können, welche den gestellten Anforderungen gerecht werden. Die extremen Bedingungen sind besonders gekennzeichnet durch:

- starke Temperaturschwankungen während Tag und Nacht auf dem Mars
- sehr niedrigen Temperaturen auf dem Flug zum Ziel
- Weltraumeigenschaften wie dem Vakuum und erhöhten Strahlungsbelastungen
- hohen Beschleunigungskräften während der Landung
- auftretende Geräusche und Vibrationen beim Start

Bereits bestehende Autofokuskameras können aus dem oben genannten hier nicht verwendet werden. Die High Resolution Kamera (HRC) wird eines der ersten fokussierbaren Kamerasysteme sein, was außerhalb der Erdatomsphäre eingesetzt werden kann.

Diese Bachelorarbeit wird sich primär mit dem Matching und der Implementierung eines einfachen stereobildausschnitt basierenden Entfernungsmessverfahrens befassen. Die

Ergebnisse, die durch diese Messung entstehen, können wiederum für die HRC genutzt werden, um diese vorfokussieren zu können. Grundlagen für die Stereoanalyse werden die Standardalgorithmen wie z.B. Korrelationsalgorithmus und Template Matching sein. Die Optimierung wird hierbei durch die Epipolargeometrie erfolgen. Diese wird dazu beitragen, dass sich die Laufzeit auf dem Mars Rover verbessern kann.

Die Bachelorarbeit gliedert sich in insgesamt drei Kapitel. Das erste Kapitel gibt einen Überblick über die ExoMars Mission, deren Zielsetzungen, den Mars Rover, den Kamera Systemen sowie Grundlegendes der Optik. Dies ist notwendig, um die Aufgaben der verwendeten Wide Angle Kameras (WAC) und der HRC in der Mission darstellen zu können. Das zweite Kapitel befasst sich mit der Umsetzung der Stereoanalyse in einem Computerprogramm. Hierbei werde ich Möglichkeiten zur Aufgabenrealisierung darstellen, um zu zeigen, warum bestimmte Algorithmen ausgewählt wurden. Des Weiteren werden die Korrelationsalgorithmen näher beleuchtet. Es wird dabei auf den Pyramid-, den Korrelation- und Template Matching Algorithmus eingegangen. Diese drei werden dabei genau auf ihre Funktionsweise, der Umsetzung und ihrer Leistung hin untersucht, um somit einen Vergleich durchführen zu können, um zu sehen welcher am besten geeignet ist. Am Schluss dieses Kapitels wird dann noch auf die Berechnung der Disparitäten eingegangen. Also wie die Entfernungen mittels der gefundenen Punkte berechnet werden können. Das dritte Kapitel ist eine Zusammenfassung und gibt einen Ausblick auf mögliche Weiterentwicklungen dieses Programms.

1.1 ExoMars Mission

Diese Mission ist die erste des Aurora Projektes, die von der Europäischen Weltraumorganisation (ESA) organisiert wird (ESA 07). Das Projekt wurde vom Europäischen Forschungsrat zusammen mit der ESA gestartet. Die Ziele dieses Programms sind es, neue Technologien zu entwickeln, mit denen es möglich ist, bemannte und unbemannte Missionen zum Mond, Mars und erdnahen Weltraumobjekten zu machen, um auf diesen weitere Forschungen durchführen zu können. Das Aurora Projekt beinhaltet hierbei folgende Missionen (ESA 04):

- ExoMars
- Entry Vehicle Demonstrator
- Mars Sample Return
- Bemannte Mondmission
- Bemannte Marsmission

Durch den Entry Vehicle Demonstrator (EDV) sollen Abläufe erprobt werden, die bei der Mars Sample Return Missionen verwendet werden können, um somit Bodenproben vom Mars zurückbringen zu können. Die Mars Sample Return Mission ist in den Jahren von 2018 bis 2022 geplant. Die anderen Ziele sind erst nach dieser Mission geplant. So ist die bemannte Reise zum Mond 2024 geplant und die Mission zum Mars 2033.

Die ExoMars Mission ist die erste des Aurora Projektes, die als Ziel den Mars hat. Dieser Planet ist der vierte in unserem Sonnensystem und ist der Nachbarplanet der Erde. Die Entfernung zwischen beiden schwankt hierbei zwischen 54,5 und 401,3 Millionen Kilometern. Ein Jahr auf dem Mars dauert genau 687 Erdtage, ein Tag 24 Stunden, 37 Minuten und 22 Sekunden. Sein Durchmesser beträgt am Äquator 6794 km, seine Masse hat das 0,107 fache der Erde. Die dünne Atomsphäre besteht aus 95% Kohlendioxid, 3% Stickstoff und 2% Argon. Der atomsphärische Druck beträgt im Vergleich zu dem auf der Erde nur 1/100. Die Färbung des Planeten entsteht durch die Eisenoxid Schicht auf der Oberfläche, welche ihn rot erscheinen lässt (Rees 07).

In diesem Unterkapitel wird genauer auf die Ziele der ExoMars Mission sowie dem Mars Rover eingegangen. Dies geschieht um die Kamerasysteme besser in das Gesamtbild der Mission einordnen zu können.

1.1.1. Ziele

Die Ziele dieser Mission bestehen aus wissenschaftlichen- und technologischen, welche nachfolgend aufgezählt werden.

Die wissenschaftlichen Ziele sind (ESA 00):

1. Suche nach Zeichen für vergangenes oder aktuelles Leben auf dem Mars.
2. Feststellung der chemischen Zusammensetzung des Bodens und des Untergrundes des Mars.
3. Bestimmung möglicher Risiken, welche durch die Marsumwelt für künftige bemannte Missionen entstehen können.
4. Untersuchung des Bodens und des Planeteninneren, um mehr über seine Entstehung und mögliche Bewohnbarkeit in Erfahrung zu bringen.

Die technologischen Ziele sind:

1. Realisieren einer Landung mit schwerer Last auf dem Mars.
2. Zeigen der Beweglichkeit und Bewerkstelligung von Bohrungen in bis zu zwei Meter tiefe auf dem Mars.
3. Sammeln von Daten für die Sample Return Mission.

1.1.2. Mars Rover

Der Mars Rover wird mit einer Raumsonde – siehe Abbildung 1: Raumsonde (ESA 00)– von der Erde zum Mars gebracht. Das Abstiegsmodul bringt die gesamte Nutzlast auf den Planeten und ermöglicht eine sichere Landung. Der Antrieb trennt sich kurz vor dem Eintritt in die Marsatomsphäre und gibt das Abstiegsmodul frei. Dieses Modul öffnet nach dem Eintritt Fallschirme um so eine Abbremsung zu ermöglichen. Kurz vor der Landung werden dann Luftpolster aufgebläht, um den Aufschlag abzumildern.

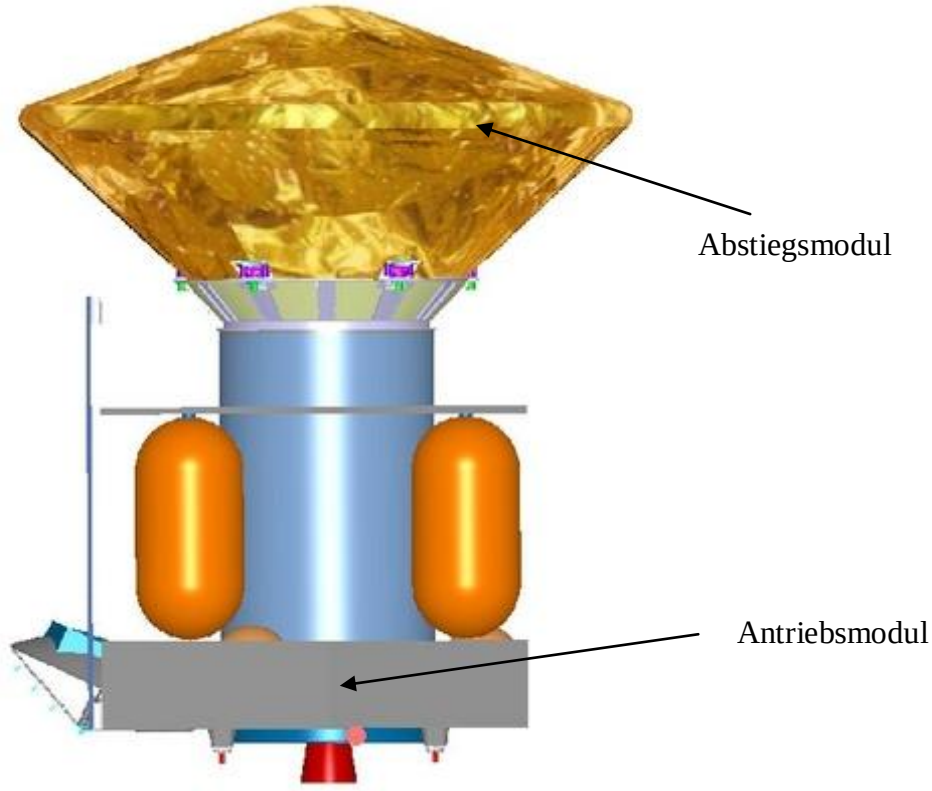

Abbildung 1: Raumsonde (ESA 00)

Das Abstiegsmodul besteht aus der Humboldt Nutzlast und dem ca. 210 kg schweren Mars Rover, auf welchem die Pasteur Nutzlast installiert ist – Abbildung 2:Humboldt Nutzlast und Mars Rover (ESA 07). Mit seinem Gewicht ist dieses Raumgefährt das komplexeste, welches bis dahin je auf dem Mars gelandet ist. Auf die Humboldt Nutzlast wird in dieser Arbeit nicht genauer eingegangen, da sie für die Stereoanalyse keine relevante Rolle spielt.

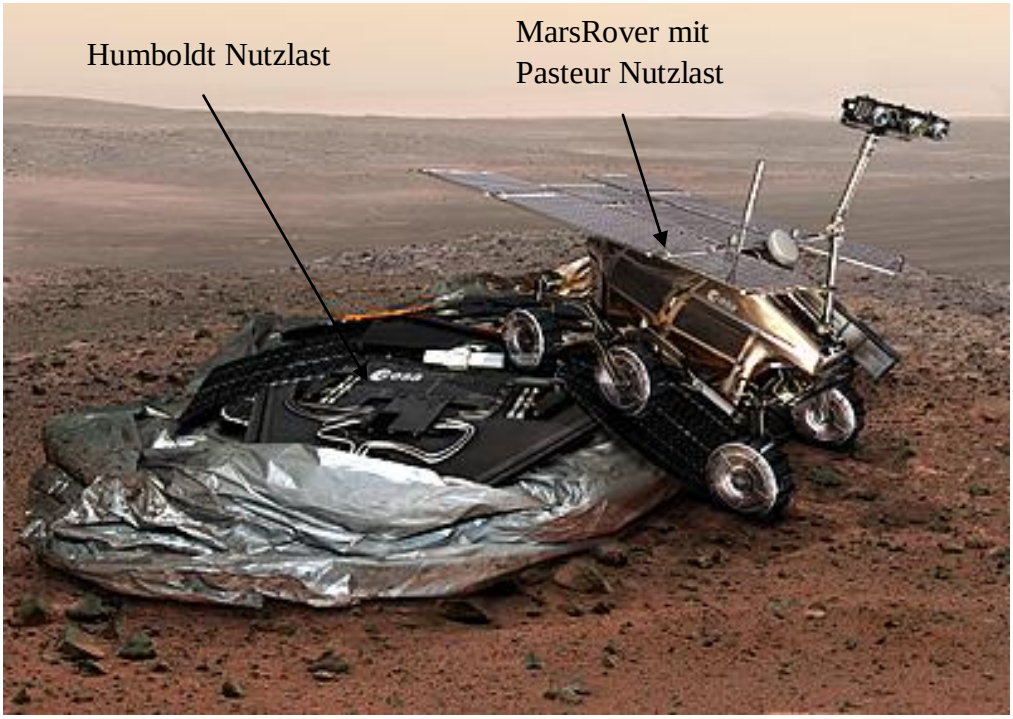

Abbildung 2:Humboldt Nutzlast und Mars Rover (ESA 07)

Der Mars Rover zusammen mit der Pasteur Last ist ein autonomes Fahrzeug, welcher sechs Räder besitzt. Es besitzt die Möglichkeiten sich frei auf der Oberfläche zu bewegen, Löcher zu bohren (mit einer Tiefe von bis zu zwei Metern), Bodenproben zu analysieren und hochauflösende Detail- und Panoramaaufnahmen des Mars anfertigen zu können. Diese Arbeit legt ihren Schwerpunkt auf das Matching der Bilder, um so die Entfernung zu bestimmen.

Abbildung 3: MarsRover (ESA 00)

Zusätzlich zu den beiden Kamerasystemen sind noch 11 weitere Instrumente auf dem Gefährt vorgesehen. Diese sollen dann exobiologische Untersuchungen durchführen und den Marsboden in bis zu zwei Metern Tiefe analysieren. Die Instrumente sind hierbei in Panorama-, Kontakt- und Analyse eingeteilt.

Panorama Instrumente:

- Panorama Kamera, welche 3D- sowie hochauflösende Detailbilder der Umgebung des Rovers machen soll
- WISDOM, die den Marsuntergrund mittels Radar untersuchen soll
- MIMA soll die mineralogische Zusammensetzung der Oberfläche erfassen, sowie die Zusammensetzung der Atomsphäre erforschen

Kontakt Instrumente:

- MIMOS 2 zur Erforschung der gesammelten Bodenproben mithilfe eines Mößbauer Spektrometers
- Close Up Imager zur Betrachtung der Gesteinsproben
- Raman Laser Spectroscope External Head Spektrometer zur Bestimmung der Zusammensetzung der Gesteinsproben. Des Weiteren sollen damit organische Verbindungen gesucht werden.
- Imaging Spectrometer and Drill Spektrometer, welcher am Bohrkopf angebracht ist, um die Eigenschaften festzustellen

Analyse Instrumente:

- Raman Laser Spektroskope Internal Head um die mineralische Zusammensetzung der Bodenproben feststellen zu können und um organische Verbindungen finden zu können
- MicrOmega ist ein Mikroskop, um die Bodenproben genauer betrachten zu können
- Martian organic detector and oxidant Instruments um organische Strukturen im Inneren der Steine zu suchen
- Mars X-Ray diffractiomenter zum Röntgen der Bodenproben und um nach Leben im Inneren des Gesteins zu suchen

Die Energieversorgung wird mithilfe von Solarmodulen und Batterien gewährleistet. Die Instrumente werden in den Nächten beheizt, um sie vor der Kälte zu schützen. Der Mars Rover muss seine Aufgaben autonom erledigen können, da die Kommunikation mit der

Erde nur ein- bis zweimal am Tag innerhalb eines kurzen Zeitfensters möglich ist. Hierbei wird diese durch die ESA, NASA und Satelliten, die sich im Marsorbit befinden, bewerkstelligt.

Der Mars Rover besitzt einen Leon 3/2 Prozessor mit 100 MHz. Dieser Prozessor besitzt keine Gleitkomma Einheit. Dieser Prozessor besitzt nur einen Kern, eine Speicherkontrolleinheit, UART, Spacewire Interface und Kommuniziert mit dem CPU-AHB Bus.

1.2. Kamera Systeme

Ein wichtiger Bestandteil des Mars Rover ist das Kamerasystem. Dieses besteht aus zwei Komponenten: der hochauflösenden Kamera (HRC), die im Deutschen Zentrum für Luft und Raumfahrt in Berlin Adlershof entwickelt wird und der Stereo Kamera (WAC). Beide Kameras, sowohl die HRC als auch die WAC, werden auf einen Mast am Fahrzeug montiert.

Des Weiteren wird es auch Kameras geben, die für die Navigation zuständig sind. Diese sollen mögliche Hindernisse erkennen und eine selbstständige Navigation ermöglichen

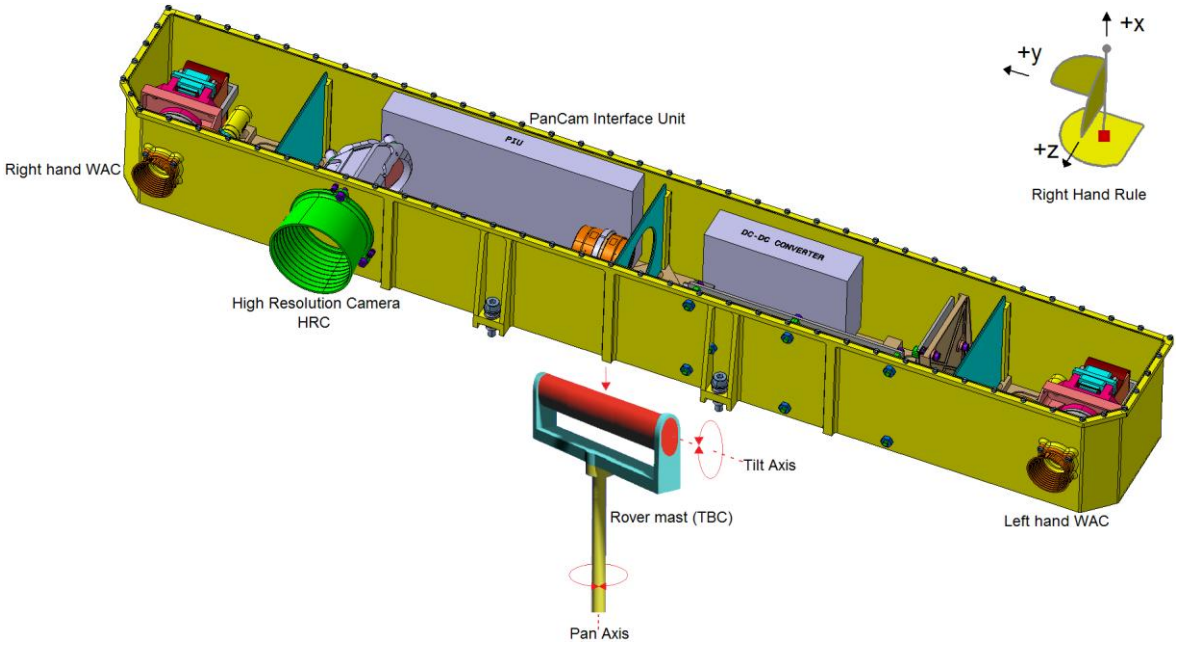

Abbildung 4:Kamerasysteme (DLR)

Die wichtigste Aufgabe der Panorama Kamera ist die Erfassung der Mars Rover Umgebung und das Finden von Objekten, bei denen sich eine weitere Untersuchung mit weiteren, auf dem Fahrzeug installierten Instrumenten lohnt.

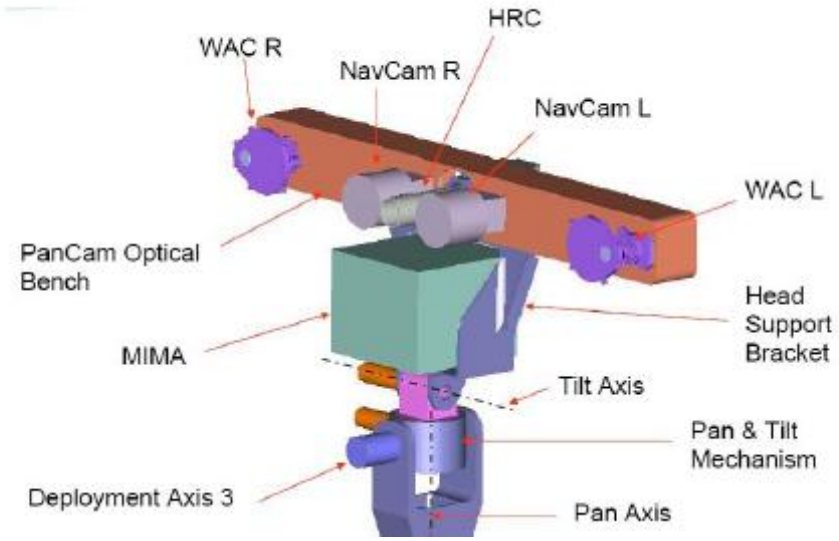

Abbildung 5: Mechanismen des Kamerasystems (DLR)

Die beiden Hauptkameras sind miteinander über die PanCam Interface Unit mit dem Raumfahrzeug verbunden. Dieses Interface ist mit dem Mars Rover verbunden und stellt die Stromversorgung sicher. Die Datenübertragung erfolgt hierbei über das Space Wire. Diese Schnittstelle ist ein Datenbus nach ECSS-E50-12A Standard und ermöglicht Datenraten von bis zu 200 MBit/s (esa 08).

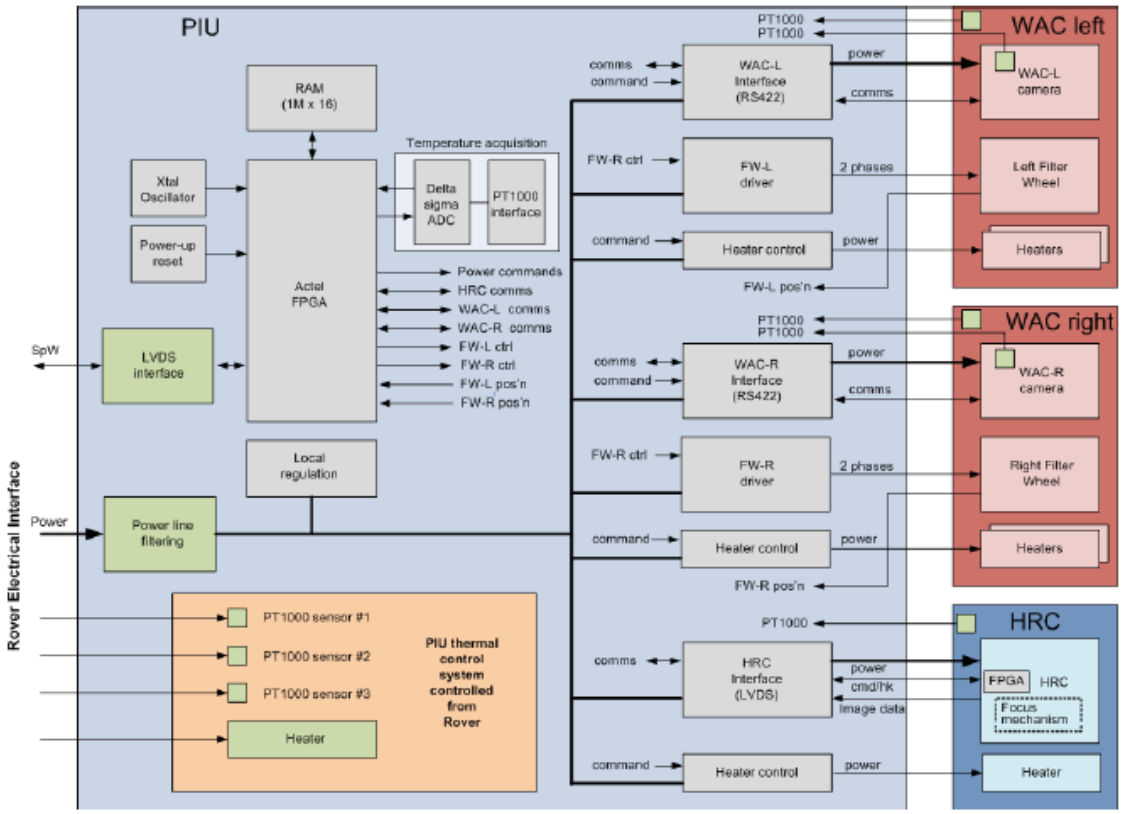

Abbildung 6: Blockschaltbild des Kamerasystems (DLR)

Die nächsten zwei Unterkapitel werden die HRC und die WAC genauer erklären und ihre Aufgaben und technischen Details aufzeigen.

1.2.1 HRC

Der Fokussierungsbereich dieser Kamera liegt zwischen 1,5 und 100 Metern. Allerdings können auch Panoramaaufnahmen gemacht werden, indem das Kamerasystem geschwenkt wird und mehrere Aufnahmen zusammengesetzt werden. Diese Bilder können dazu genutzt werden, Ziele zu identifizieren, die Umgebung zu erfassen und um Objekte, die auf dem Planeten zu finden sind, wissenschaftlich zu untersuchen. Darüber hinaus kann man mithilfe der Kamera den Mars Rover selbst überwachen. Auf dem nachfolgenden Bild kann man die HRC sehen.

Abbildung 7: High Resolution Camera (HRC) (DLR)

Auf diesem Bild kann man die Kamera ohne Verkleidung sehen. An dieser Stelle sieht man den Sensor, den Motor der zur Fokussierung dient und die Ansteuerungselektronik.

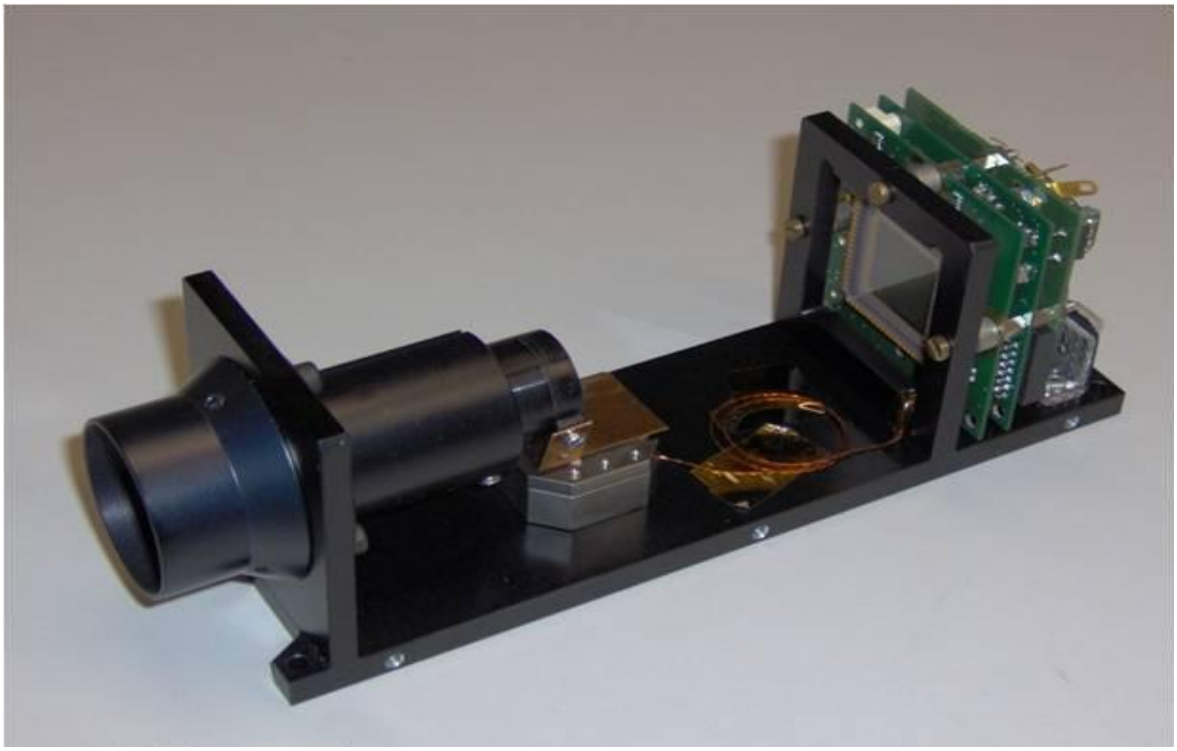

Abbildung 8: HRC ohne Verkleidung (DLR)

Als Bildsensor wird der Star1000 verwendet, dies ist ein Graustufensensor mit einer Auflösung von 1024 x 1024 Pixeln. Das generierte Bild wird eine Bildtiefe von 10 Bit haben. Somit ist es möglich 1024 verschiedene Grauwerte dazustellen. Auf dem Sensor wird ein Filter mit einem blauen, panchromatischen und roten Bereich angebracht sein. Der panchromatische Bereich lässt nur Licht mit einer Wellenlänge von 400 nm bis 780 nm auf den Sensor fallen. Dieser Bereich kommt dabei dem menschlichen Sehen sehr nah.

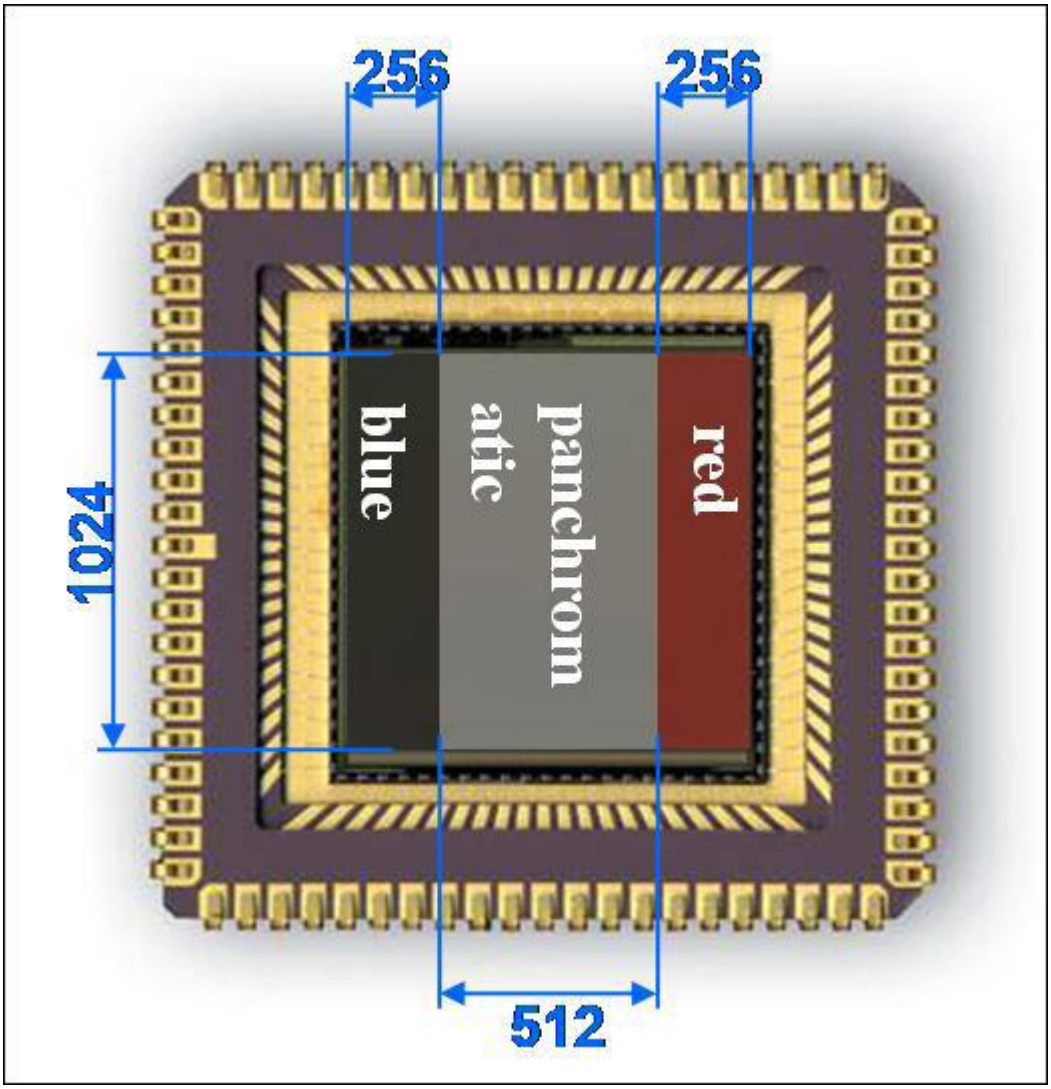

Abbildung 9: Star1000 Bildsensor mit Farbfilter (DLR)

Um Farbbilder zu erstellen, müssen mehrere Bilder von einem Objekt angefertigt werden. Diese müssen dann überlagert und der panchromatische Bereich muss angepasst werden.

1.2.2. WAC

Diese Kamera ist das zweite Instrument des Panorama Kamerasystems. Der Verbund besteht aus zwei Kameras, welche 50 cm voneinander entfernt sind, wodurch Stereobilder ermöglicht werden. Mithilfe dieser Bilder kann ein dreidimensionales Umgebungsmodel erstellt werden. Dieses Modell bringt Informationen, die für die Wegplanung des Raumfahrzeugs auf dem Mars sehr wichtig sind. Die Kameras besitzen eine Auflösung von 1024 x 1024 mit einem Grauwert CDD. Farbbilder können generiert werden, indem verschiedene Farbfilter, mittels eines Drehrades, vor den Sensor geschoben werden. Außerdem müssen mehrere Bilder miteinander überlagert werden, um so ein Farbbild zu erstellen. Das Blickfeld der Kamera beträgt 34^{O} und ist somit größer als das der HRC. Durch dieses große Blickfeld, ist die Wide Angle Camera besonders geeignet für Panoramabilder. Die Abbildung zeigt den mechanischen Aufbau der Kamera.

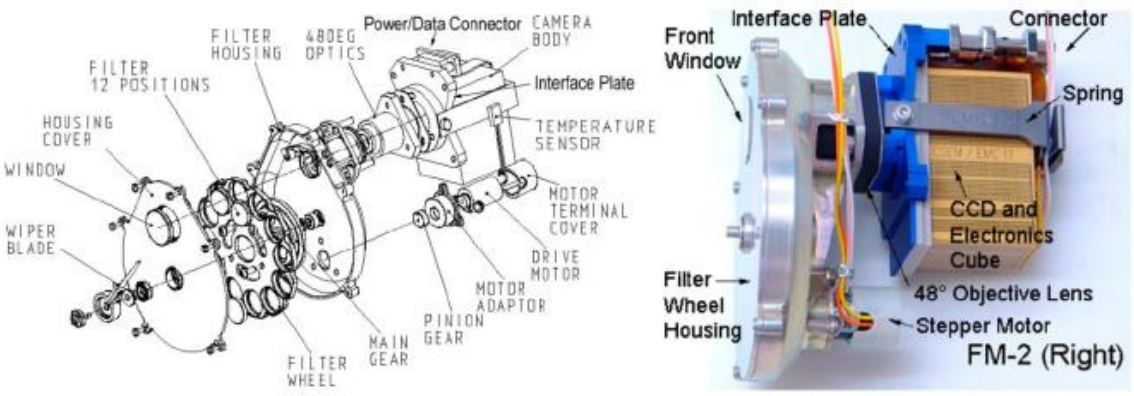

Abbildung 10: WAC Aufbau (DLR)

Die WAC wurde am Mullard Space Laboratory der Universität in London entwickelt.

Tabelle 1: Daten HRC und WAC (DLR)

	HRC	WAC
Sensor:	1024x1024 Graustufen APS Sensor mit aufgeklebten Farbfilter	1024x1024 CCD Graustufen Sensor
Blickfeld:	8°	34°
Aussteuerbereich:	10 Bit = 1024 Grauwerte	10-16 Bit (variabel)
Datenvolumen:	32Mbit/img (Farbbild)	10Mbits/img
Durchschnittliche Leistung:	0.9W	1.80W
Masse (Breadboard):	415g	363g
Einsatztemperatur	-70°C - +70°C	-120°C - +50°C
Ruhebereich Temperatur:	-150°C - +90°C	-150°C - +50°C
Besonderheiten:	- Autofokus 1,5m - unendlich - Aufgeklebte Farbfilter zur Farbbild Generierung	- Stereofähigkeit -12 verschiedene Filter auswählbar - Farbbild Generierung möglich

1.3. Optik

In diesem Unterkapitel werden die Grundlagen der Optik erklärt, wodurch man besser verstehen kann wieso die HRC vorfokussiert werden muss. Hierbei werden die Grundlagen der Kameraoptik und die Schärfentiefe erklärt. Die Optik ist für ein scharfes Bild sehr wichtig und wird daher bei der Disparitätenberechung wieder vorkommen.

1.3.1. Kameraoptik

Eine einfachere Abbildung einer Kamera ist in dem Bild Abbildung 11: Optischer Strahlengang (Fermu 09). Zu sehen sind ein Linsensystem und eine Bildebene. Mithilfe

einer Blende lässt sich der Lichteinfall steuern, je kleiner die Blendenanzahl ist, desto Schärfer wird auch das Bild. Ein wichtiger Punkt für ein scharfes Bild ist die Brennweite f´. Dies ist die Entfernung zwischen dem Linsenmittelpunkt und dem Brennpunkt des parallel zur Linse einfallenden Lichtes. Eine Formel, welche die Gegenstandsweite g, Bildweite b und die Brennweite f´ in Verbindung bringt, lässt sich mit dem Strahlensatz ermitteln.

$$\frac{1}{b}+\frac{1}{g}=\frac{1}{f'}$$

Formel 1: Strahlensatz

Die HRC wird eine feste Blende von 8 besitzen, sowie eine Brennweite von 100 mm haben.

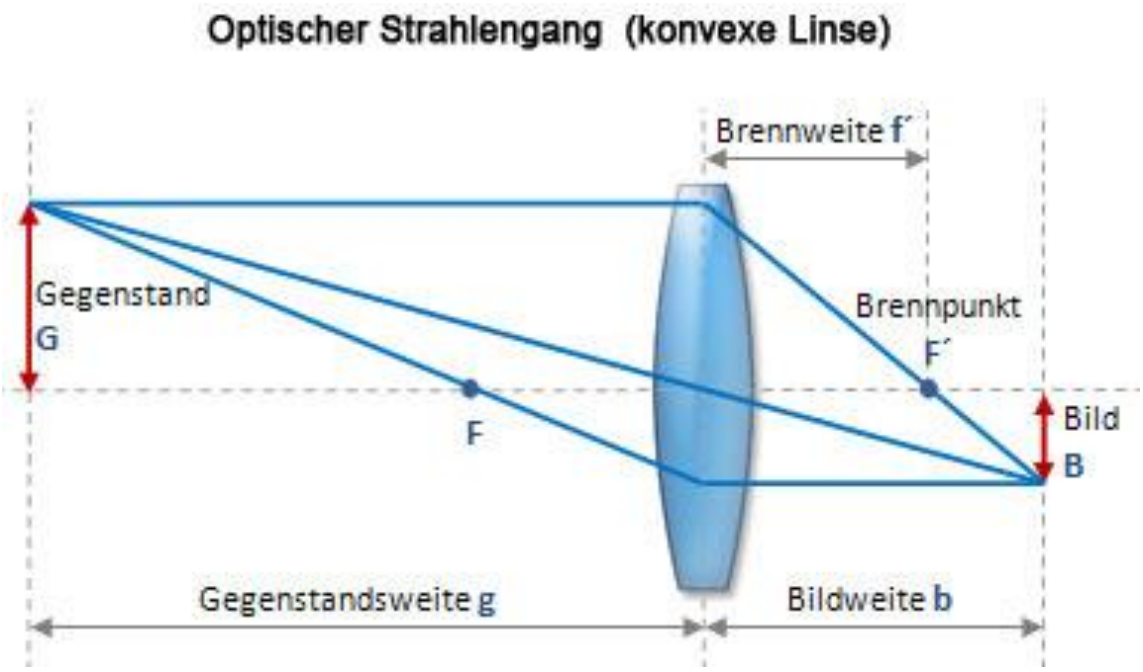

Abbildung 11: Optischer Strahlengang (Fermu 09)

1.3.2. Schärfentiefe

Dies ist der Bereich, in dem das Objekt auf dem Sensor scharf abgebildet wird. Der Schärfentiefebereich wird durch den Mindestdurchmesser des Unschärfekreises N beeinflusst - Abbildung 12: Schärfenpunkte (Grandi 09). Der fokussierte Punkt P wird auf der Bildebene scharf abgebildet. Befindet sich ein Objekt nicht auf der fokussierten Ebene, so wird dieses Objekt unscharf dargestellt. Je weiter ein Objekt von dieser Ebene entfernt liegt, desto größer wird der Unschärfekreis, wodurch dieses Objekt unschärfer abgebildet wird. Die Unschärfe wird erst dann erkennbar, wenn die Strahlen aus unterschiedlichen Entfernungen einen Unschärfekreis erzeugen, der größer ist als der Mindestdurchmesser N. Der Bereich zwischen dem Punkt Q (Nah Punkt) und Punkt R (Fern Punkt) ist der, in dem alle Objekt scharf dargestellt werden. Dabei entsprechen die Lichtgänge dieser Punkte den Unschärfekreisen mit dem Mindestdurchmesser. Der fokussierte Punkt P hat seinen Brennpunkt genau auf der Bildebene. Somit ist der Durchmesser des Unschärfekreises 0. Alle anderen Punkte, die außerhalb von R liegen, werden unscharf dargestellt.

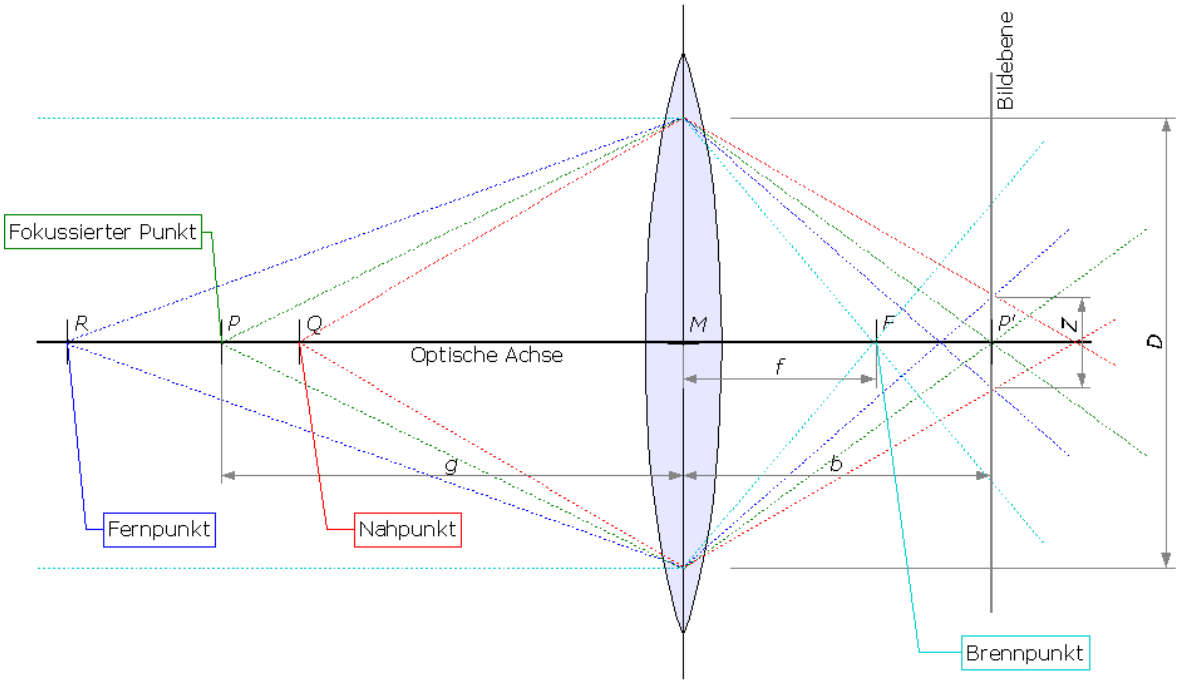

Abbildung 12: Schärfenpunkte (Grandi 09)

Der Abstand vom Mittelpunkt zum Nahpunkt lässt sich mittels Formeln berechnen. (Ulric 01,S. 61-66)

Nahpunkt:

$$\frac{1}{\frac{1.01}{g}+\frac{0.01}{b}}$$

Die Tabelle unten zeigt die Entfernung eines Objektes. Ferner ist daraus ersichtlich, ab wann dieses Objekt scharf eingestellt ist. Hierbei wird auch das Ende des Tiefenschärfebereichs angezeigt. Depth of Field (DOF) gibt an wie viel Tiefenschärfe insgesamt existiert, bis im Unendlichen keine Tiefenschärfe mehr wahrnehmbar ist.

Tabelle 2: Tiefenschärfe (DLR)

Object	Start	End	DOF
1,00	0,97	1,03	0,06
1,10	1,07	1,13	0,06
1,20	1,16	1,24	0,08
1,30	1,26	1,35	0,09
1,40	1,35	1,45	0,10
1,50	1,44	1,56	0,12
1,60	1,53	1,66	0,13
1,70	1,63	1,77	0,14
1,80	1,72	1,88	0,16
1,90	1,81	1,99	0,18
2,00	1,90	2,11	0,21
3,00	2,79	3,24	0,45
4,00	3,64	4,43	0,79
5,00	4,45	6,70	2,25
6,00	5,23	7,63	2,40
7,00	5,98	8,43	2,45
8,00	6,69	9,93	3,24
9,00	7,38	11,51	4,13
10,00	8,05	13,19	5,14
15,00	11,01	23,52	12,51
20,00	13,49	38,64	25,15
25,00	15,60	62,88	47,28
30,00	17,41	108,07	90,66
35,00	18,99	222,08	203,09
40,00	20,38	1063,83	1043,45
42,00	20,89	20000,00	19979,11

Im Bild unten ist zu sehen, dass ein Objekt in fünf Metern fotografiert werden soll. Der Nahpunkt liegt bei einer Entfernung von 4,43 Metern. Der Punkt, der am weitesten entfernt von der Kamera ist und noch scharf dargestellt wird, ist der Fernpunkt. Dieser liegt bei 5,74 Metern. Der Bereich zwischen diesen beiden Punkten ist der Schärfebereich oder auch Depth of Field.

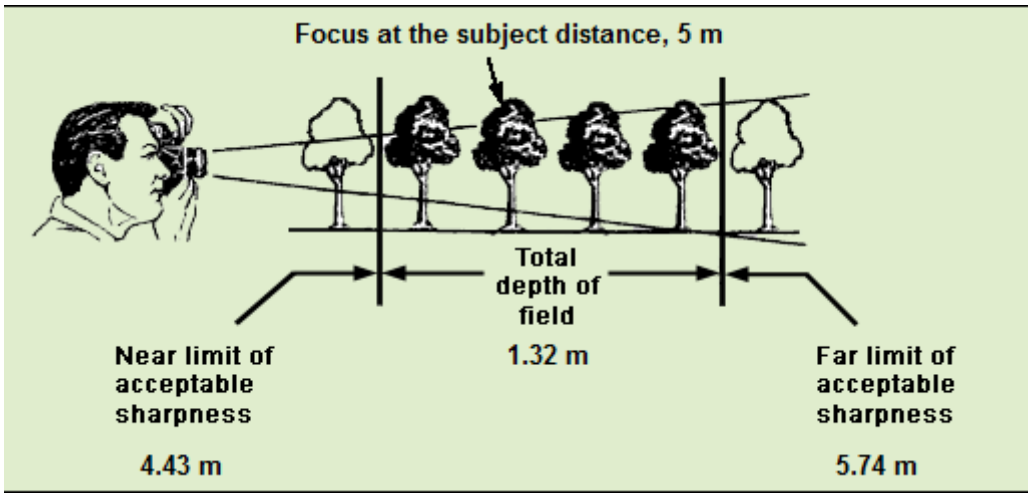

Abbildung 13: Tiefenschärfe Erklärung (Flemi 04)

2. Stereoanalyse

Die Stereoanalyse stellt Methoden und Algorithmen bereit, um in Stereobildpaare Entfernungen zu berechnen. Hierbei werden Informationen bezüglich der geometrischen Anordnung, der Kamera und der Fotosensoren z.B. der Pixelbreite genutzt. Mithilfe dieser Daten lassen sich Tiefeninformationen zu den beiden Bildern gewinnen. Dem Computerprogramm liegt der Stereonormalfall zu Grunde, da diese zahlreiche Vorteile mit sich bringt.

Im Stereonormalfall wird davon ausgegangen, dass sich die beiden Kameras auf einer Bildebene befinden und die selben Kameraparameter besitzen. Des Weiteren wird davon ausgegangen, dass die optischen Achsen parallel und senkrecht zur Basislinie verlaufen (Linie zwischen Ol und Or). Der Abstand der beiden Kameras wird als T angenommen und auch als Basis bezeichnet. Die Distanz zwischen der Bildebene und dem optischen Zentrum heißt Fokallänge. Sie ist bei beiden Kameras jeweils mit f angegeben. Die Abbildung unten zeigt den Stereonormalfall.

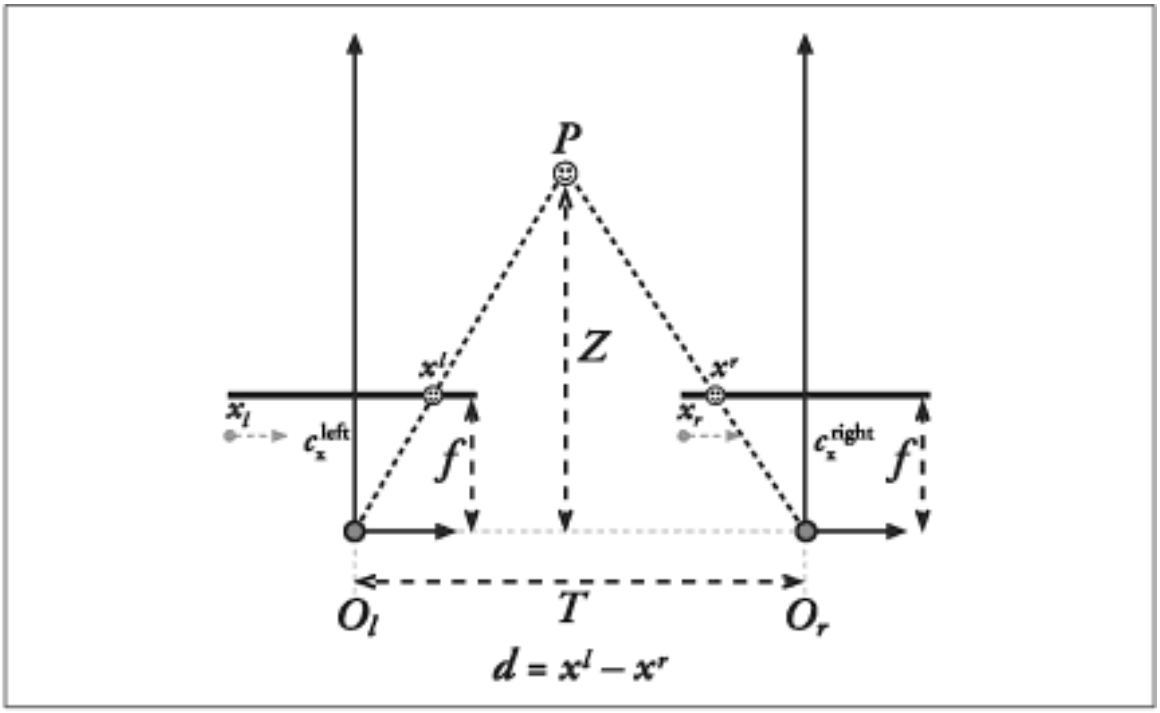

Abbildung 14: Steroanalyse (Gary 08, S. 416-417)

Der Vorteil dieser Anordnung liegt darin, dass es einfacher ist korrespondierende Punkte in den Stereobildern zu finden, da diese nur in einer Bildzeile liegen können. Dadurch wird die Suche nach den Korrespondenzen von zwei auf eine Ebene eingeschränkt. In dem Computerprogramm dient das linke Bild als Reverenz und es wird nur noch im rechten Bild nach dem entsprechenden Gegenstück gesucht.

Die nachfolgenden Unterkapitel werden die Theorie und die Praxis in dem Computerprogramm zeigen. So wird das Kapitel 2.1 genauer auf die auf die Möglichkeiten der Umsetzung des Programms eingehen. Das Kapitel 2.3 mit seinen Unterkapiteln zeigt die einzelnen Korrelationsverfahren, ihre Funktionsweise, die Umsetzung im Programm und ihre Performance. Darüber hinaus wird am Ende noch ein Vergleich, der den am besten geeigneten Algorithmus für den Mars Rover aufzeigt, geführt. Im Kapitel 2.4 wird auf die Disparitätenberechnung genauer eingegangen, d. h. wie aus den verschiedenen Informationen ein Tiefenwert gewonnen werden kann.

2.1. Testumgebung und Realisierung

Bei den Möglichkeiten der Realisierung gab es kaum Spielraum mit der Umsetzung des Programms. Verschiedene Möglichkeiten ergaben sich nur mit den Verfahren der Bildsuche. Es wurden bei den Algorithmen solche ausgewählt, die eine Komplexität von sehr niedrig bis sehr hoch besitzen.

Die Bilder wurden verwendet, weil sie eine mit dem Mars vergleichbare Landschaft darstellen. Sie stammen aus Spitzbergen, da diese Umgebung sich sehr gut für Tests des Mars Rover eignet.

Abbildung 15: Screenshot Program zeigt das Entwickelte Werkzeug.

1: Reverenz Bild	2: Suchbild
3: Algorithmus ändern	4: Matrixgröße ändern
5: Bilder laden/ Berechnen /Standartfall	6: Y-Offset
7: Statusmeldungen	8: Differenzbild

Abbildung 15: Screenshot Programm

Bei der Berechnung wird von dem einfachsten Fall ausgegangen. Kameraparameter werden von der WAC Test Umgebung, welche am DLR entwickelt wurde, genommen. Abbildung 16: WAC Testumgebung zeigt das Modell des Kamerasystems. Dies ist nicht der ExoMars Kamerakopf, sondern nur ein Testaufbau. Hingegen werden bei anderen Stereoanalysefällen Verzeichnungsobjektive benötigt. Auf diese kann jedoch bei der Testumgebung verzichtet werden. Dies resultiert daraus, dass sie den Rechenaufwand steigern und vernachlässigbare Verbesserungen bringen.

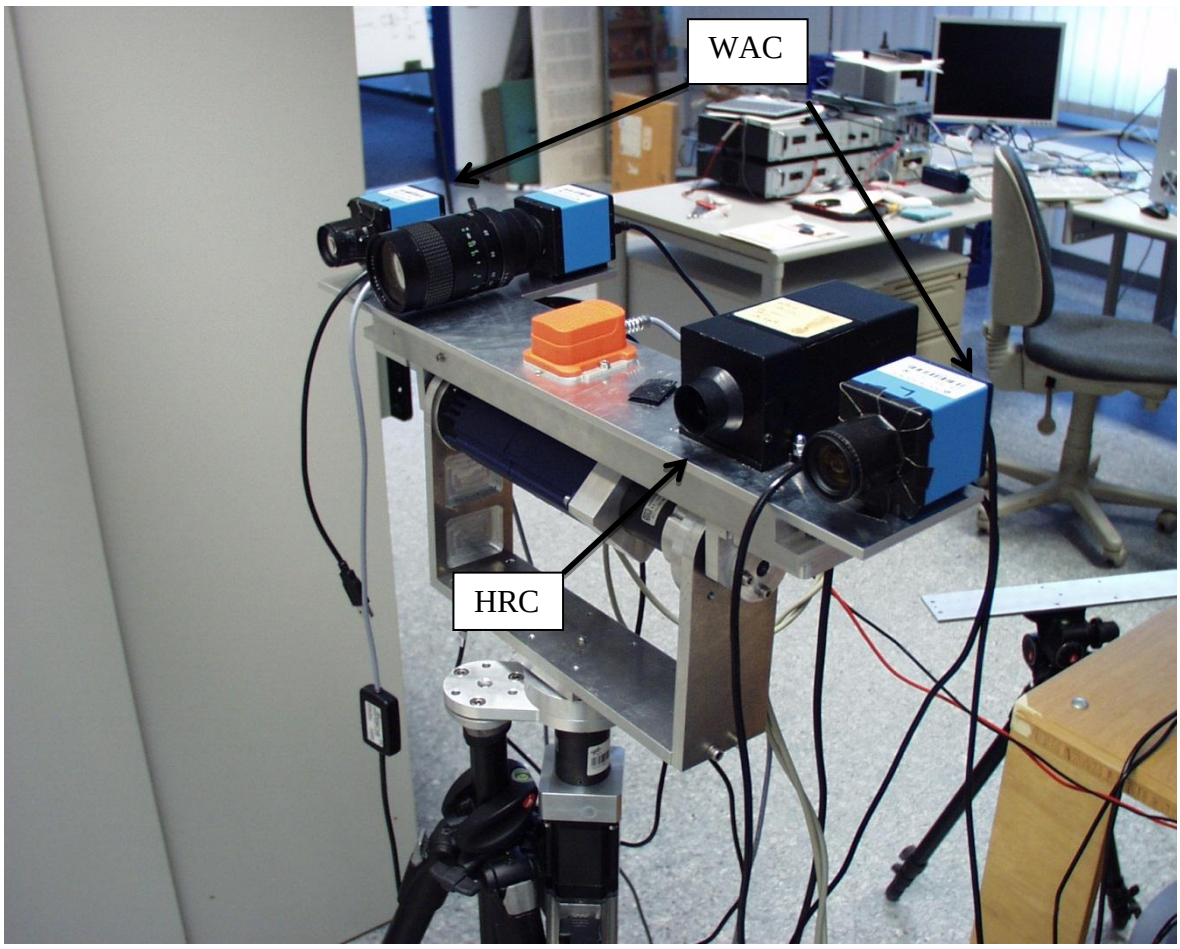

Abbildung 16: WAC Testumgebung (DLR)

2.2. Epipolargeometrie zur Vereinfachung der Stereoanalyse

Die Epipolargeometrie oder auch Kernstrahlgeometrie kommt aus der Geometrie und stellt die Beziehungen zwischen verschiedenen Fotos eines Objektes dar. Durch dieses mathematische Modell lassen sich Gemeinsamkeiten, also Bildpunkte finden, welche in beiden Bildern vorkommen.

Die Kernstrahlgeometrie wird vorrangig in der Photogrammmetrie eingesetzt, die durch die heutigen Computer vereinfacht durchgeführt werden kann. Eine Epipolarlinie ist die Gerade die entsteht, wenn sich die Bildebene und die Kernebene scheiden. Dabei scheiden sich alle Epipolarlinien im gemeinsamen Epipol. Dies ist der Punkt für eine Gerade, welche durch die Projektionszentren beider Bilder und der Bildebene läuft.

In dem entwickelten Programm sind die Epipolarlinien besser zu finden, da die Kameras auf einer Ebene liegen und nur 50 cm horizontal zueinander verschoben sind.

Auf dem Bild unten ist ein Beispiel für Epipolarlinien zu sehen. Das linke Bild ist die Reverenz und der rote Bereich stellt den zu suchenden Bereich dar. Das Bild in der Mitte ist das Suchbild und der rote Bereich stellt den gesuchten Abschnitt dar. Die Epipolarlinie geht durch beide Bilder im genau gleichen Bereich. Das rechte Bild zeigt den HRC-Suchausschnitt für den die Entfernung bestimmt werden soll.

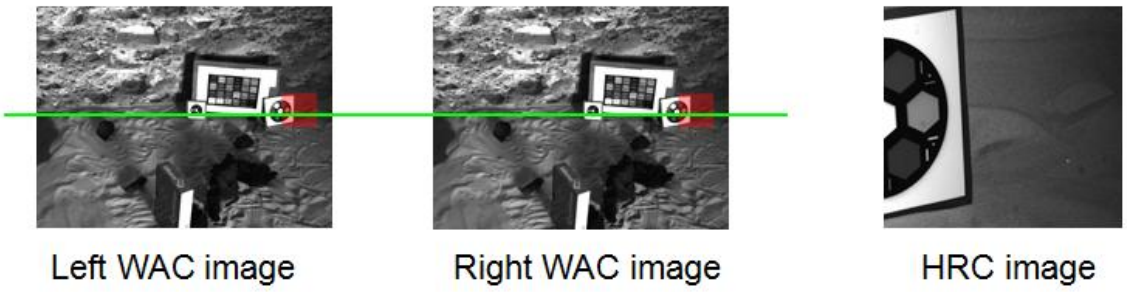

Abbildung 17: Epipolarlinien (DLR)

2.3. Mustererkennung

Korrelationsverfahren beschreiben wie eine oder mehrere Variablen zueinander stehen. So gibt der berechnete Wert den Grad der Ähnlichkeit an.

Hierbei können die Daten in Streudiagrammen angeordnet werden, um eine bessere Veranschaulichung zu erlangen. Es werden dabei, die zu vergleichenden Datensätze in je einem Diagramm eingetragen. So ist in Abbildung 18: Streudiagramm ein solches Diagramm zu sehen, das einen Datensatz zeigt.

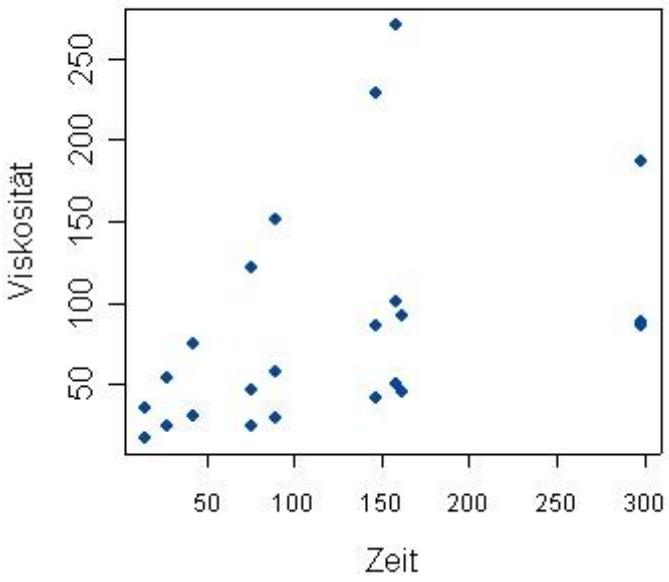

Abbildung 18: Streudiagramm (Bredn 03)

Korrelationsverfahren werden in vielen Bereichen angewandt.

- So werden hier verschiedene Wellenmuster miteinander vergleichen, um so in der Spracherkennung Wörter erkennen zu können. Hierbei bestimmt die Korrelation, den Grad der Ähnlichkeit zu bereits bestehenden Mustern.
- So sind in dem Programm drei verschiedene Verfahren zum Einsatz gekommen. Zwei von diesen Verfahren ähneln sich in ihrer Vorgehensweise und sind speziell auf die Bildverarbeitung abgestimmt. Das dritte Verfahren ist hierbei anders. Es findet auch in anderen Bereichen Verwendung. In dem Verlauf dieser Arbeit hat sich gezeigt, dass die speziell angepassten Verfahren mit größeren Matrizen zu Recht kommen.

Dies sind nur Beispiele aus technischen Gebieten. Weitere Bereiche existieren beispielsweise in der Kapitalwirtschaft.

Im weiteren Verlauf dieses Kapitels werden die ausgewählten Verfahren genauer erläutert. So wird im Kapitel 2.3.1. und den Unterkapiteln die Funktionsweise des Pyramidalgorithmus genauer erklärt werden. Ferner wird gezeigt, wie dieser Algorithmus in C# und im Programm selbst umgesetzt wurde. Im Kapitel 2.3.2. und folgenden wird der Korrelationsalgorithmus erklärt und wie beim Pyramidalgorithmus seine Umsetzung im

Programm verdeutlicht. Kapitel 2.3.3. und Unterkapitel befasst sich mit dem Template Matching, dem einfachsten der ausgewählten Algorithmen. Kapitel 2.3.4. befassen sich mit dem Vergleich der drei Algorithmen und wird deren Performance zeigen. In diesem Kapitel wird sich auch zeigen welcher der Algorithmen am besten geeignet ist, um auf einen Mars Rover installiert zu werden.

2.3.1. Pyramidalgorithmus

Der Pyramidalgorithmus beruht auf einem Quadtree. Dies ist eine spezielle Baumstruktur, die in der Informatik Anwendung findet. In dieser Struktur existieren eine Wurzel, Knoten und sogenannte Kinder. Das Wort Quadtree leitet sich von der Anzahl der Kinder, was vier sind und dem Wort Tree für die Baumstruktur ab.

In der Abbildung 19: Quadtree mit der Tiefe 4 und Abbildung 20: Quadtree im Quadrat ist eine solche Struktur zu sehen. Der schwarze Punkt oben stellt die Wurzel dar. Die blauen und roten Punkte stellen die Knoten dar. Die Kinder werden durch hellblaue Punkte symbolisiert. Das Ganze lässt sich auch auf einen Würfel übertragen. Hierbei stellt der schwarze Teil die Gesamtheit und die anderen jeweils die Verkleinerung dar.

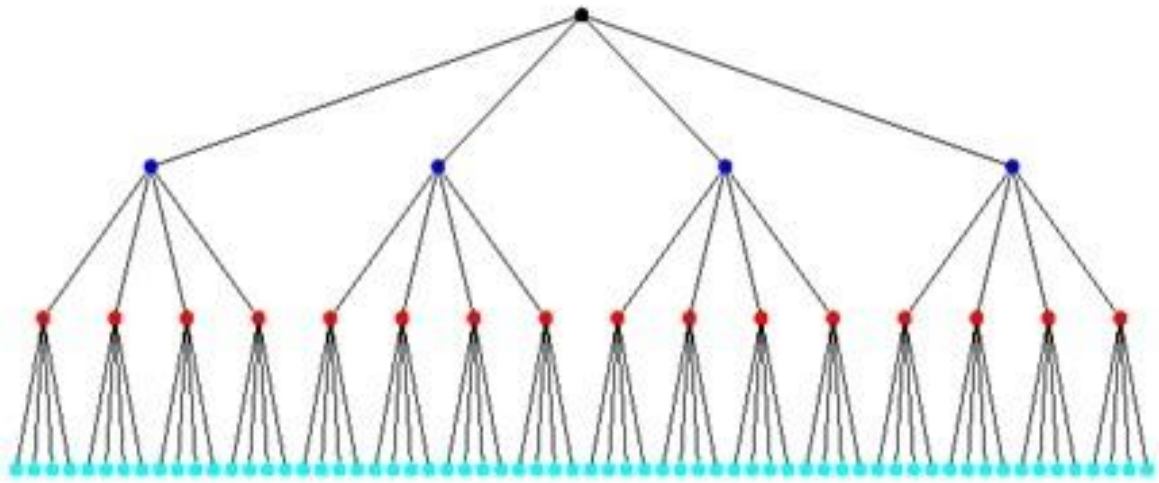

Abbildung 19: Quadtree mit der Tiefe 4 (Richa 01)

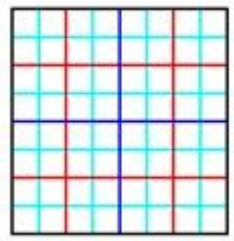

Abbildung 20: Quadtree im Quadrat (Richa 01)

Die Baumstruktur wird hauptsächlich zum Organisieren von zweidimensionalen Grafiken im Computerbereich eingesetzt. Die Wurzel ist hierbei die eine quadratische Fläche. Diese Fläche wird in vier gleichgroße Quadranten geteilt, bis man die erforderliche Auflösung erreicht hat. Mithilfe dieser Zerteilung kann die Grundfläche beliebig aufgelöst werden. Werden dreidimensionale Daten verwendet, wird meistens ein Octtree eingesetzt. (Brink 08)

Quadtrees haben folgende Einsatzgebiete:

- Repräsentation von Bildern
- Erkennung von Kollisionen bei 2-D Daten
- Zur Indizierung von Flächen
- Gittererzeugung

2.3.1.1. Funktionsweise

Die Funktionsweise im Programm ist umgekehrt zur normalen Baumstruktur. Das heißt, dass hier von den Blättern über die Knoten zur Wurzel gegangen wird. Daraus ergibt sich, dass von der Pixelebene ausgegangen wird, aus dieser Ebene werden dann 4 x 4 Matrizen gebildet. Dabei wird über die Pixel in der Matrix der Mittelwert bestimmt. Dies wird dann solange getan, bis nur noch eine große Ebene vorhanden ist.

Abbildung 21: Ebenen zeigt die Bilder, die bei der Berechnung der Pyramide von der Pixelebene bis zur Hauptebene entstehen. Sowohl bei der Reverenz als auch bei den zu vergleichenden Bildern wird dies einmal getätigt. Nach der Berechnung wird der Vergleich

ausgeführt. Dabei wird jede Ebene einzeln mit der Reverenz Block für Block verglichen und die Differenz über alle acht Ebenen aufaddiert. Oben links ist das Original des Suchbildes. Das zweite Bild daneben wurde schon einmal verarbeitet und ist in der Auflösung bereits gröber. Diese Aufteilung wird solange wiederholt bis nur noch eine Graustufe vorhanden ist.

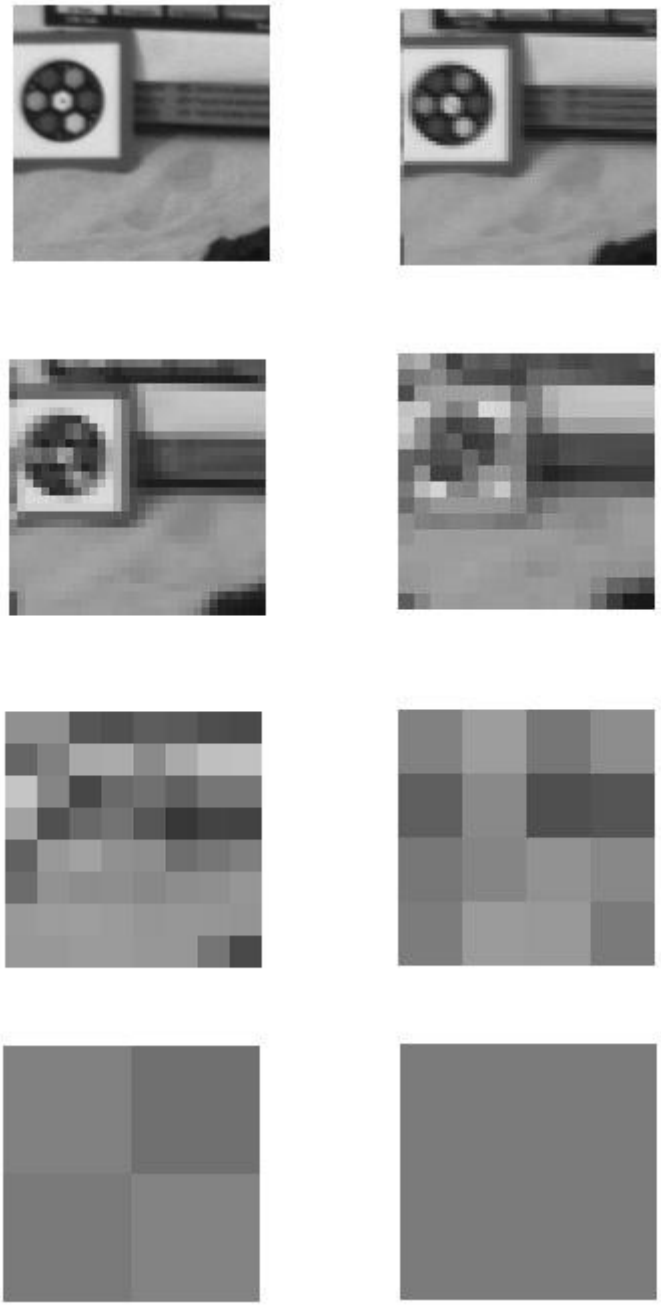

Abbildung 21: Ebenen (DLR)

Das linke Bild ist das Original, es besitzt 128*128 Pixel.
Das rechte Bild ist gröber. Es besteht aus 4096 4x4 Matrizen.

Das linke Bild besteht aus 1024 4x4 Matrizen. Die Auflösung wird also gröber.
Das rechte Bild hat 256 4x4 Matrizen mit Graustufen.

Das linke Bild hat 64 4x4 Matrizen, welche Grauwerte aus dem vorherigen Bild enthalten.
Das rechte Bild hat16 4x4 Matrizen mit den berechneten Werten.

Das linke Bild hat 4 4x4 Matrizen.
Das rechte Bild enthält nur noch eine Graustufe, welche durch die Berechnung entstanden ist.

2.3.1.2. Umsetzung im Programm

Die Umsetzung des Algorithmus wird nun Anhand des Quelltextes verdeutlicht und erläutert werden.

Der Quelltext in Anhang 1: Auflösungsebenen zeigt die sechs Ebenen der Pyramide, die siebente (Pixelebene) Ebene benötigt kein eigenes Element, da diese bei der Berechnung der sechsten Ebene übergeben wird. Der Mittelwert gibt diesen über alle sechs Ebenen berechneten Wert an und der Wert Groesse gibt an, wie viele Ebenen es geben soll. So wird standartmäßig eine Matrix von 128 x 128 verwendet. Hierbei wird dann ab Ebene 6 angefangen zu berechnen.

Der in Anhang 2: Pyramidalgorithmus gezeigte Quelltext dient der Berechnung der Pyramide. Hier wird standardmäßig bei Ebene 6 begonnen. Dieser wird eine 128 x 128 Matrix übergeben und es werden mithilfe dieser der Mittelwert und eine 64 x 64 Matrix berechnet. Dieses wird immer weiter durchgeführt, bis am Ende nur noch ein Wert über alle Matrizen vorhanden ist. Das Abbruchkriterium ist dann gegeben, wenn nur noch ein Wert vorhanden ist. Vergleichen lässt sich dieser Vorgang mit den Bildern aus Abbildung 21: Ebenen. Der Start mit der 128 x 128 Auswahl befindet sich oben links.

Die Variablen zahl1 und zahl2 dienen dazu die Position innerhalb der Matrix zu bestimmen, da immer eine 2 x 2 Matrix einen Mittelwert bilden soll. In den vier Schleifen wird einmal die übergebene Matrix durchgegangen und zum anderen die 2 x 2 Matrix. Der berechnete Mittelwert wird in eine neue Matrix gespeichert und der nächsten Berechnungsstufe übergeben, da diese sich alle ähneln, wird darauf nicht genauer eingegangen.

Der in Anhang 3: Vergleich Pyramidalgorithmus gezeigte Quelltext zeigt die Berechnung des Unterschiedes über zwei Pyramiden. Hierbei werden alle Ebenen durchgegangen. Von der Reverenz Matrix wird jeder einzelne Wert der Ebenen, mit der zu vergleichenden Ebene abgezogen. Der daraus resultierende Wert gibt an, wie gut die beiden Pyramiden übereinstimmen. Ein Wert von 0 wäre hierbei eine perfekte Übereinstimmung. Diese wird es jedoch nur bei zwei gleichen Bildern geben, da wenn mit zwei Kameras aufgenommen wird, immer ein leichter Unterschied entsteht. Sei es durch unterschiedliche Helligkeiten oder Farbunterschiede.

In einer weiteren vorgelagerten Methode wird nun der minimale Unterschied ausgewählt, da dieser meist dem gesuchtem entspricht.

2.3.2. Korrelationsalgorithmus

Die nachfolgenden Ausführungen beziehen sich auf Bildausschnitte, die miteinander verglichen werden. Hier wird die Korrelation oder auch die Kreuzkorrelation hergenommen, um ein Maß für die Ähnlichkeit zu finden. Als eine Korrelation bezeichnet man einen Algorithmus, der eine automatische Zuordnung von homologen Bildausschnitten zur Punktübertragung oder für einen Vergleich von Bildmustern im Bildraum bezeichnet. Selbiges wird auch Matching genannt.

Dieses ist der allgemeinste Algorithmus von den drei Verwendeten, er lässt sich auf beliebige Matrizen anwenden, um so den Grad der Ähnlichkeit zu finden.

2.3.2.1. Funktionsweise

Die Korrelation berechnet sich aus der Kovarianz Cov (X, Y) und dem geometrischen Mittel zu den Varianzen Var (X) und Var (Y) (Prusc 05, S. 34-37):

$$p_{(x,y)} = \frac{Cov(X,Y)}{\sqrt{Var(x) * Var(Y)}}$$

Die Kovarianz ist eine Maßzahl für den Zusammenhang zweier Variablen. Die Varianz gibt ein Maß für die Abweichung zu einem Erwartungswert an.

Mithilfe der Division der Streuungen, wird die Kovarianz auf ±1 normiert. Damit kann der Korrelationskoeffizient folgende Werte annehmen:

+1= Übereinstimmung beider Proben

0 = Keine Übereinstimmung beider Proben

-1= Vorzeichenwechsel

Die Varianzen und Kovarianzen werden durch die Mittelwerte $\bar{x}$,$\bar{y}$ und dem Aufsummieren der Quadrate bzw. der Kreuzprodukte berechnet. Hieraus folgt der Korrelationskoeffizient:

$$r_{x,y} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}((x_i - \bar{x})^2 \sum_{i=i}^{n}(y_i - \bar{y})^2}}$$

Dadurch, dass im Programm eine Reverenz existiert, muss der Mittelwert über x nicht mehr jedes Mal berechnet werden, wodurch die Laufzeit verbessert werden kann.

2.3.2.2. Umsetzung im Programm

Die Umsetzung des Algorithmus innerhalb des Programms wird nun Anhand des Quelltextes verdeutlicht und erläutert werden.

Der Code in Anhang 4: Korrelationsalgorithmus Reverenz dient dazu die Sum1 zu berechnen, damit diese nicht immer wieder berechnet wird. Der Quelltext wird einmal nach der Auswahl des Reverenzbereichs ausgeführt.

Dieser Teil des Codes im Anhang 5:Korrelationsalgorithmus Berechnung dient dazu die eigentlichen Berechnungen immer wieder durchzuführen. Am Ende der Berechnungen werden alle Variablen außer Sum1 wieder auf 0 gesetzt, damit keine Fehlberechnungen zustande kommen.

2.3.3. Template Matching

Dieser Algorithmus ist der einfachste von den drei ausgewählten. Es geht hierbei um den Vergleich zweier Bilder und ist daher so einfach, da Pixel für Pixel verglichen wird, ohne vorherige Berechnungen. Probleme bei der Erkennung von korrekten Bildbereichen können sein:

- Zu geringer Kontrast im ausgewählten Template
- Rotation
- Helligkeitsänderungen
- Minimale Verschiebungen

Man kann das Template Matching in zwei Kategorien einteilen. Zum einen das funktionsbasierende Matching. Dieses dient dazu, leicht erkennbare Bereiche zu finden, wie Ränder oder Ecken. Andererseits gibt es das musterbasierende Matching. Dieses wird dazu genutzt, ein bestimmtes Bildmuster in einem anderen wiederzufinden. (Brune 09) In dieser Arbeit wird das musterbasierende Matching verwendet, da der Suchbereich das ganze Bild umfassen soll und nicht nur die leicht erkennbaren Bereiche des Bildes. Es kann hierbei in Kauf genommen werden, dass das Muster nicht 100 prozentig wiedergefunden wird, da es nur zu einer Schätzung der Entfernung kommen soll. Sollten beim Erkennen zu große Diskrepanzen entstehen, so muss ein neues Muster gewählt werden, um mit diesem dann eine neue Suche zu starten.

Das Template Matching hat viele Einsatzgebiete innerhalb der Informatik.

- Gesichtserkennung
- Datenverarbeitung beim DataMining
- Optische Qualitätskontrolle
- Bildverarbeitung

Dieses lässt sich durch die Vorteile erklären:

- Hohe Geschwindigkeit
- Geringer Speicherverbrauch
- Einfach zu implementieren

Ein Nachteil zu diesem Algorithmus ist, dass Fehlfunde entstehen, wenn sich die Daten zu stark ähneln.

2.3.3.1. Funktionsweise

Die Funktionsweise dieses Algorithmus ist sehr einfach und lässt sich auf eine Formel zurückführen:

$$Diff = \sum_{x=0}^{Reihen} \sum_{y=0}^{Zeilen} Bild1[x,y] - Bild2[x,y]$$

Zu der Formel ist noch zu sagen, dass mit Graustufenbildern gerechnet wird, das heißt, dass nur ein Farbkanal des RGB Farbraumes gebraucht wird. Siehe Abbildung 22: Stein und Auswahl(DLR)

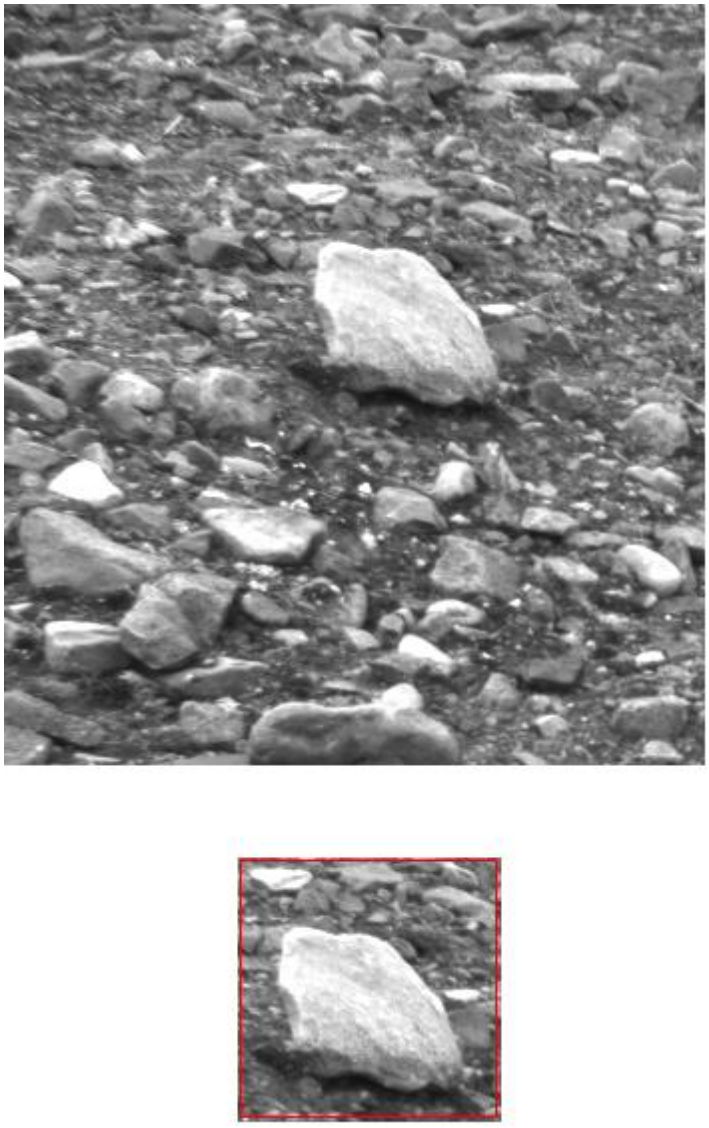

Abbildung 22: Stein und Auswahl(DLR)

Wendet man nun die Formel auf das Bild und die Auswahl an, so wird sich eine Differenz von 0 ergeben, da beide aus demselben Bild stammen. Da es sich aber um eine Stereoanalyse handelt, wird Abbildung 23: Horizontal verschobenes Bild(DLR) verwendet. Dabei wird eine minimale Differenz von 543235 berechnet.

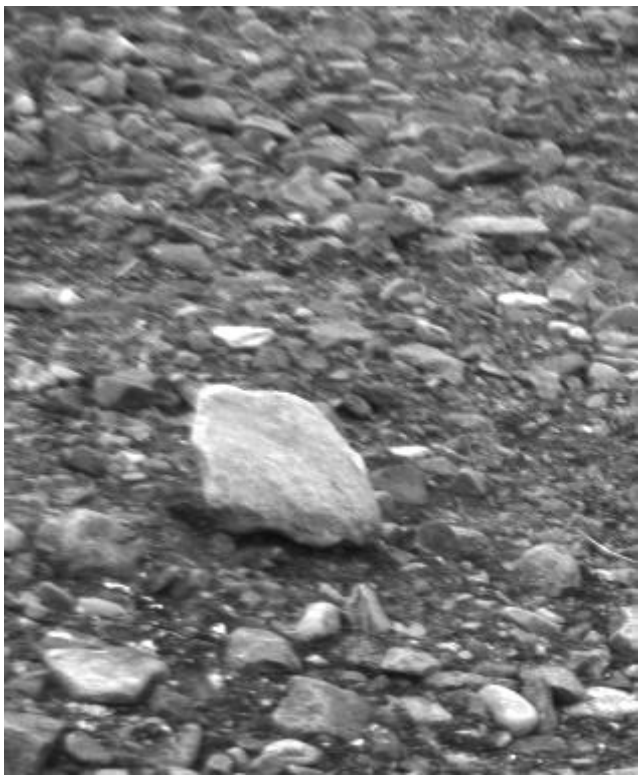

Abbildung 23: Horizontal verschobenes Bild(DLR)

2.3.3.2. Umsetzung im Programm

Die Einfachheit des Algorithmus kann man schon daran erkennen, dass dies alles ist was man benötigt siehe Anhang 6: Template Matching. Der Methode wird die Reverenz und der Suchraum als Matrix übergeben. Die Variable Groesse dient dazu, um festzustellen wie groß die Matrix ist. Hierbei muss es sich jedoch immer um eine quadratische Matrix handeln. Mit den beiden Schleifen werden dann auch die Reverenz und der Suchraum durchgegangen. Es wird die Differenz jedes Pixels von der Reverenz mit den Pixeln des Suchraums gebildet.

In einer weiteren vorgelagerten Methode wird dann die niedrigste Differenz ausgewählt, da diese meist dem gesuchtem entspricht.

2.3.4. Vergleich der Verfahren

Bei dem Vergleich der Verfahren werden die Abbildung 24: Linkes Bild WAC (DLR) und Abbildung 25:Rechtes Bild WAC eingesetzt, funktioniert aber auch mit Abbildung 26: Testumgebung WAC Links (DLR) und Abbildung 27: Testumgebung WAC Rechts (DLR) In beiden Bildern ist eine Fläche zu sehen, in der viele Steine zu sehen sind. Für den Vergleich wird der große Stein auf der oberen linken Hälfte verwendet, weil:

- Hoher Kontrast auf derselben Bildzeile
- Gute Unterscheidbarkeit zu den anderen Steinen
- Auch für kleinere Suchmatrizen erkennbar

Abbildung 24: Linkes Bild WAC (DLR)

Abbildung 25:Rechtes Bild WAC (DLR)

In den nachfolgenden Ausführungen wird jeder Algorithmus erst alleine für sich auf folgende Merkmale hin untersucht:

- Geschwindigkeit
- Anzahl der Schleifenaufrufe
- Speicherverbrauch

Auf diese drei Merkmale hin, wird jede der Suchmatrizen untersucht, ob ein hinreichendes Ergebnis gefunden werden kann. Im Anschluss daran, werden Vor- und Nachteile aller Algorithmen aufgezeigt und es wird eine Empfehlung ausgesprochen, welcher Algorithmus sich mit welcher Suchmatrix für den Mars Rover eignet.

Abbildung 26: Testumgebung WAC Links (DLR)

Abbildung 27: Testumgebung WAC Rechts (DLR)

In dem nachfolgenden Diagramm wird die Anzahl der Schleifendurchläufe des Pyramidalgorithmus gezeigt. Zu sehen ist das die 128 x 128 Matrix am meisten Zeit und das die 64 x 64 Matrix deutlich weniger Zeit benötigt. Die anderen zwei Matrizen benötigen beide unter 5 Sekunden.

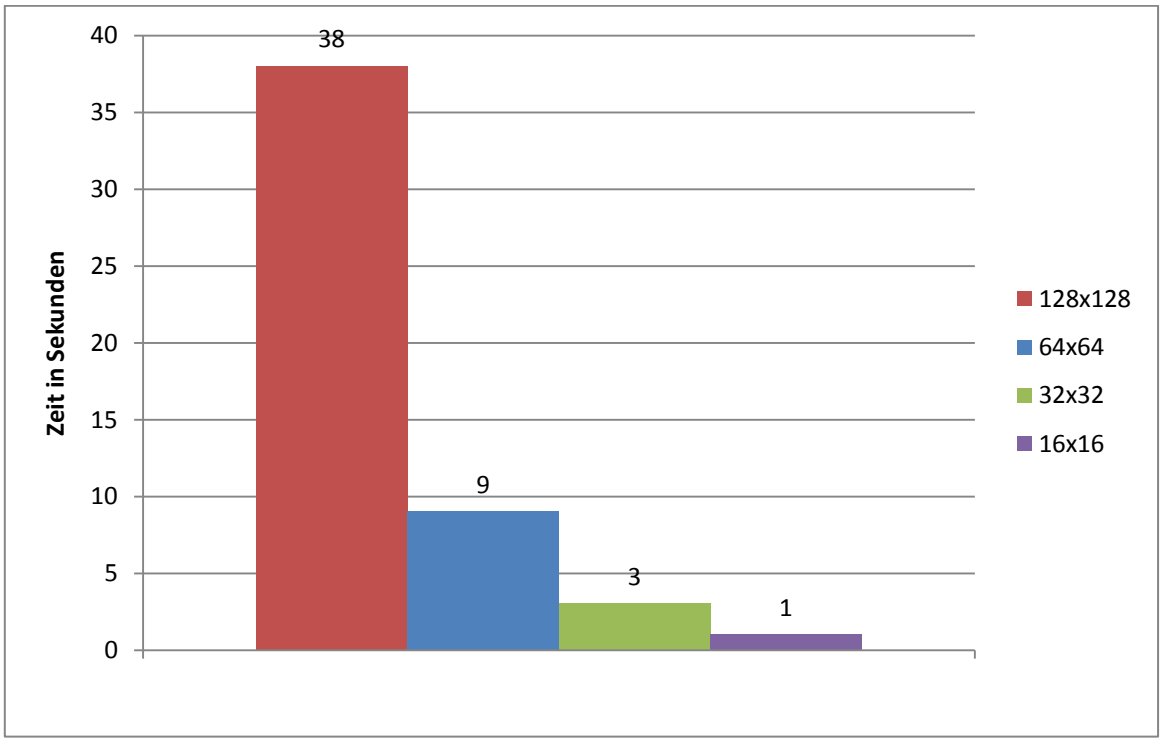

Abbildung 28: Pyramidalgorithmus Geschwindigkeit

In Abbildung 29: Pyramidalgorithmus Schleifenaufrufe kann man erkennen, dass auch hier die 128 x 128 Matrix den größten Wert besitz. Des Weiteren sinken auch hier die anderen Matrizen stark im Vergleich zur ersten.

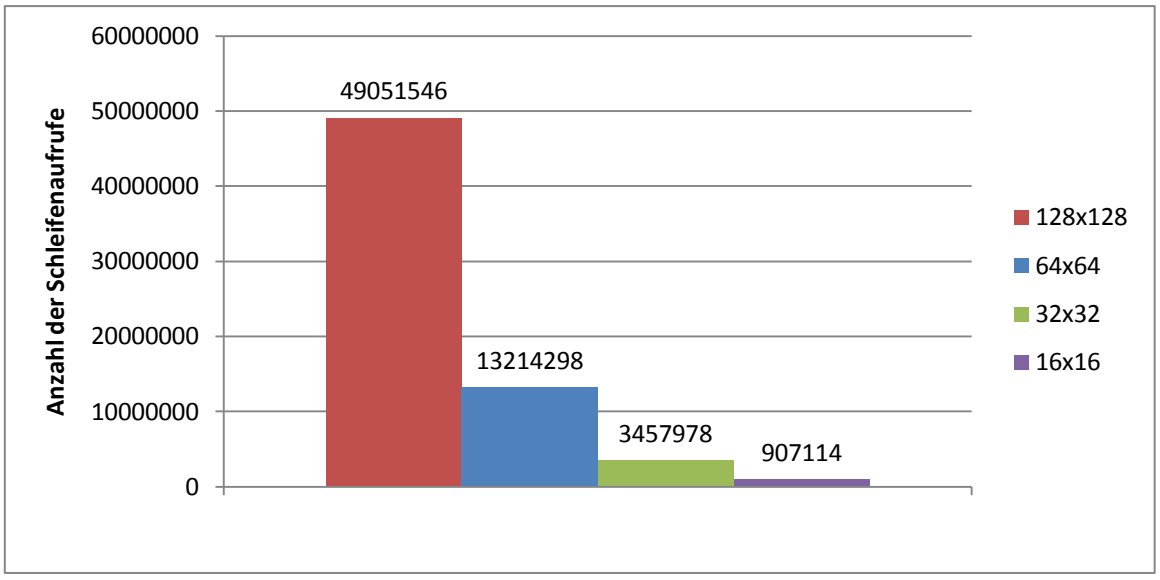

Abbildung 29: Pyramidalgorithmus Schleifenaufrufe

In Abbildung 30: Pyramidalgorithmus Speicherverbrauch in KByte ist zu sehen, dass alle vier Matrizen einen fast gleichen Speicherverbrauch besitzen. Dieser schwankte auch je nachdem, wie oft man das Programm neu gestartet hatte. Zu sehen sind die Mittelwerte des Speicherverbrauchs jedes Versuchs.

Der Algorithmus fand die anderen Bildausschnitte ohne Probleme, wobei ab der 32 x 32 Matrix darauf zu achten ist, dass es ein Ausschnitt mit sehr hohem Kontrast zu den anderen ist.

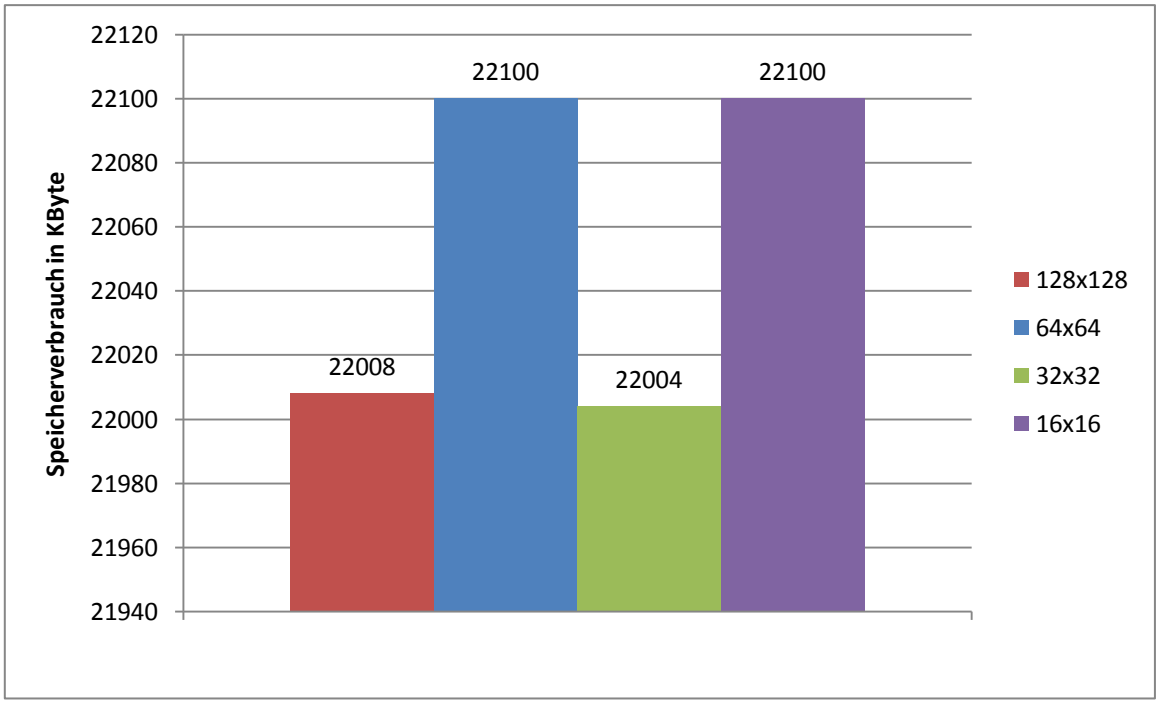

Abbildung 30: Pyramidalgorithmus Speicherverbrauch in KByte

In Abbildung 31: Korrelationsalgorithmus Geschwindigkeit ist zu sehen, dass die 128 x 128 Matrix am meisten Zeit benötigt um ein Ergebnis zu liefern. Die anderen Matrixgrößen sind wesentlich schneller, wobei die 64 x 64 Matrix nur 9 Sekunden benötigt.

Im Laufe der Untersuchung des Algorithmus kam heraus, dass dieses Verfahren zu stark kompensiert. Er findet kaum den richtigen Ausschnitte im anderen Bild und liefert immer nur Ausschnitte die so ähnlich aussehen. Aufgrund dieser Merkmale wurde bei der Untersuchung des Verfahrens zweimal dasselbe Bild verwendet. Dies bedeutet, dass der Algorithmus nur den Ausschnitt wiederfinden musste, was er auch immer erfüllt hat.

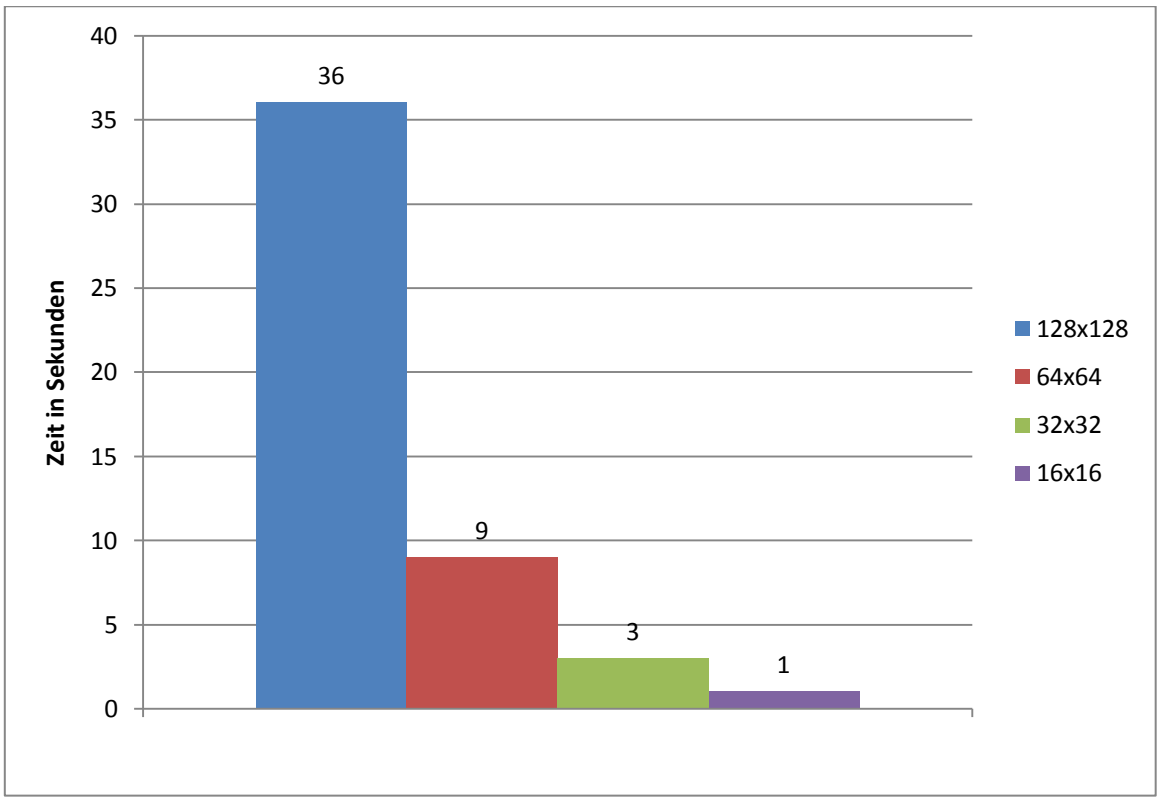

Abbildung 31: Korrelationsalgorithmus Geschwindigkeit

In der Abbildung 32: Korrelationsalgorithmus Schleifenaufrufe ist zu erkennen, dass wie in Abbildung 31 die größte Matrix am meisten Aufrufe bzw. Zeit benötigt.

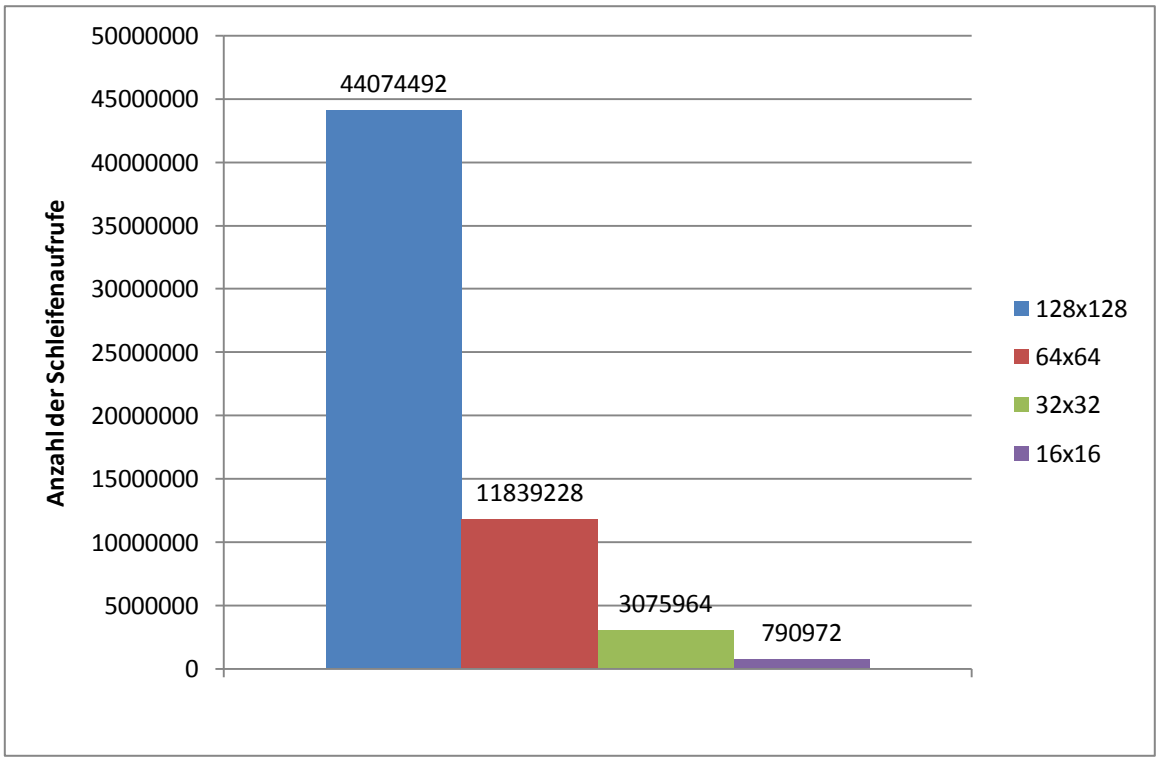

Abbildung 32: Korrelationsalgorithmus Schleifenaufrufe

In Abbildung 33: Korrelationsalgorithmus Speicherverbrauch ist zu sehen, dass die vier Matrizen wieder annähernden Speicherverbrauch haben. Die kleinen Schwankungen lassen sich durch unterschiedliche Bedingungen beim Start erklären, sodass man sagen kann, dass alle ungefähr gleichviel Speicher benötigen.

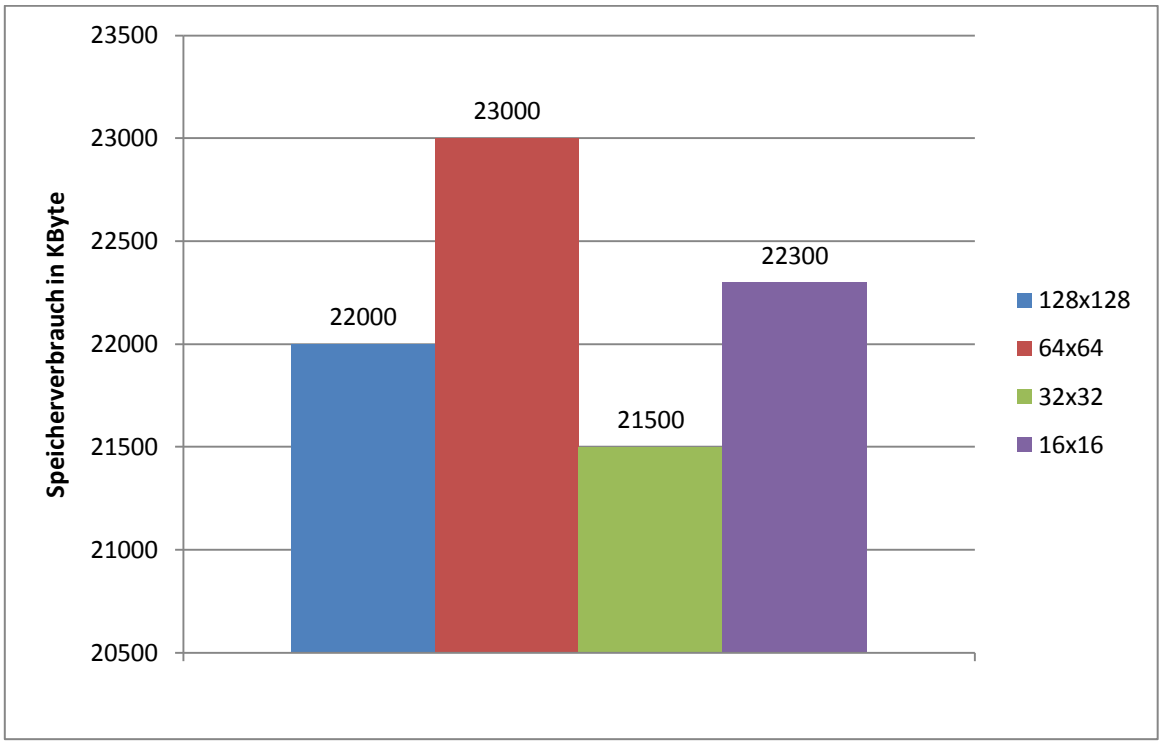

Abbildung 33: Korrelationsalgorithmus Speicherverbrauch in KByte

In Abbildung 34: Template Matching Geschwindigkeit ist zu sehen, dass auch hier die 128 x 128 Matrix am meisten Zeit benötigt, aber im Vergleich zu den anderen beiden Algorithmen verbraucht dieser am meisten Zeit. Auch bei der kleineren 64 x 64 Matrix wird im Vergleich zu den anderen mehr Zeit benötigt. Erst ab der 32 x 32 Matrix gleicht sich die Zeit an.

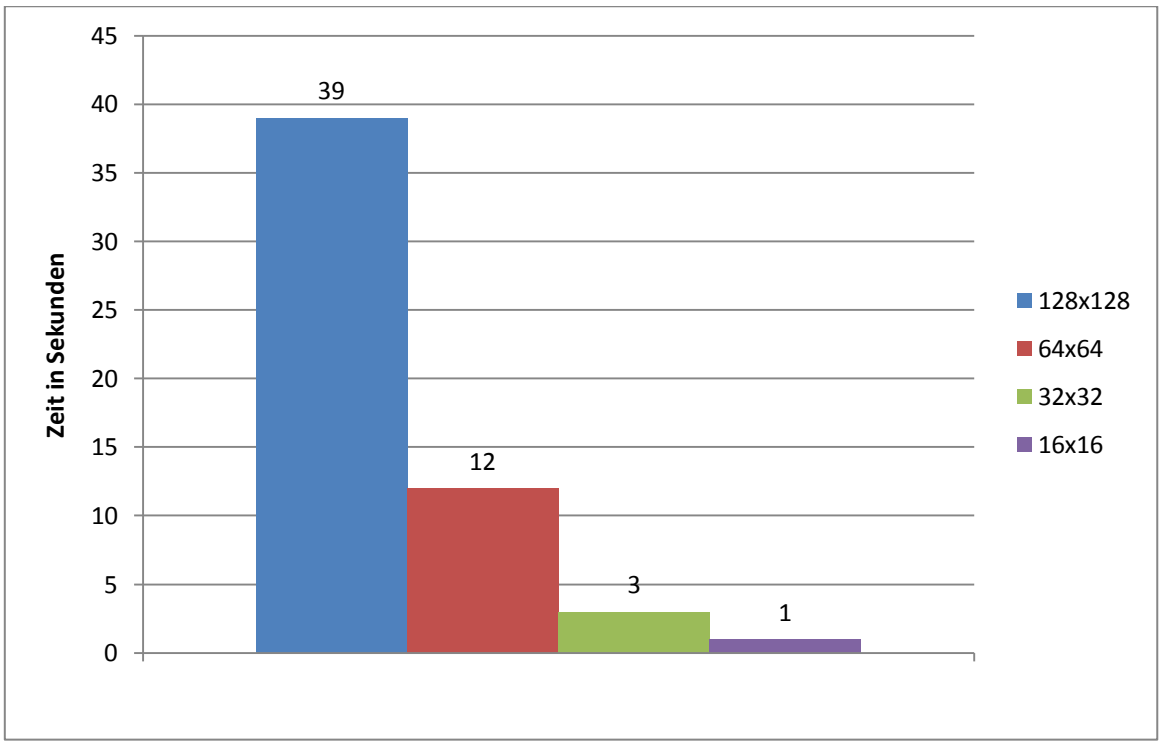

Abbildung 34: Template Matching Geschwindigkeit

In Abbildung 35: Template Matching Schleifenaufrufe ist zu erkennen, dass alle Matrizen weniger Aufrufe benötigen als die anderen. Aber auch hier existiert ein großer Abstand zwischen der 128 x 128 Matrix und der 64 x 64 Matrix.

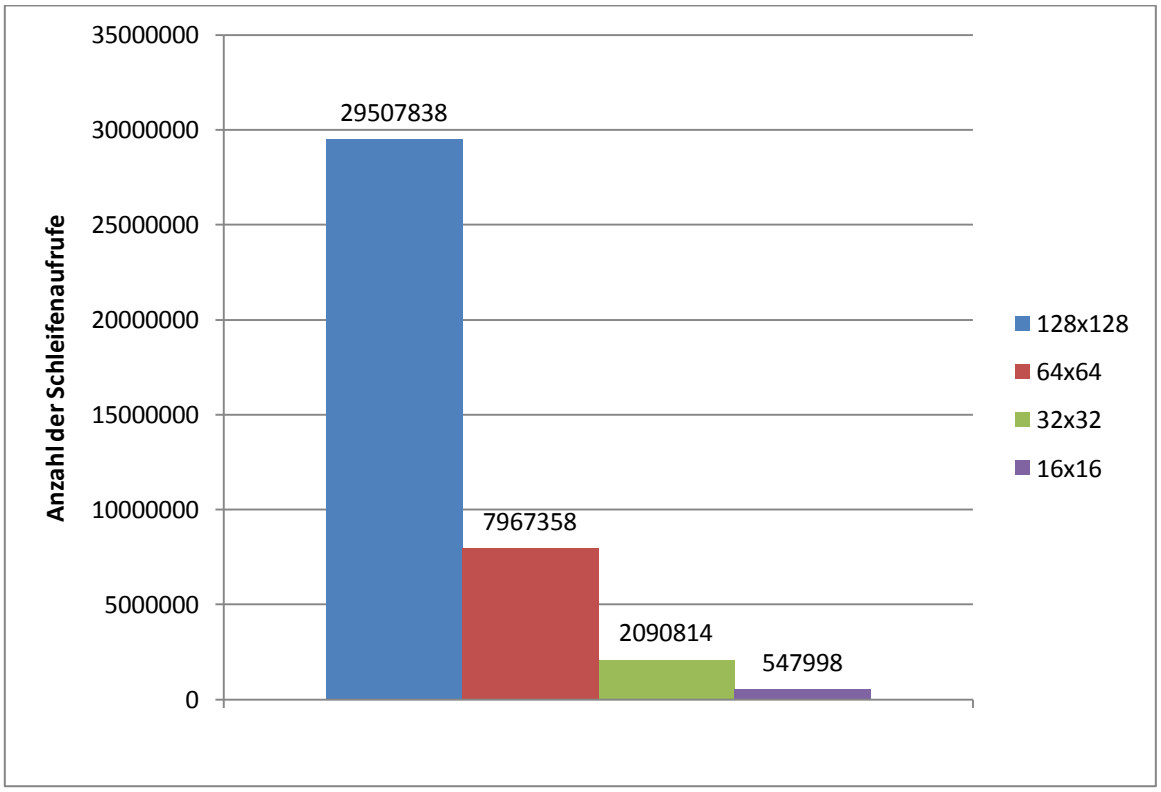

Abbildung 35: Template Matching Schleifenaufrufe

In Abbildung 36: Template Matching Speicherverbrauch kann man sehen, dass sich auch hier der Speicherverbrauch angleicht. Im Vergleich zu den anderen benötigen alle ungefähr denselben Speicher.

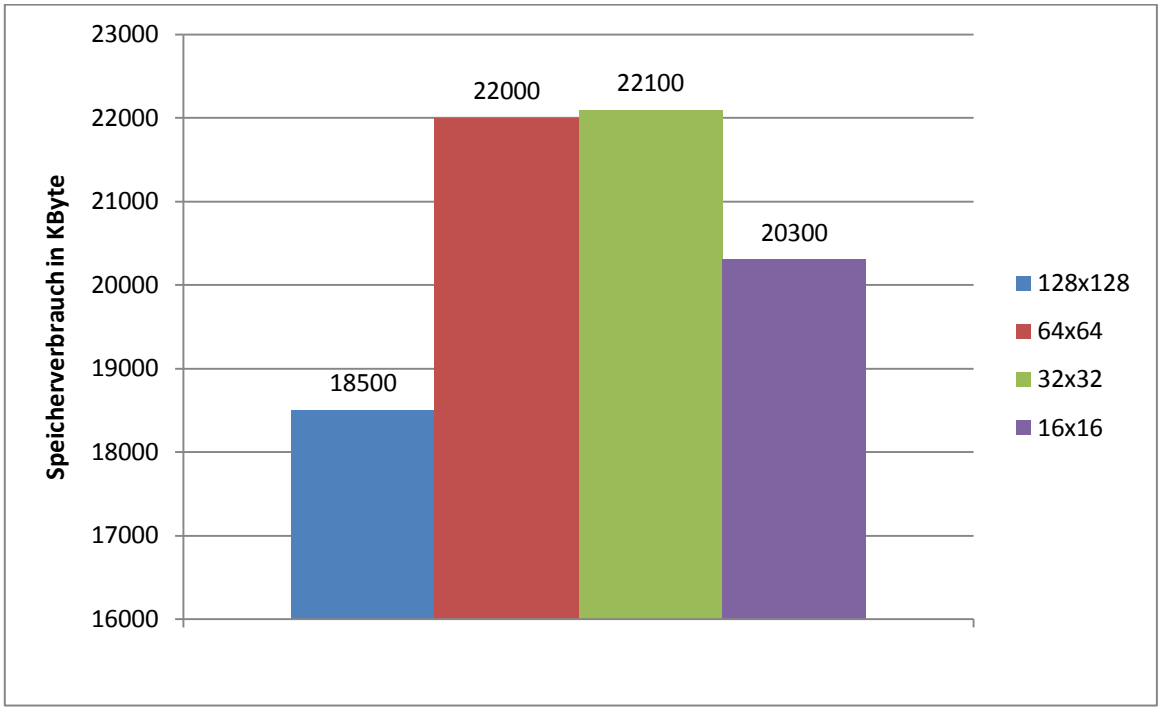

Abbildung 36: Template Matching Speicherverbrauch in KByte

Die Empfehlung für den Mars Rover fällt aufgrund der Vergleiche, auf den Pyramidalgorithmus mit der 64 x 64 Matrix. Dieses lässt sich dadurch erklären, dass er am wenigsten Aufrufe besitzt und immer noch ein gutes Ergebnis bei der Mustererkennung liefert. Der Korrelationsalgorithmus fällt aus der Bewertung, da der Prozessor keine Gleitkomma Einheit besitzt und solche Berechnungen nur mit größerem Aufwand durchführen kann.

2.4. Disparitäten- und Entfernungsberechnung

Diese lässt sich sehr einfach mithilfe einer Formel erklären:

$$d = x^1 - x^2$$

Hierbei ist x^1 die Koordinate in der Reverenz und x^2 die in dem Suchbild. Vollzieht man diese Rechnung erhält man die Disparität mit der die Entfernung berechnen werden kann.

Die Berechnung der Entfernung lässt sich mit dieser Formel realisieren.

$$\frac{T-(x^l-x^r)}{Z-f}=\frac{T}{Z} \quad \Rightarrow \quad Z=\frac{fT}{x^l-x^r}$$

Abbildung 37: Formel Entfernungsberechnung (DLR)

Hierbei ist T = 49,9 cm und f= 8,02 mm, diese müssen dann zusammen mit den X-Koordinaten einsetzen um die Entfernung zu bekommen. Die Pixelbreite beträgt 4,65 µm und wird für die Disparität benötigt.

Tabelle 3: Disparitäten im Vergleich zeigt die Disparitäten im Vergleich. Alle gingen vom gleichen Ursprung aus. Zu sehen ist, dass keiner der Algorithmen die gleiche Disparität findet wie ein anderer. Bei dem Pyramidalgorithmus gibt es eine starke Abweichung bei der 32 x 32 Matrix. Dies lässt sich durch Kontrastunterschiede erklären.

Tabelle 3: Disparitäten im Vergleich

	Pyramidalg.	Korrelationsalg.	Template Matching
128x128	138	147	129
64x64	130	267	123
32x32	328	130	117
16x16	111	98	109

Setzt man nun alle Werte in die Formel ein, so kommt folgende Berechnung zustande. Es wurde alles auf Meter umgerechnet.

$$\frac{0{,}499m * 0{,}00802m}{130 * 4{,}65\mu m} = 6{,}62m$$

Das heißt, dass man auf eine Entfernung von ca. 6,6 Meter kommt, bei dem Pyramidalgorithmus mit der 64 x 64.

Das Bild unten zeigt die Disparitäten des Pyramidalgorithmus. Zu sehen ist, dass sich die ersten beiden und die letzte Matrix nicht sehr stark voneinander abweichen. Die 32 x 32 Matrix weicht stark von der Normalität der anderen drei ab. Erklären lässt sich dies dadurch, dass die vorhandene Datenmenge immer kleiner wird. Die 16 x 16 Matrix hat wieder ein akzeptables Ergebnis gefunden. Das heißt, dass ab einer Matrixgröße von 32x32 abwärts von einer Verschlechterung der Werte auszugehen ist.

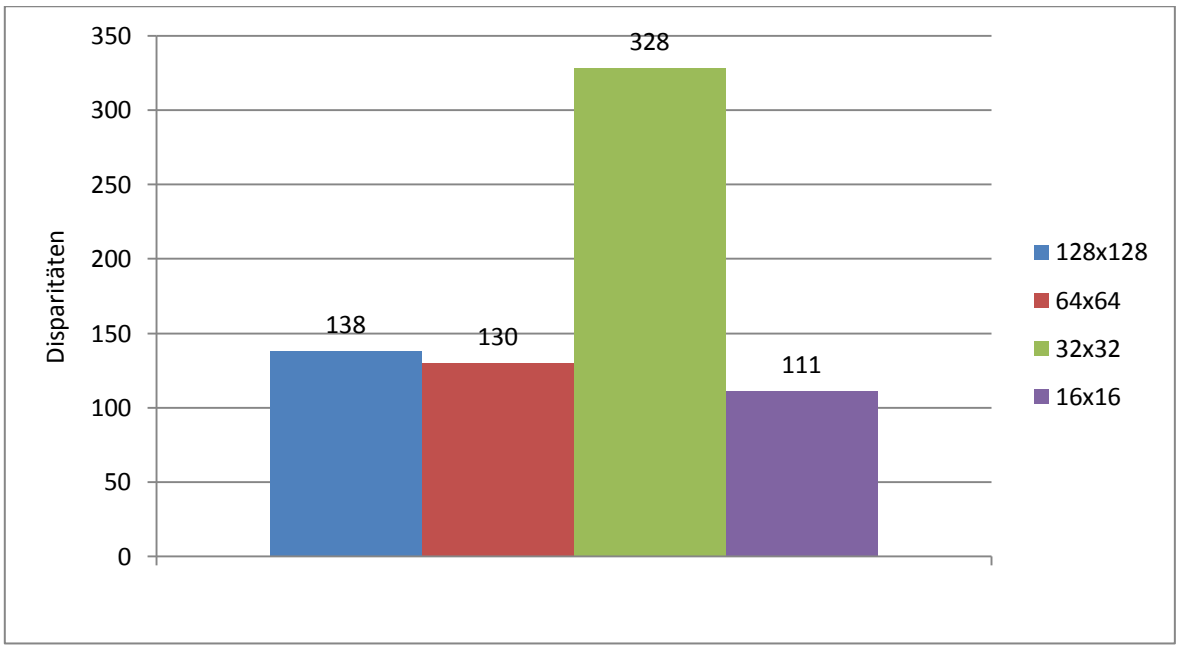

Abbildung 38: Disparitäten Pyramidalgorithmus

Die Abbildung 39: Disparitäten Korrelationsalgorithmus zeigt die Disparitäten des Korrelationsalgorithmus. Zu erkennen ist, dass die 32 x 32 Matrix das beste Ergebnis liefert, wenn der Pyramidalgorithmus als Reverenz genutzt wird. Auch hier existiert ein Ausreißer, der eine starke Abweichung vom Normalmaß des Algorithmus besitzt. Dies ist die 64 x 64 Matrix. Ausgehend vom Pyramidalgorithmus, kann man diesen Algorithmus nicht verwenden, da seine Schwankungen und Ergebnisse zu schlecht sind.

Abbildung 40: Disparitäten Template Matching zeigt die Disparitäten des Template Matchings. Es zeigt sich, dass dieser Algorithmus wenige Schwankungen in den berechneten Werten besitzt. So gibt es maximal eine Abweichung von 20 Pixeln.

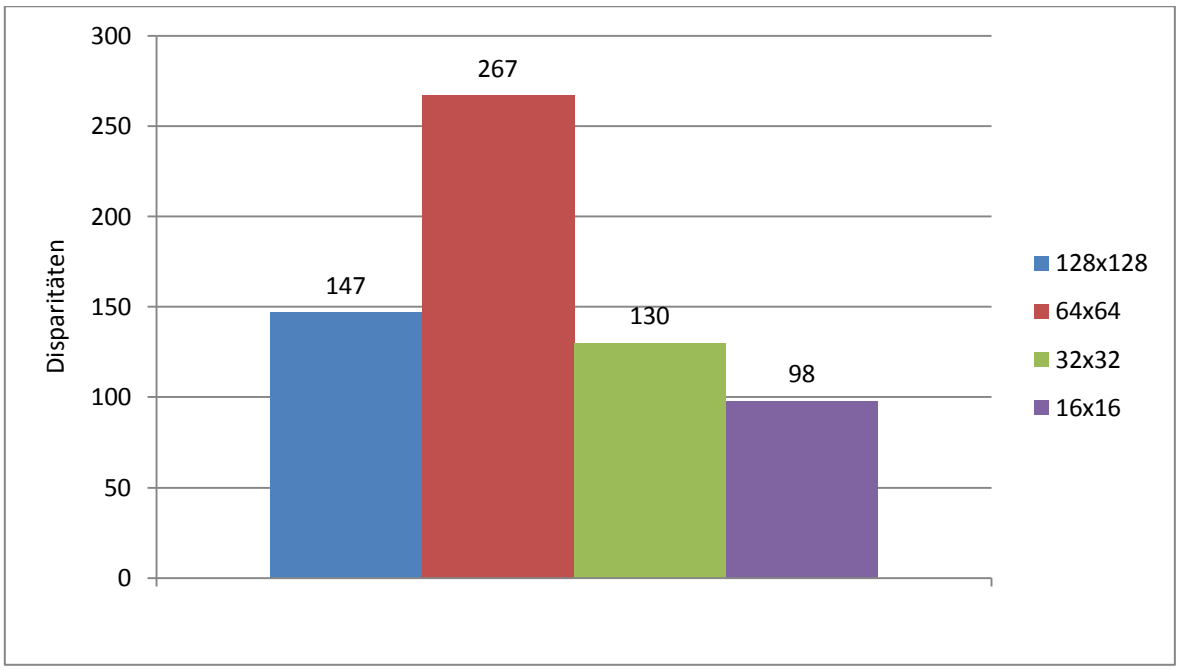

Abbildung 39: Disparitäten Korrelationsalgorithmus

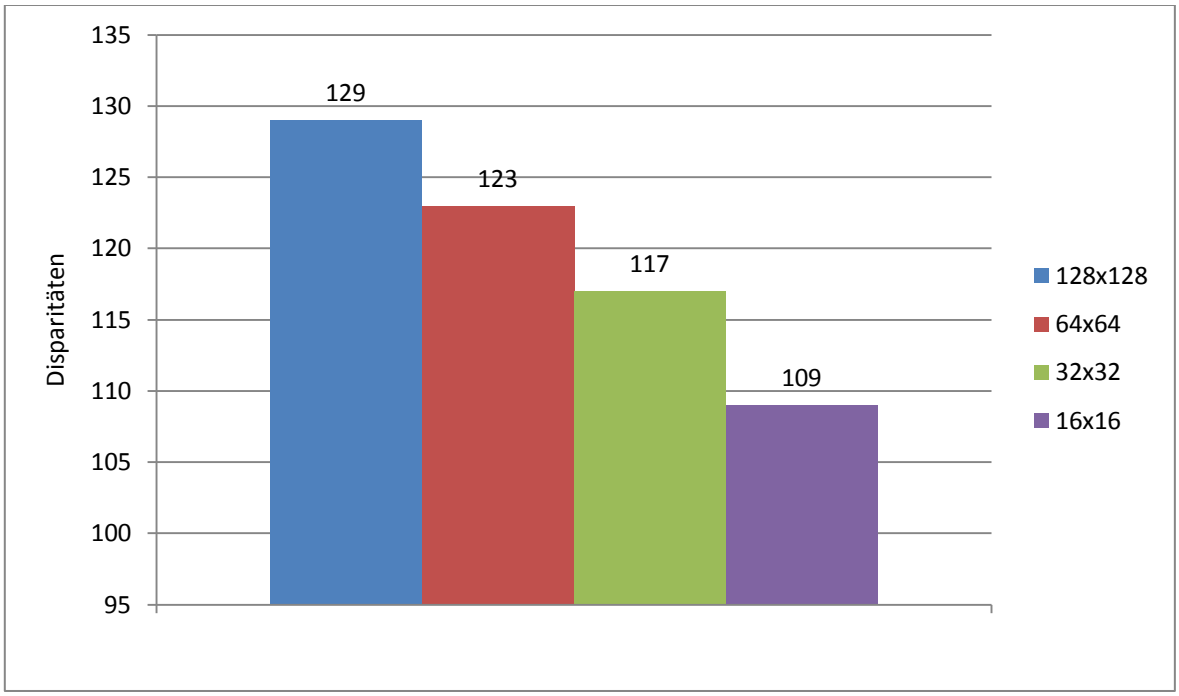

Abbildung 40: Disparitäten Template Matching

Alles in allem lässt sich über die drei Algorithmen sagen, dass nur zwei von ihnen geeignet sind. Dieses lässt sich vor allem daran fest machen, dass der Korrelationsalgorithmus zu große Schwankungen und schlechte Ergebnisse liefert.

2.5. Fehlerabschätzung

Die Fehlerabschätzung soll zeigen, wie stark die Abweichung ist, wenn der Algorithmus sich um Pixel verrechnet. Dazu wird die Entfernung aus der Disparitäten Berechnung als Reverenz genommen und ein Pixel Abweichung in die Formel eingefügt.

$$\frac{0{,}499m * 0{,}00802m}{129 * 4{,}65\mu m} = 6{,}67m$$

Daraus ergibt sich ein Fehler von.

$$6{,}67m - 6{,}62m = 0{,}05m$$

Bei dieser Entfernung ergibt sich aus der Tabelle, dass dieser Fehler vernachlässigbar ist.

Tabelle 4: Ausschnitt aus Tiefenschärfetabelle

4,00	3,64	4,43	0,79
5,00	4,45	6,70	2,25
6,00	5,23	7,63	2,40

Wenn nun die ersten beiden Werte des Pyramidalgorithmus hergenommen werden um zu sehen, ob eine Abweichung von acht Pixeln akzeptabel ist, ergeben sich folgende Rechnungen.

$$\frac{0{,}499m * 0{,}00802m}{138 * 4{,}65\mu m} = 6{,}23m$$

Daraus ergibt sich ein Fehler von:

$$6{,}62m - 6{,}23m = 0{,}39m$$

Bei einem Fehler von ca. 40 cm kann man diesen auch noch tolerieren, wobei dieser schon näher an die Grenzen des Tiefenschärfebereichs kommt.

Durch diese Berechnung ist auch zu sehen, dass man von einer groben Abschätzung ausgehen kann, welche bei 5 cm pro Pixel liegt. Bei der Berechnung der Abstände aus Tabelle 3: Disparitäten im Vergleich ergab sich Tabelle 5: Abstände Berechnet. In dieser sind die Abstände jeder Disparität zu sehen. Aus diesen Daten ergibt sich, dass je größer

die Disparität desto kleiner die Entfernung. Ist die die Disparität jedoch klein wird die Entfernung größer.

Tabelle 5: Abstände Berechnet

	Pyramidalg.	Korrelationsalg.	Template Matching
128x128	6,23m	5,85m	6,67m
64x64	6,62m	3,22m	6,99m
32x32	2,62m	6,62m	7,35m
16x16	7,75m	8,78m	7,89m

3. Zusammenfassung und Ausblick

Für den Erfolg der Exo Mars Mission ist es notwendig hochauflösende Bilder in optimaler Qualität aufzunehmen. Hierzu ist es notwendig, einen Algorithmus, der einen Geschwindigkeitsvorteil bei der Fokussierung bringt, zu implementieren. Da die HRC die erste fokussierbare Kamera ist, die für den Weltraumeinsatz konzipiert wurde, gibt es kaum Erfahrungen auf diesem Gebiet. Diese Bachelorarbeit trägt dazu bei, dass das Wissen über die Stereoanalyse auf die Kameras angewandt wird. Das in C# entwickelte Programm lieferte hierbei sehr gute Ergebnisse.

Zu Beginn dieser Arbeit werden verschiedene Algorithmen vorgestellt, welche beim Bildvergleich genutzt werden. Diese sind in das Programm übernommen und unterschiedlichen Test ausgesetzt worden. Hierbei stellte sich heraus, dass der Pyramidalgorithmus besonders geeignet ist. Er liefert gute Ergebnisse zu einer akzeptablen Laufzeit.

Aus den in dieser Arbeit getätigten Schlussfolgerungen ergibt sich, dass ab einer Entfernung von 5 Metern größere Fehler gemacht werden können, da hier der Tiefenschärfebereich bei 2,25 Metern liegt.

Die Ansteuerung der Kamera ist in dieser Arbeit nicht möglich gewesen, da diese nur einen Pizo-Motor besitzt, der keinen Encoder enthält. Daraus ergibt sich, dass ein gefahrener Schritt nicht immer gleich groß ist und somit keine genauen Fokuseinstellungen möglich sind. Die Flugkamera wird einen Schrittmotor mit Encoder besitzen, wodurch eine genaue Fokussierung anhand der Tabelle möglich wird.

Eine mögliche Verbesserung an dem Programm liegt in einer Durchschnittsbildung. Es kann ein Durchschnitt aus den fünf Werten gebildet werden. Diese können aus dem Reverenzpunkt und vier Punkten, die jeweils um fünf Pixel nach oben, unten, links und rechts verschoben sind, genommen werden. Mithilfe dieser neuen Entfernungsdaten kann die Reverenz verifiziert werden. Des Weiteren können auch noch Bild verbessernde Maßnahmen getroffen werden. So zum Beispiel eine Bildnormalisierung mithilfe der Histogramm Methode. Dies führt dazu, dass die besseren Ergebnisse geliefert werden können. Bei den verbessernden Maßnahmen muss jedoch darauf geachtet werden, dass diese nicht zu viel Rechenleistung benötigen und ohne Kommazahlen auskommen. Das muss gewährleistet sein, um die Voraussetzungen für den Mars Rover zu erfüllen.

Mögliche nächste Schritte sind die Übertragung dieses Algorithmus auf das Zielsystem. Anschließend müssen Tests durchgeführt werden, um zu sehen ob sich die Vermutungen bestätigen.

Literaturverzeichnis

(Bredn 03) Bredner, Barbara. Statistische Beratung und Lösungen. 15. 08 2010 <http://www.bb-sbl.de/assets/images/tutorial/scatterplot.jpg>.

(Brink 08) Brinkhoff, Thomas. Geodatenbanksysteme in Theorie und Praxis: Einführung in objektrelationale Geodatenbanken unter besonderer Berücksichtigung von Oracle Spatial. Wichmann, 2008.

(Brune 09) Brunelli, Roberto. Template Matching Techniques in Computer Vision: Theory and Practice. John Wiley & Sons, 2009.

(Richa 01) Dr. Richard Carl Dehmel. Version 2001 <http://www.cs.berkeley.edu/~demmel/cs267/lecture26/Quadtree1.gif>.

(DLR) Deutsches Zentrum für Luft- und Raumfahrt

(ESA 00) ESA. Version 2000 <http://spacewire.esa.int/content/Home/HomeIntro.php>. ESA exomars..

(ESA 07) European Space Agency Version 2007 <http://www.esa.int/SPECIALS/ExoMars/SEMK39JJX7F_0.html>. ESA ExoMars Mission Information Sheet.

(esa 08) ESA Version 2008 <http://esamultimedia.esa.int/docs/ESA_ExoMars_Mission_Information_Sheet_rev01Feb08.pdf>.

(ESA 04) ESA. The Aurora Programme. Version 2004 http://esamultimedia.esa.int/docs/Aurora/Aurora625_2.pdf>.

(Fermu 09) Fermum, Lars. Optischer Strahlengang Bikonvex. Version 2009 <http://www.vision-doctor.de/images/stories/optik/grundlagen/Optik_Strahlengang_bikonvex.jpg>

(Flemi 04) Fleming, Don.Version 2004 DOF Master. <http://www.dofmaster.com/dofjs.html>.

(Gary 08) Gary, Bradski und Kaehler Adrian. „Learning OpenCV.“ United States of America: O'Reilly, 2008. 416-417.

(Prusc 05) Pruscha, Helmut. „Statistisches Methodenbuch.“ Springer, 2005. 34-37.

(Rees 07) Rees, Martin. Universe: The Definitive Visual Guide. Penguin , 5. September 2007.

(Ulric 01) Ulrich Clamor, Schmidt-Ploch. Die Lochkamera. SP-Verlag, 2001. 61-66.

(Grandi 09) Gandi79 . Version 2009

<http://de.wikipedia.org/wiki/Datei:Optik_Schaerfentiefe_Strahlengang_01.png>.

Abbildungsverzeichnis

Anhang

Anhang 1: Auflösungsebenen

```
//Auflösungsebenen
        private int ebene0;
        private int[,] ebene1 = new int[2, 2];
        private int[,] ebene2 = new int[4, 4];
        private int[,] ebene3 = new int[8, 8];
        private int[,] ebene4 = new int[16, 16];
        private int[,] ebene5 = new int[32, 32];
        private int[,] ebene6 = new int[64, 64];
        private int mittelwert = 0;
        private int groesse;
```

Anhang 2: Pyramidalgorithmus

```
private void berechnungEbene0(int[,] arg1)
        {
            //Mittelwert bilden bilden und Ebene0 zuteilen
            mittelwert += ebene1[0, 0];
            mittelwert += ebene1[0, 1];
            mittelwert += ebene1[1, 0];
            mittelwert += ebene1[1, 1];
            mittelwert = mittelwert / 4;
            ebene0 = mittelwert;
            mittelwert = 0;
            //Statistik
            Schleifen.schleifendurchlaeufe++;
        }

        private void berechnungEbene1(int[,] arg1)
        {
            int zaehl1 = 0;
            int zaehl2 = 0;

            //Y-Koordinate durchgehen
            for (int i = 0; i < 2; i++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                //X-Koordinate durchgehen
                for (int q = 0; q < 2; q++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    //4x4Matrix gurchgehen
                    for (int y = 0; y < 2; y++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        for (int x = 0; x < 2; x++)
                        {
                            //Statistik
                            Schleifen.schleifendurchlaeufe++;
                            //Mittelwerte addieren
                            mittelwert += arg1[zaehl1, zaehl2];
                            //zaehl1 um eins erhöhen
                            zaehl1++;
                        }
                        //zaehl2 um eins erhöhen und zaehl1 q*2 zuweisen
                        zaehl2++;
```

```
                    zaehl1 = q * 2;
                }
                //zaehl1 q*2 zuweisen und zaehl2 i*2 zuweisen
                zaehl1 = q * 2;
                zaehl2 = i * 2;
                //mittelwert berechnen
                mittelwert = mittelwert / 4;


                //Mittelwert dem Punkt der Matrix zuweisen und Mittelwert 0
setzen
                ebene1[q, i] = mittelwert;
                mittelwert = 0;
            }
        }
        //Nächste Ebene Berechnen
        berechnungEbene0(ebene1);
    }

    private void berechnungEbene2(int[,] arg1)
    {
        int zaehl1 = 0;
        int zaehl2 = 0;

        //Y-Koordinate durchgehen
        for (int i = 0; i < 4; i++)
        {
            //Statistik
            Schleifen.schleifendurchlaeufe++;
            //X-Koordinate durchgehen
            for (int q = 0; q < 4; q++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                //4x4Matrix gurchgehen
                for (int y = 0; y < 2; y++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    for (int x = 0; x < 2; x++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        //Mittelwerte addieren
                        mittelwert += arg1[zaehl1, zaehl2];
                        //zaehl1 um eins erhöhen
                        zaehl1++;
                    }
                    //zaehl2 um eins erhöhen und zaehl1 q*2 zuweisen
                    zaehl2++;
                    zaehl1 = q * 2;
                }
                //zaehl1 q*2 zuweisen und zaehl2 i*2 zuweisen
                zaehl1 = q * 2;
                zaehl2 = i * 2;
                //mittelwert berechnen
                mittelwert = mittelwert / 4;


                //Mittelwert dem Punkt der Matrix zuweisen und Mittelwert 0
setzen
                ebene2[q, i] = mittelwert;
                mittelwert = 0;
            }
```

```
            }
            //Nächste Ebene Berechnen
            berechnungEbene1(ebene2);
        }

        private void berechnungEbene3(int[,] arg1)
        {
            int zaehl1 = 0;
            int zaehl2 = 0;

            //Y-Koordinate durchgehen
            for (int i = 0; i < 8; i++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                //X-Koordinate durchgehen
                for (int q = 0; q < 8; q++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    //4x4Matrix gurchgehen
                    for (int y = 0; y < 2; y++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        for (int x = 0; x < 2; x++)
                        {
                            //Statistik
                            Schleifen.schleifendurchlaeufe++;
                            //Mittelwerte addieren
                            mittelwert += arg1[zaehl1, zaehl2];
                            //zaehl1 um eins erhöhen
                            zaehl1++;
                        }
                        //zaehl2 um eins erhöhen und zaehl1 q*2 zuweisen
                        zaehl2++;
                        zaehl1 = q * 2;
                    }
                    //zaehl1 q*2 zuweisen und zaehl2 i*2 zuweisen
                    zaehl1 = q * 2;
                    zaehl2 = i * 2;
                    //mittelwert berechnen
                    mittelwert = mittelwert / 4;


                    //Mittelwert dem Punkt der Matrix zuweisen und Mittelwert 0
setzen
                    ebene3[q, i] = mittelwert;
                    mittelwert = 0;
                }
            }
            //Nächste Ebene Berechnen
            berechnungEbene2(ebene3);
        }

        private void berechnungEbene4(int[,] arg1)
        {
            int zaehl1 = 0;
            int zaehl2 = 0;

            //Y-Koordinate durchgehen
            for (int i = 0; i < 16; i++)
            {
                //Statistik
```

```
                    Schleifen.schleifendurchlaeufe++;
                    //X-Koordinate durchgehen
                    for (int q = 0; q < 16; q++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        //4x4Matrix gurchgehen
                        for (int y = 0; y < 2; y++)
                        {
                            //Statistik
                            Schleifen.schleifendurchlaeufe++;
                            for (int x = 0; x < 2; x++)
                            {
                                //Statistik
                                Schleifen.schleifendurchlaeufe++;
                                //Mittelwerte addieren
                                mittelwert += arg1[zaehl1, zaehl2];
                                //zaehl1 um eins erhöhen
                                zaehl1++;
                            }
                            //zaehl2 um eins erhöhen und zaehl1 q*2 zuweisen
                            zaehl2++;
                            zaehl1 = q * 2;
                        }
                        //zaehl1 q*2 zuweisen und zaehl2 i*2 zuweisen
                        zaehl1 = q * 2;
                        zaehl2 = i * 2;
                        //mittelwert berechnen
                        mittelwert = mittelwert / 4;


                        //Mittelwert dem Punkt der Matrix zuweisen und Mittelwert 0
setzen
                        ebene4[q, i] = mittelwert;
                        mittelwert = 0;
                    }
                }
                //Nächste Ebene Berechnen
                berechnungEbene3(ebene4);
            }

            private void berechnungEbene5(int[,] arg1)
            {
                int zaehl1 = 0;
                int zaehl2 = 0;

                //Y-Koordinate durchgehen
                for (int i = 0; i < 32; i++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    //X-Koordinate durchgehen
                    for (int q = 0; q < 32; q++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        //4x4Matrix gurchgehen
                        for (int y = 0; y < 2; y++)
                        {
                            //Statistik
                            Schleifen.schleifendurchlaeufe++;
                            for (int x = 0; x < 2; x++)
                            {
                                //Statistik
```

```
                            Schleifen.schleifendurchlaeufe++;
                            //Mittelwerte addieren
                            mittelwert += arg1[zaehl1, zaehl2];
                            //zaehl1 um eins erhöhen
                            zaehl1++;
                        }
                        //zaehl2 um eins erhöhen und zaehl1 q*2 zuweisen
                        zaehl2++;
                        zaehl1 = q * 2;
                    }
                    //zaehl1 q*2 zuweisen und zaehl2 i*2 zuweisen
                    zaehl1 = q * 2;
                    zaehl2 = i * 2;
                    //mittelwert berechnen
                    mittelwert = mittelwert / 4;


                    //Mittelwert dem Punkt der Matrix zuweisen und Mittelwert 0
setzen
                    ebene5[q, i] = mittelwert;
                    mittelwert = 0;
                }
            }
            //Nächste Ebene Berechnen
            berechnungEbene4(ebene5);
        }

        public void berechnungEbene6(int[,] arg1)
        {
            int zaehl1 = 0;
            int zaehl2 = 0;

            //Y-Koordinate durchgehen
            for (int i = 0; i < 64; i++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                //X-Koordinate durchgehen
                for (int q = 0; q < 64; q++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    //4x4Matrix gurchgehen
                    for (int y = 0; y < 2; y++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        for (int x = 0; x < 2; x++)
                        {
                            //Statistik
                            Schleifen.schleifendurchlaeufe++;
                            //Mittelwerte addieren
                            mittelwert += arg1[zaehl1, zaehl2];
                            //zaehl1 um eins erhöhen
                            zaehl1++;
                        }
                        //zaehl2 um eins erhöhen und zaehl1 q*2 zuweisen
                        zaehl2++;
                        zaehl1 = q * 2;
                    }
                    //zaehl1 q*2 zuweisen und zaehl2 i*2 zuweisen
                    zaehl1 = q * 2;
                    zaehl2 = i * 2;
                    //mittelwert berechnen
```

```
                    mittelwert = mittelwert / 4;


                    //Mittelwert dem Punkt der Matrix zuweisen und Mittelwert 0
setzen
                    ebene6[q, i] = mittelwert;
                    mittelwert = 0;
                }
            }
            //Nächste Ebene Berechnen
            berechnungEbene5(ebene6);
        }
```

Anhang 3: Vergleich Pyramidalgorithmus

```
public int unterschiedBerechnen(Aufloesungsmatrix arg1)
        {
            int unterschied = 0;
            int i = 0;
            int q = 0;

            unterschied = Math.Abs(ebene0 - arg1.getEbene0());

            for (i = 0; i < 2; i++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                for (q = 0; q < 2; q++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    unterschied += Math.Abs(ebene1[q, i] - arg1.getEbene1()[q, i]);
                }
            }

            for (i = 0; i < 4; i++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                for (q = 0; q < 4; q++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    unterschied += Math.Abs(ebene2[q, i] - arg1.getEbene2()[q, i]);
                }
            }
            if (groesse >= 16)
            {
                for (i = 0; i < 8; i++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    for (q = 0; q < 8; q++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        unterschied += Math.Abs(ebene3[q, i] - arg1.getEbene3()[q,
i]);
                    }
                }
            }

            if (groesse >= 32)
```

```
            {
                for (i = 0; i < 16; i++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    for (q = 0; q < 16; q++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        unterschied += Math.Abs(ebene4[q, i] - arg1.getEbene4()[q,
i]);
                    }
                }
            }

            if (groesse >= 64)
            {
                for (i = 0; i < 32; i++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    for (q = 0; q < 32; q++)
                    {
                        //Statistik
                        Schleifen.schleifendurchlaeufe++;
                        unterschied += Math.Abs(ebene5[q, i] - arg1.getEbene5()[q,
i]);
                    }
                }
            }
            if (groesse == 128)
            {

                for (i = 0; i < 64; i++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    for (q = 0; q < 64; q++)
                    {
                        Schleifen.schleifendurchlaeufe++;
                        unterschied += Math.Abs(ebene6[q, i] - arg1.getEbene6()[q,
i]);
                    }
                }
            }

            return unterschied;

        }
```

Anhang 4: Korrelationsalgorithmus Reverenz

```
public void berechneReverenz(int[] arg1)
        {
            original = arg1;
            //Summe der Reverenz bilden
            for (int i = 0; i < arg1.Length; i++)
            {
                Schleifen.schleifendurchlaeufe++;
                sum1 += arg1[i];
            }
            //Durchschnitt bilden
            sum1 = sum1 / arg1.Length;

        }
```

Anhang 5:Korrelationsalgorithmus Berechnung

```
public double berechneCorrelation(int[] arg1)
        {

           for (int i = 0; i < arg1.Length; i++)
            {
                //Summe bilden
                sum2 += arg1[i];
                //Quadrat bilden aus dem Original minus dem
                //Durchschnitt der Reverenz und alle Werte zusammenaddieren
                sum4 += Math.Pow((original[i] - sum1), 2);
                //Statistik
                Schleifen.schleifendurchlaeufe++;
            }
            //Durchschnitt bilden
            sum2 = sum2 / arg1.Length;


            for (int i = 0; i < arg1.Length; i++)
            {
                /*Reverenzpixel vom Durchschnitt der Reverenz abziehen, Suchpixel
                 * vom Durchscnitt der Suchmartix abziehen, Produkt aus beiden
bilden
                 */
                sum3 += (original[i] - sum1) * (arg1[i] - sum2);
                //Quadrat bilden aus Suchpixel minus Durchscnitt der Suchmatrix
                sum5 += Math.Pow((arg1[i] - sum2), 2);
                //Statistik
                Schleifen.schleifendurchlaeufe++;
            }
            //Statistik
            Schleifen.schleifendurchlaeufe++;
            //Wurzel aus sum4 und sum5
            var_x = Math.Sqrt((sum4 * sum5));
            //Korrelationskoeffizient berechnen aus sum3 geteilt durch var_x
            korrekoef = sum3 / var_x;

            //Sum2 bis Sum5 wieder auf 0 setzen
            sum2=0;
            sum3 = 0;
            sum4 = 0;
            sum5 = 0;
            //Rückgabe des Korrelationskoeffizient
            return korrekoef;
        }
```

Anhang 6: Template Matching

```
public int berechneTemplateUnterschied(int[,] rev, int[,] suchmatrix, int groesse)
        {
            //Zwischenwert und Unterschied auf 0 setzen
            int unterschied = 0;
            int zwischenwert = 0;

            for (int i = 0; i < groesse; i++)
            {
                //Statistik
                Schleifen.schleifendurchlaeufe++;
                for (int q = 0; q < groesse; q++)
                {
                    //Statistik
                    Schleifen.schleifendurchlaeufe++;
                    //Zieht den Reverenzpixel vom Suchpixel ab, absoluter Wert wird
verwendet
                    zwischenwert = Math.Abs(rev[q, i] - suchmatrix[q, i]);
                    //hinzuaddieren zum Unterschied
                    unterschied += zwischenwert;
                }
            }
            //Rückgabe des Unterschieds
            return unterschied;
        }
```