Lehrstuhl für Raumfahrttechnik
Prof. Prof. h.c. Dr. Dr. h.c.
Ulrich Walter

Technische Universität München

Bachelorarbeit

Integration eines Echtzeit-Algorithmus zur Objektrekonstruktion aus Stereo-Bilddaten in eine Simulationsumgebung zur Nahbereichsnavigation im Orbit

LRT-BA 2016/06

Autor:

Mirko Gschwindt

Betreuer: Dipl.-Ing. Martin Dziura
Lehrstuhl für Raumfahrttechnik / Institute of Astronautics
Technische Universität München

Zusammenfassung

Das Thema „Weltraumschrott“ ist ein großes Problem in der Raumfahrt, da er nicht kontrollierbar ist und somit mit Raumfahrzeugen auf den Orbit-Bahnen kollidieren und sie beschädigen kann. Deshalb beschäftigt sich die Raumfahrtforschung mit dem Gebiet des *On Orbit Servicing* (OOS), also der Reparatur von Satelliten und Beseitigung von Weltraumschrott. Der Lehrstuhl für Raumfahrttechnik (LRT) der Technischen Universität München (TUM) hat zu diesem Zweck einen Simulator für Nahbereichsoperationen, das *Robotic Actuation and On-Orbit Navigation Laboratory* (RACOON), entwickelt.

Im Rahmen dieser Arbeit wurde ein echtzeitfähiger Stereoalgorithmus zur Tiefenbilderstellung aus Stereo-Bildpaaren in die Simulationsumgebung integriert. Dies stellt eine Grundlage zur Objektrekonstruktion von unkooperativen *Targets* im Weltall dar. Die Auswahl des Algorithmus wurde durch Betrachtung und Vergleich verschiedener *State-of-the-Art*-Stereoalgorithmen getroffen. Hierbei wurden alle potentiellen Algorithmen kategorisiert und anschließend anhand der gestellten Anforderungen überprüft.

Implementiert wurde der Algorithmus durch Erweiterung eines bestehenden Software-*Frameworks* zur 3D-Rekonstruktion. Testergebnisse zeigen, dass sich per Stereorekonstruktion zwar Tiefenbilder in RACOON erstellen lassen, die Lichtbedingungen allerdings so erschwerend sind, dass weitere Nachbearbeitung notwendig ist.

Abstract

The topic of space debris is a big issue of space flight, because it is not controllable and can therefore collide with and damage space crafts on their orbits. Hence space flight research is engaged in *On Orbit Servicing* (OOS), i.e. the repair of satellites and removal of space debris. For this purpose the Institute of Astronautics at Technische Universität München (TUM) developed a simulator for close range operations, the *Robotic Actuation and On-Orbit Navigation Laboratory* (RACOON).

As part of this thesis a real time stereo algorithm for the creation of depth images from pairs of stereo images was incorporated into the simulation environment. This establishes a basis for object reconstruction of uncooperative targets in space. The algorithm was selected by inspection and comparison of various state-of-the-art stereo algorithms. All potential algorithms were categorized and reviewed according to the set requirements.

The algorithm was implemented by expanding an existing software framework for 3D reconstruction. Test results show that depth images can be created by stereo reconstruction in RACOON, however the lightning conditions hinder the reconstruction so much, that further post processing is necessary.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

AD	*Absolute Intensity Difference*
ASWA	*Addaptive Support-Weight Aggregation*
CAD	*Computer-Aided Design*
EKF	*Extended Kalman Filter*
FWMAV	*Flapping Wing Micro Air Vehicle*
GPU	*Graphics Processing Unit*
ISS	*International Space Station*
LCIDR	*Low-Complexity Iterative Disparity Refinement*
LRT	Lehrstuhl für Raumfahrttechnik
MS	*Mesh Stereo*
MSE	*Mesh Stereo Extended*
NCC	*Normalized Cross Correlation*
OOS	*On Orbit Servicing*
OpenCV	*Open Source Computer Vision*
RACOON	*Robotic Actuation and On-Orbit Navigation Laboratory*
RANSAC	*Random Sample Consensus*
RGBD	*Red Green Blue Depth*
SAD	*Sum of Absolute Differences*
SD	*Squared Intensity Difference*
SDK	*Software Development Kit*
SGBM	*Semit-Global Block Matching*
SGM	*Semi-Global Matching*
SLAM	*Simultaneous Localization and Mapping*
SNCC	*Summed Normalized Cross-Correlation*
SSD	*Sum of Squared Differences*
S/C	*Space Craft*
TUM	Technische Universität München
WTA	*Winner Takes it All*

1. Einleitung

1.1. Motivation

Im Bereich der Raumfahrt ist Weltraumschrott (engl. *space debris*) seit einigen Jahren ein wichtiges Thema. Ausgediente Satelliten werden im Idealfall auf einen Friedhofsorbit gelenkt, um nicht mit aktiven Satelliten zu kollidieren. Dies geschieht aber aus Gründen wie Fehlfunktionen des Satelliten, abgebrochenem Kontakt oder Treibstoffmangel nicht immer, weshalb einige Satelliten sich auch nach ihrem Lebensende noch auf aktiven Orbits befinden und gefährlichen Weltraumschrott darstellen. Ein Ansatz dieses Problem zu lösen ist die Überführung dieser nicht mehr steuerbaren Satelliten auf einen Friedhofsorbit durch externes Eingreifen mittels eines anderen Satelliten. Dieser Vorgang sowie das Reparieren und Warten von Satelliten im Orbit wird als *On Orbit Servicing* (OOS) bezeichnet. Das *Chaser* genannte aktive *Space Craft* (S/C) nähert sich dabei dem antriebslosen *Target* in einem Manöver, das „Rendezvous“ genannt wird, an. Sobald der *Chaser* nah genug ist, kann ein *Docking*-Manöver durchgeführt werden, welches *Chaser* und *Target* verbindet. Während *Docking*-Manöver mit einem kooperativen Zielobjekt (beispielsweise der *International Space Station* (ISS)) in der Raumfahrt mittlerweile Routinearbeit sind, stellen sie bei einem unkooperativen *Target*, wie z.B. Weltraumschrott, noch immer einige Schwierigkeiten dar. Das angezielte Objekt kann beispielsweise eine unbekannte Form aufweisen oder unberechenbare Bewegungen ausführen. Ein besonders aktuelles Forschungsthema ist das autonome OOS, bei dem der Chaser-Satellit ohne menschliches Zutun selbstständig ein Rendezvous mit anschließendem Docking durchführt (vgl. [3]).

Um bei unbekannter Form oder Lage des *Targets* diese bestimmen zu können, wird eine Form der Visualisierung benötigt. Im Fall des autonomen OOS muss insbesondere ein 3D-Modell des Zielobjektes erstellt werden. Hierzu gibt es verschiedene Ansätze, welche im Stand der Technik noch genauer erläutert werden. Zur Erforschung der Nahbereichsoperationen robotischer Weltraumsysteme befindet sich im LRT der TUM das RACOON-Lab, eine Simulationsumgebung in der Manöver zum Warten von Satelliten und zur Entfernung von Weltraumschrott simuliert werden können. Hier soll die Möglichkeit geprüft werden, mit einer Stereokamera eine 3D-Objektrekonstruktion von unkooperativen *Targets* in Echtzeit durchzuführen.

1.2. Ziele und Aufgabenstellung

Ziel der Arbeit ist es, einen echtzeitfähigen Algorithmus zur 3D-Rekonstruktion aus Stereobilddaten in die Simulationsumgebung RACOON zu integrieren. Dazu werden verschiedene Algorithmen miteinenander verglichen, wovon einer ausgewählt und anschließend implementiert wird. Der Algorithmus muss Daten einer Stereokamera, die Bilder eines Satellitenmodells aufzeichnet, einlesen und anschließend eine Matrix mit Tiefenwerten des Satellitenmodells zurückgeben. Zur weiteren Verarbeitung des Tiefenbildes soll eine Schnittstelle zu bisher bestehender Software im RACOON geschaffen werden.

1.3. Arbeitsmethodik

Die Recherche der Arbeit umfasst ein weites Spektrum von Artikeln, Websites, Büchern und Präsentationen. Es wurde gezielt nach echtzeitfähigen Stereoalgorithmen und den mathematischen Hintergründen der Stereogeometrie recherchiert. Des Weiteren war auch die Suche nach ähnlichen Anwendungsbeispielen und Robotik in der Raumfahrt im allgemeinen ein Teil der Recherche. Im Anschluss wurden verschiedene Algorithmen anhand online verfügbarer Metriken miteinander verglichen und eine Auswahl getroffen. Die Implementierung des Algorithmus erfolgte in C++. Es wurden weiterhin Tests durchgeführt.

1.4. Stand der Technik

1.4.1. Stereoalgorithmen

Das Gebiet der Stereorekonstruktion ist gut erforscht und es wurden zahlreiche Veröffentlichungen zu diesem Thema geschrieben. Dabei gibt es für verschiedene Anwendungsfälle sehr unterschiedliche Arten von Algorithmen, die verschiedene Methodiken verwenden und unterschiedliche Zielsetzungen haben. Beispiele hierfür sind die Rekonstruktion von urbanen Gegenden [4], die autonome Navigation von Flugobjekten [5], [6] oder das autonome OOS im Weltall [3], [7]. Während es mittlerweile sehr präzise Rekonstruktionsalgorithmen gibt, sind viele der genannten Zielsetzungen von Echtzeitfähigkeit abhängig. Stereoalgorithmen basieren auf der Stereogeometrie von zwei Bildern desselben Objekts, aufgenommen von leicht versetzten Standpunkten. Ein Standardwerk zur Stereo- und Mehr-Sicht-Geometrie ist das Buch „Multiple View Geometry in computer vision“ [1] von Hartley und Zisserman. Es beschreibt die mathematischen Grundlagen der Stereogeometrie und zeigt auf, wie aus zwei Bildern ein 3D-Modell rekonstruiert werden kann. Die in diesem Buch beschriebenen Methoden stellen die Grundlage für alle Stereoalgorithmen dar.

Auf dieser Basis wurden zahlreiche Verfahren entwickelt, die aus einem Stereo-Bildpaar ein Tiefenbild rekonstruieren. Der aufwändige Teil ist dabei die Korrespondenzenfindung, also das Bestimmen, welcher Punkt in der primären Ansicht welchem Punkt in der sekundären Ansicht entspricht. Sobald man dieses Problem gelöst hat, kann man per Stereogeometrie leicht das zugehörige Tiefenbild finden. Zur Lösung des Korrespondenzenproblems wurden im Laufe der Jahre viele Ansätze vorgestellt. Bekannte Beispiele sind „Graph Cuts“ [8], „Dynamic Programming“ [9], „Scanline Optimization“ [10], „Belief Propagation“ [11] und „Normalized Cross Correlation“ [12]. Einige dieser Methoden weisen allerdings lange Laufzeiten auf, weshalb in den letzten Jahren viele Abwandlungen und neue Algorithmen entstanden sind, um Echtzeitfähigkeit zu erreichen. Gallup et al. [4] schlagen einen *plane-sweep*-Algorithmus zur 3D-Rekonstruktion von urbanen Gegenden aus Stereo-Bilddaten vor. Dieser Algorithmus ist echtzeitfähig und nutzt beispielsweise den *Graph Cuts*-Ansatz. De Wagter el al. [5] stellen einen Algorithmus zur Unterstützung des autonomen Fluges eines *Flapping Wing Micro Air Vehicle* (FWMAV) vor. Das FWMAV ist dabei mit einer Stereokamera ausgestattet und soll autonom Hindernissen ausweichen. Hierzu wurde ein an das spezielle Anwendungsfeld angepasster Algorithmus namens „LongSeq“ entwickelt, der einen Kompromiss aus hoher Bildverarbeitungsrate und Genauigkeit der Hinderniserkennung darstellt.

Auf dem Gebiet der Raumfahrt werden Stereoalgorithmen beim autonomen OOS eingesetzt. Hier sind vor allem sogenannte *Simultaneous Localization and Mapping* (SLAM)-Algorithmen von Bedeutung [3]. SLAM-Algorithmen dienen zur Positions- und Lageabschätzung bei unbemannten Flugkörpern. In der Raumfahrt ist die Anwendung vor allem bei der Nahbereichsnavigation von Satelliten, also bei *Rendezvous* und *Docking* zu finden. Zur autonomen Navigation des Satelliten ist Echtzeitfähigkeit äußerst wichtig. Daher wird auf diesem Gebiet die Verwendung verschiedener Stereoalgorithmen auf unbemannten S/Cs erforscht. Schnitzer et al. [3] stellen ein Konzept eines solchen SLAM-Algorithmus vor und prüfen ihre Ergebnisse mithilfe eines Simulators. Einen anderen Ansatz präsentieren Sonnenburg et al. [13], die einen *Extended Kalman Filter* (EKF)-SLAM-Algorithmus zur Rekonstruktion eines unbekannten, unkooperativen Target-S/C vorschlagen.

1.4.2. RACOON-Laboratory

Das RACOON-*Laboratory* [14] ist ein Forschungssimulator am LRT der TUM zur Simulation von teleoperierten OOS-Missionen. Das *Mission Statement* lautet: „Im Racoon-Lab werden Technologien für Nahbereichsoperationen robotischer Weltraumsysteme end-to-end entwickelt und evaluiert“. *end-to-end* bedeutet dabei, dass das gesamte System, vom Roboter im Weltall bis zum Menschen am Boden, simuliert wird. Um ein realistisches Missionsumfeld bereitzustellen, besteht RACOON aus einem *Mission Control Center* und einer Weltall-

Simulationsumgebung, in der Hardware-Komponenten unter realistischen Bedingungen getestet werden können. Weiterhin besteht ein sich sich stetig erweiterndes Software-Paket mit einer Mechanik-Simulation, *Computer-Aided Design* (CAD)-Modellen und Visualisierungs-Tools. Der Forschungsschwerpunkt von RACOON ist die Wartung von Satelliten und die Entfernung von Weltraumschrott. Die Testumgebung simuliert Nahbereichsoperationen in einer Entfernung von weniger als $20\,\mathrm{m}$ Abstand zum *Target*-S/C, ohne *Docking*. In dieses Projekt fließen einige Studien- und Forschungsarbeiten ein, die RACOON stetig weiter verbessern und neue Funktionalitäten hinzufügen. Das RACOON-Lab verfügt über ein um 5 Achsen rotierbares Satellitenmodell, einen jeweils 3-achsig translatorisch und rotatorisch beweglichen Sensorarm, der den *Chaser*-Satelliten simuliert, sowie eine auf Schienen fahrbare Beleuchtungseinrichtung, die die Lichtbedingungen im Weltall simuliert. Die in dieser Arbeit verwendete Bumblebee-Kamera ist auf dem Sensorarm montiert.

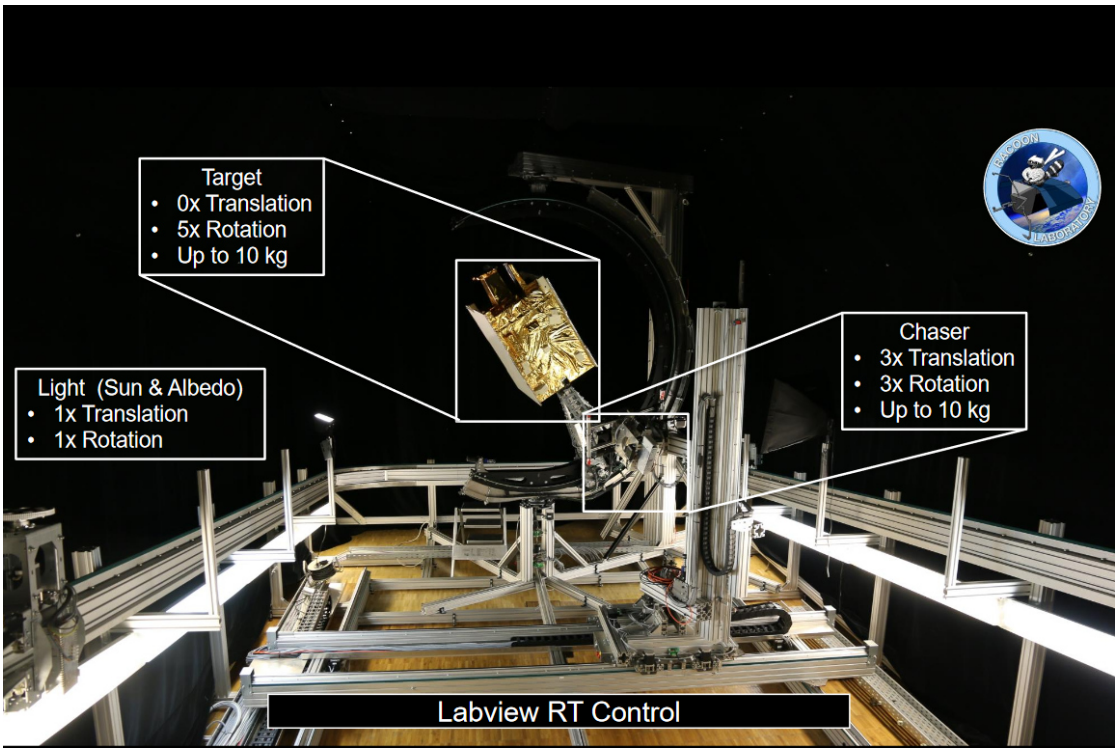

Abb. 1.1.: Aufbau des RACOON-Laboratory

Durch die Verknüpfung von *Mission Control Center* und *Hardware-in-the-Loop*-Simulator lassen sich komplette OOS-Missionsszenarien in RACOON simulieren. Ein wichtiger Bestandteil des OOS ist die Form- und Lageerkennung eines *Target*-S/C. An dieser Stelle reiht sich diese Arbeit in die Entwicklung am RACOON ein.

2. Algorithmenvergleich

2.1. Anforderungen an den Algorithmus

Der ausgewählte Algorithmus soll einigen Anforderungen gerecht werden. Seine Funktion ist das Berechnen eines Tiefenbildes aus zwei Bildern eines Satellitenmodells, die mittels einer Stereokamera aufgenommen werden. Genauere Spezifikationen werden in 2.1 erläutert.

Tab. 2.1.: Anforderungsspezifikationen des Algorithmus

Anforderungen	**Spezifizierung**
Echtzeitfähigkeit	Der Algorithmus soll eine Verarbeitungsfrequenz von mindestens $4\,\mathrm{Hz}$ aufweisen, also 4 Bildpaare pro Sekunde verarbeiten können.
Eingangsdaten	Die zu verarbeitenden RGB-Kamerabilder sind in der Auflösung $640 \times 480\,\mathrm{px}$.
Lichtverhältnisse	Der Algorithmus muss robust gegenüber den im RACOON voliegenden Lichtverhältnissen sein, welche die Lichtverhältnisse im Weltall simulieren.
Rechenleistung	Der Algorithmus soll auf dem Rechner in der Simulationsumgebung fließend ablaufen.
Verfügbarkeit	Der Algorithmus muss öffentlich verfügbar sein.

2.2. Mathematische Grundlagen der Stereogeometrie

Die mathematischen Grundlagen der Stereogeometrie werden ausführlich im Buch „Multiple View Geometry in computer vision“ [1] von Hartley und Zisserman im Kapitel „Two-View Geometry“ beschrieben. Im Folgenden soll eine Übersicht über das Thema verschafft werden. Die Stereogeometrie befasst sich mit der Rekonstruktion von 3D-Punkten aus zwei Kamerabildern einer Szene. Diese Bilder können entweder simultan mittels einer Stereokamera oder durch Translation einer einzelnen Kamera relativ zur Szene entstehen. Diese beiden Situationen sind geometrisch äquivalent und es wird daher im Weiteren von

einer Kameravorrichtung bestehend aus einer linken und einer rechten Kamera ausgegangen. Jeder der beiden Kameras wird eine Kameramatrix $\mathbf{P}$, $\mathbf{P}'$ zugeordnet, wobei ' einen Zusammenhang zur zweiten Kameraansicht anzeigt. Es gelten folgende Gleichungen:

$$\mathbf{x} = \mathbf{P}\mathbf{X} \tag{2.1}$$

$$\mathbf{x}' = \mathbf{P}'\mathbf{X} \tag{2.2}$$

$\mathbf{x}$ respektive $\mathbf{x}'$ stellen die zweidimensionalen Bildpunkte in der jeweiligen Ansicht als dreidimensionalen Vektor (mit der Z-Komponente = 1) dar. $\mathbf{X}$ ist ein dreidimensionaler Punkt, der als vierdimensionaler Vektor in homogenen Koordinaten angegeben wird. $\mathbf{P}$ und $\mathbf{P}'$ sind 3x4 Matrizen.
Gleichungen (2.1) und (2.2) beschreiben das Mapping einer *Pinhole*-Kamera von Punkten im Raum auf zweidimensionale Bildpunkte. Die Bildpunkte $\mathbf{x}$ und $\mathbf{x}'$ korrespondieren, da sie den selben 3D-Punkt abbilden.
Die epipolare Geometrie ist die intrinsische projektive Geometrie zwischen zwei Ansichten. Sie ist unabhängig von der Struktur der Szene und hängt lediglich von den internen Kameraparametern und der relativen Position der Kameras zueinander ab. Die Fundamentalmatrix $\mathbf{F}$ beschreibt diese intrinsische Geometrie. $\mathbf{F}$ ist eine 3x3 Matrix vom Rang 2. Es gilt:

$$\mathbf{x'}^{\mathrm{T}}\mathbf{F}\mathbf{x} = \mathbf{0} \tag{2.3}$$

Abb. 2.1 veranschaulicht die Geometrie der Punktkorrespondenzen. Die beiden Kameras werden durch ihre Kamerazentren $\mathbf{C}$ und $\mathbf{C}'$ und die entsprechenden Bildebenen dargestellt. Die Kamerazentren, der 3-dimensionale Punkt $\mathbf{X}$ und die Bildpunkte $\mathbf{x}$ und $\mathbf{x}'$ liegen alle auf auf einer gemeinsamen Ebene π, der sogenannten Epipolarebene. Die Verbindungslinie der beiden Kamerazentren heißt Grundlinie. Die Epipole $\mathbf{e}$ und $\mathbf{e}'$ sind die Schnittpunkte der Grundlinie mit den Bildebenen. Die Schnittlinien der Bildebenen mit der Epipolarebene heißen Epipolarlinien l und l'. Aus dieser Geometrie lässt sich bei Kenntnis von zwei Punkten aus $\mathbf{x}$, $\mathbf{x}'$ und $\mathbf{X}$ der dritte Punkt ermitteln. Bei der Suche nach einem korrespondierenden Punkt $\mathbf{x}'$ zu $\mathbf{x}$ beschränkt sich bei nicht bekannter Position von $\mathbf{X}$ die mögliche Position von $\mathbf{x}'$ auf l'. Dieser Zusammenhang wird auch bei Stereokorrespondenz-Algorithmen genutzt.

Die Fundamentalmatrix $\mathbf{F}$ ergibt sich aus den Kameramatrizen.

$$\mathbf{F} = [\mathbf{e}']_{\times}\mathbf{P}'\mathbf{P}^{+} \tag{2.4}$$

wobei $[\mathbf{e}']_{\times}$ die Kreuzproduktmatrix von $\mathbf{e}'$ und $\mathbf{P}^{+}$ die Pseudoinverse von $\mathbf{P}$ ist, d.h. $\mathbf{P}\mathbf{P}^{+} = \mathbf{I}$. Weiterhin gilt

$$\mathbf{e}' = \mathbf{P}'\mathbf{C} \tag{2.5}$$

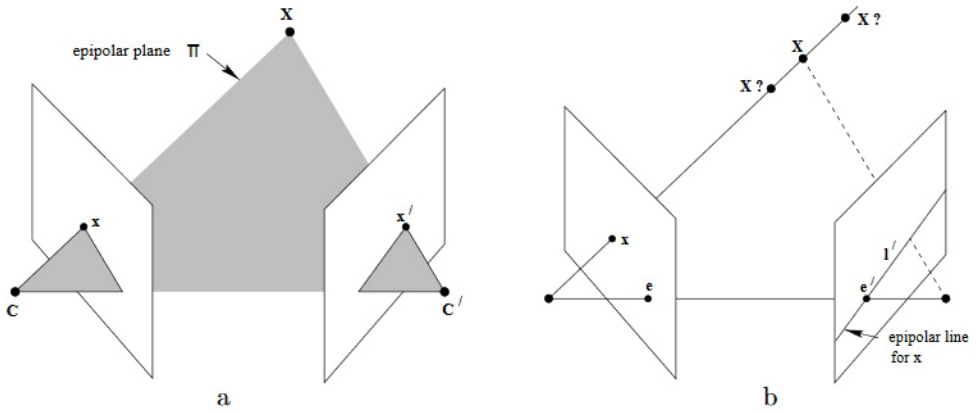

Abb. 2.1.: Punktkorrespondenz-Geometrie [1]

mit $\mathbf{PC} = \mathbf{0}$, wobei das Kamerazentrum $\mathbf{C}$ als 4-dimensionaler Vektor mit dem Wert 1 an vierter Position dargestellt wird. Falls eine Menge von Punktkorrespondenzen in zwei Ansichten die Fundamentalmatrix eindeutig bestimmen, dann können die Szene und die Kameras allein aus diesen Korrespondenzen bestimmt werden und jegliche Rekonstruktionen aus diesen Korrespondenzen sind projektiv äquivalent.
Um aus Punktkorrespondenzen ein Tiefenbild zu ermitteln, wird zuerst ein sogenanntes Disparitätsbild (engl. disparity image) erstellt. Dabei wird oft von *rectified images* ausgegangen, also von zwei Ansichten, in denen die Epipolaren kollinear und parallel zur x-Achse verlaufen. Gegeben sei ein Pixel $\mathbf{p}$ an der Position (x, y) in der linken und ein korrespondierendes Pixel $\mathbf{p}'$ an der Position (x', y) in der rechten Ansicht. Dann gilt für die Disparität d an dieser Stelle [15]:

$$d(\mathbf{p}, \mathbf{p}') = x - x' \tag{2.6}$$

Eine höhere Disparität bedeutet eine geringere Entfernung. Die Disparitäten werden in einem Disparitätsbild in Graustufen dargestellt, wobei dunklere Gebiete eine geringere Disparität (und damit eine größere Entfernung) darstellen.

2.3. Vergleich verschiedener Stereoalgorithmen

Stereoalgorithmen lassen sich auf verschiedene Weisen kategorisieren. In dieser Arbeit wurde die Unterteilung in lokale, globale und semiglobale Stereoalgorithmen gewählt. Die Unterschiede dieser Algorithmen liegen in der Art, auf die Punktkorrespondenzen ermittelt werden. Einen Vergleich der meisten gängigen Stereo-Algorithmen liefert die *Stereo Vision*-Website des Middlebury College [16].

2.3.1. Lokale Algorithmen

Die Präsentationsfolien von Prof. Navab und Dr. Unger [17] der TUM beschreiben das Verfahren der lokalen Matching-Algorithmen. Lokale Algorithmen vergleichen einzelne Pixel oder lokale Ausschnitte aus einer Ansicht mit denen der anderen Ansicht. Zur Korrespondenzfindung wird zu jedem Pixel eine *Matching Cost* zu anderen Pixeln berechnet. Je geringer diese *Matching Cost*, desto größer die Ähnlichkeit. Zwei Beispiele für die Berechnung der *Matching Cost* sind *Absolute Intensity Difference* (AD) (Gl. 2.7) und *Squared Intensity Difference* (SD) (Gl. 2.8).

$$|I_L(x,y) - I_R(x,y)| \tag{2.7}$$

$$(I_L(x,y) - I_R(x,y))^2 \tag{2.8}$$

$I_L(x,y)$ und $I_R(x,y)$ sind dabei die Intensitäten der Pixel. Ein einfacher lokaler Algorithmus ist der *Winner Takes it All* (WTA) Algorithmus, der zu jedem Pixel diejenige Korrespondenz mit der geringsten *Matching Cost* auswählt. Eine Verbesserung der Genauigkeit dieses Verfahrens kann durch *Cost Aggregation* entstehen. Hierbei wird ein Fenster bestehend aus mehreren Pixeln um den betrachteten Pixel herum gewählt und eine Kostenfunktion anhand der Summe der *Matching Costs* jedes Pixels im Fenster errechnet. Beispiele hierfür sind *Sum of Absolute Differences* (SAD) (Gl. 2.9) und *Sum of Squared Differences* (SSD) (Gl.2.10).

$$\sum_{x,y \in W} |I_L(x,y) - I_R(x,y)| \tag{2.9}$$

$$\sum_{x,y \in W} (I_L(x,y) - I_R(x,y))^2 \tag{2.10}$$

W steht hierbei für die Umgebung des Fensters (engl. window). Die Verknüpfung eines WTA Algorithmus mit SAD wird *Cross Correlation* genannt. Tab. 2.2 zeigt die Vor- und Nachteile einer lokalen Methode mit *Cross Correlation* auf.

Tab. 2.2.: Vor- und Nachteile von Cross Correlation

Vorteile	**Nachteile**
einfache Implementierung	Annahme konstanter Tiefe im Fenster
echtzeitfähige Anwendung	Schwierigkeiten bei repetitiven Elementen
geringe Speicheranforderungen	Schwierigkeiten bei größeren einheitlichen Flächen

Zwei der schnellsten lokalen Algorithmen, die in der Middlebury-Vergleichsliste [16] aufgeführt werden sind der IDR-Algorithmus und *Summed Normalized Cross-Correlation* (SNCC).

2.3.1.1. IDR

Der IDR-Algorithmus [15] ist der schnellste lokale Algorithmus der Middlebury-Evaluation. IDR ist ein Echtzeit-Algorithmus, der eine „two-pass"-Annäherung der *Addaptive Support-Weight Aggregation* (ASWA) und eine *Low-Complexity Iterative Disparity Refinement* (LCIDR)-Technik verwendet. Lokale Methoden verwenden im allgemeinen Stütz-Fenster zur Korrespondenzenfindung. ASWA [18] versucht adaptiv ein geeignetes Fenster zu jedem Pixel zu finden. Hierzu wird in einem gegebenen Fenster jedem Pixel ein *support weight* zugeordnet. Das *support weight* berechnet sich aus den geometrischen und photometrischen Beziehungen zum betrachteten Pixel. Diese Methode verbessert die *Matching*-Genauigkeit im Vergleich zu Methoden, die versuchen, die optimale Form oder Position des Stütz-Fensters zu bestimmen [19] [20]. Bei ASWA wird ein quadratisches Fenster um den betrachteten Pixel $\mathbf{p}$ gelegt. Jedem Pixel $\mathbf{q}$ dieses Fensters wird nun ein support weight zugeordnet. $\Delta_c(\mathbf{p},\mathbf{q})$ sei die Farbdifferenz und $\Delta_g(\mathbf{p},\mathbf{q})$ die geometrische Distanz zwischen $\mathbf{p}$ und $\mathbf{q}$, ausgewertet mit der Euklidischen Metrik. Das *support weight* von $\mathbf{q}$ bezogen auf $\mathbf{p}$ berechnet sich zu

$$w(\mathbf{p},\mathbf{q}) = -\exp\left(\frac{\Delta_c(\mathbf{p},\mathbf{q})}{\gamma_c} - \frac{\Delta_g(\mathbf{p},\mathbf{q})}{\gamma_g}\right) \tag{2.11}$$

wobei die Werte von γ_c und γ_g empirisch ermittelt werden. Um ein passendes *Match* zu finden, wird zu einem Pixel $\mathbf{p}$ im entsprechenden Korrespondenzgebiet der zweiten Ansicht gesucht. Sind $\mathbf{p}$ das betrachtete Pixel in der Referenzansicht, $\mathbf{p}'$ ein Pixel im Korrespondenzgebiet der zweiten Ansicht und $\Omega_{\mathbf{p}}$ und $\Omega_{\mathbf{p}'}$ die entsprechenden Stützfenster, dann berechnet sich die *Matching Cost* C nach folgendem Prinzip:

$$C(\mathbf{p},\mathbf{p}') = \frac{\sum_{\mathbf{q}\in\Omega_{\mathbf{p}},\mathbf{q}'\in\Omega_{\mathbf{p}'}} w(\mathbf{p},\mathbf{q})w(\mathbf{p}',\mathbf{q}')\delta(\mathbf{q},\mathbf{q}')}{\sum_{\mathbf{q}\in\Omega_{\mathbf{p}},\mathbf{q}'\in\Omega_{\mathbf{p}'}} w(\mathbf{p},\mathbf{q})w(\mathbf{p}',\mathbf{q}')} \tag{2.12}$$

wobei $\delta(\mathbf{q},\mathbf{q}')$ eine beliebige Distanzmessung zwischen Pixeln $\mathbf{q}$ und $\mathbf{q}'$ ist, typischerweise die Summe der absoluten Farbdifferenzen.

Bei der *two-pass*-Methode des IDR-Algorithmus wird ASWA durch *Cross-Correlation* in vertikaler und anschließend in horizontaler Richtung angenähert. Diese Annäherung verringert die Rechendauer, allerdings unter Qualitätseinbußen.

Weiterhin verwendet der IDR-Algorithmus LCIDR, um die Genauigkeit der

disparity map zu erhöhen. Bei dieser Methode wird nach Ausführen des WTA-Algorithmus die niedrigste *Matching Cost* mit der zweitniedrigsten verglichen. Dieses Verhältnis wird mit einem Vertrauenslevel angegeben. Das Vertrauenslevel vergleicht also die *Matching Cost* des gewinnenden Pixels mit der des nächsten Konkurrenten. Sobald die erste Iteration des *Stereo Matching* abgeschlossen ist, können Disparitätsabschätzungen zusammen mit den Vertrauensleveln genutzt werden, um die *Matching Cost* in nachfolgenden Iterationen zu optimieren. Dies geschieht durch das Hinzufügen von *cost penalties* zu Disparitätskandidaten, die zu weit von ihrem Erwartungswert abweichen.

Der IDR-Algorithmus erzielt in den Middlebury-Testsets gute Disparitätsergebnisse und ist mit einer durchschnittlichen Rechendauer von $0,36\,\mathrm{s/MP}$ pro Testpaar im Bereich der echtzeitfähigen Algorithmen einzuordnen. In der Testumgebung lief der Algorithmus auf einer NVIDIA GeForce TITAN Black GPU [16].

2.3.1.2. Summed Normalized Cross Correlation

Einer der schnelleren lokalen Algorithmen der Middlebury-Vergleichsseite ist SNCC [21]. SNCC verwendet eine zweistufige *Matching Cost*. In der ersten Stufe führt der Algorithmus eine Normalisierte *Cross Correlation* aus und aggregiert die ermittelten Korrelationswerte danach in einer zweiten Stufe. Dieses Verfahren lässt sich effizient implementieren und ist weniger empfindlich gegenüber hochkontrastigen Ausreißern als klassische lokale Stereoverfahren.
Für zwei Fenster aus den Kamerabildern I^L (links) und I^R (rechts) ist *Normalized Cross Correlation* (NCC) definiert als:

$$\rho_x = \frac{\frac{1}{|p(x)|}\sum_{\hat{x}\in p(x)}(I^L_{\hat{x}} - \mu^L_x)(I^R_{x'+d} - \mu^R_{x+d})}{\sigma^L_x \sigma^R_{x+d}} \tag{2.13}$$

mit

$$\mu_x = \frac{1}{|p(x)|}\sum_{\hat{x}\in p(x)} I_{\hat{x}} \tag{2.14}$$

$$\sigma_x = \sqrt{\frac{1}{|p(x)|}\sum_{\hat{x}\in p(x)} (I_{\hat{x}} - \mu_x)^2} \tag{2.15}$$

In den obenstehenden Gleichungen ist x die Position des betrachteten Pixels in der linken Ansicht, $p(x)$ ist das Set der Pixelkoordinaten des linken Fensters und $p(x+d)$ ist das Set der Pixelkoordinaten des rechten Fensters, d.h. d bezeichnet die Disparität zwischen rechtem und linkem Fenster. Da NCC von starken Kontrasten leicht verfälscht wird, verwendet SNCC eine zweistufige Filterung, um dieses Problem zu zu lösen. In der ersten Stufe wird NCC nach Gl. 2.13

mit einer sehr kleinen Filtergröße von 3x3 oder 5x5 berechnet. In der zweiten Stufe wird dann ein Summationsfilter auf die Ergebnisse der NCC-Filterung angewandt. Dies mittelt die Korrelationswerte über die Umgebung jedes Pixels an jeder Disparität. Dadurch wird die Feinstruktur des Bildes bewahrt und zur gleichen Zeit das Rauschen in der Schätzung verringert. Die neue Kostenfunktion ist damit definiert als:

$$\bar{\rho_x} = \frac{1}{|\tilde{p}(x)|} \sum_{\tilde{\tilde{x}} \in \tilde{p}(x)} \rho_{\tilde{\tilde{x}}} \tag{2.16}$$

wobei $\rho_{\tilde{\tilde{x}}}$ durch die Korrelationsgleichung 2.13 definiert wird und $\tilde{p}(x)$ das Set der Pixelkoordinaten des Summationsfilters ist.

Mit einer Rechenzeit von $1,02\,\mathrm{s/MP}$ liefert SNCC einen echtzeitfähigen *Stereo Matching*-Algorithmus. Die Qualität des Disparitätsbildes liegt im Vergleich der Middlebury-Website im Mittelfeld. Die Tests wurden auf einem i5-*Core*-Prozessor mit 3,1GHz Taktung durchgeführt. [16]

2.3.2. Globale Algorithmen

Im Gegensatz zu lokalen Algorithmen, die Blöcke mehrerer Pixel miteinander vergleichen, versuchen globale Algorithmen jedem Pixel einer Ansicht ein korrespondierendes Pixel in der zweiten Ansicht zuzuordnen. Dazu werden üblicherweise Energiefunktionen aufgestellt und minimiert. Kim, Kolmogorov und Zabih [22] beschreiben dieses Vorgehen wie folgt: $\mathcal{P}$ sei ein Set von Pixeln in der Primäransicht und $I_1 = \{I_1(\mathbf{p}) \mid \mathbf{p} \in \mathcal{P}\}$ und $I_2 = \{I_2(\mathbf{p}) \mid \mathbf{p} \in \mathcal{P}\}$ seien die Intensitäten in der Primär- und Sekundäransicht. Die zu bestimmende Größe ist die Disparitätskonfiguration $f = \{f_\mathbf{p} \mid \mathbf{p} \in \mathcal{P}\}$ in der Primäransicht. Jedes $f_\mathbf{p}$ repräsentiert die Korrespondenz zwischen dem Pixel $\mathbf{p}$ in der Primäransicht und dem Pixel $\mathbf{p} + f_\mathbf{p}$ in der Sekundäransicht. Die meisten globalen Algorithmen verwenden eine Annahme konstanter Helligkeit, d.h. $I_1(\mathbf{p}) \cong I_2(\mathbf{p} + f_\mathbf{p})$. Energieminimierungs-Algorithmen definieren die beste Disparitätskonfiguration f als diejenige, die die Energie minimiert, welche aus einem *Smootheness*- und einem Daten-Ausdruck besteht.

$$E(f) = E_{smooth}(f) + E_{data}(f). \tag{2.17}$$

Der *Smootheness*-Ausdruck verhängt eine Strafe für Konfigurationen, die räumliche Ebenheit verletzen. Der Daten-Ausdruck verhängt eine Strafe für Konfigurationen, die inkonsistent mit den beobachteten Intensitäten I_L und I_R des linken/ rechten Bildes sind. Der Standard-Daten-Term ist

$$E_{data}(f) = \sum_{\mathbf{p}} D_\mathbf{p}(f_\mathbf{p}) \tag{2.18}$$

wobei $D_{\mathbf{p}}$ ein Maß der Pixel-Verschiedenheit zwischen $I_1(\mathbf{p})$ und $I_2(\mathbf{p}+f_{\mathbf{p}})$ ist. Bei Annahme von konstanter Helligkeit gilt

$$D_{\mathbf{p}}(f_{\mathbf{p}}) = \rho(I_1(\mathbf{p}), I_2(\mathbf{p}+f_{\mathbf{p}})) \tag{2.19}$$

wobei ρ ein beliebiges Distanzmaß ist.

Globale Algorithmen liefern in der Regel bessere Ergebnisse, als lokale Algorithmen, allerdings auf Kosten von längeren Rechenzeiten. Häufig anzutreffende Verfahren sind *Graph Cuts* [8], *Dynamic Programming* [9], *Scanline Optimization* [10] und *Belief Propagation* [11]. Einer der exaktesten globalen Algorithmen der Middlebury-Vergeleichsseite ist *Mesh Stereo Extended* (MSE).

2.3.2.1. Mesh Stereo Extended

MSE ist eine erweiterte Form von *Mesh Stereo* (MS) [23]. Auf die Unterschiede wird später genauer eingegangen.
MS ist ein globaler Stereoalgorithmus, der neben einer Disparitätskarte zusätzlich direkt ein 3D-Dreiecks-Netz erstellt. Hierfür werden die Eingangsbilder in 2D Dreiecke mit geteilten Eckpunkten aufgeteilt. Anschließend werden durch Energieminimierung passende Disparitätswerte für die Eckpunkte berechnet. Das Anheben der 2D-Triangulation auf 3D liefert ein korrespondierendes Netz. Jedem Dreieck i wird eine Normale $\mathbf{n}_i = (n_{x,i}, n_{y,i}, 1)^{\mathrm{T}}$ und eine Disparität d_i seines Schwerpunktes $(\bar{x}_i, \bar{y}_i)$ zugeordent. $\mathbf{N}$ und $\mathbf{D}$ bezeichnen das Set der Normalen und Disparitäten aller Dreiecke. Die gesamte *Matching Cost* ist definiert als die Summe der Kosten aller Dreiecksebenen.

$$E_{MatchingCost}(\mathbf{N}, \boldsymbol{D}) = \sum_i \rho_{TRIANGLE}(\mathbf{n}_i, d_i) \tag{2.20}$$

Weiterhin wird eine Gleichmäßigkeitsbedingung für die Normalenpaare aneinander angrenzender Dreiecke mit ähnlicher Erscheinung eingeführt

$$E_{NormalSmooth}(\mathbf{N}) = \sum_{i,j\in\mathcal{N}} \omega_{ij}(\mathbf{n}_i - \mathbf{n}_j)^{\mathrm{T}}(\mathbf{n}_i - \mathbf{n}_j) \tag{2.21}$$

wobei $\omega_{ij} = \exp(-||c_i - c_j||_1/\gamma_1)$ die Ähnlichkeit der mittleren Farbwerte c_i, c_j der Dreiecke misst und γ_1 den Einfluss der Farbänderungen auf die Gewichtung beeinflusst. Jedem geteilten Eckpunkt s wird eine *Splitting*-Wahrscheinlichkeit α_s zugeordnet. Die *Splitting*-Wahrscheinlichkeiten aller Eckpunkte werden mit α bezeichnet. Ein *Prior* zu den *Splitting*-Wahrscheinlichkeiten wird durch einen Daten- und einen Ebenheitsterm modelliert

$$E_{SplitPenalty}(\boldsymbol{\alpha}) = \sum_s \alpha_s \tau_s \tag{2.22}$$

$$E_{SplitSmooth}(\boldsymbol{\alpha}) = \sum_{s,t \in \mathcal{N}} \omega_{st}(\alpha_s - \alpha_t)^2 \tag{2.23}$$

wobei die Strafe τ_s adaptiv gesetzt wird, basierend auf den Hinweisen, ob s an einer starken Kante liegt. $s,t \in \mathcal{N}$ bedeutet, dass s und t benachbarte Eckpunkte sind und ω_{st} ist eine Gewichtung, die durch die Distanz der visuellen Komplexitätslevel der Orte s und t bestimmt wird. Der Datenterm fördert *Splitting*, falls s auf einer starken Kante liegt. Der Ebenheitsterm fördert Gleichmäßigkeit, falls s und t von ähnlicher visueller Komplexität sind.

Die bisherigen Energieterme werden nun in einem Angleichungsterm kombiniert. Der Term fordert eine hohe Angleichung an geteilten Eckpunkten, die auf kontinuierlichen Oberflächen liegen, während er Eckpunkten, deren *Splitting*-Wahrscheinlichkeit hoch ist, mehrere Tiefenwerte erlaubt

$$E_{Alignment}(\mathbf{N}, \mathbf{D}, \boldsymbol{\alpha}) = \sum_{s}(1 - \alpha_s) \cdot \sum_{i,j \in G_s} \frac{1}{2}\omega_{ij}(\mathcal{D}_i(x_s, y_s) - \mathcal{D}_j(x_s, y_s))^2 \tag{2.24}$$

wobei G_s das Set von Dreiecken, die sich den Eckpunkt s teilen bezeichnet. $\mathcal{D}_i(x_s, y_s)$ bezeichnet die von der Ebenengleichung des Dreiecks i abgeleitete Disparität von s. An Stellen mit kontinuierlicher Tiefe (α_s nahe 0) fördert der Angleichungsterm eine Einigung der Disparität an s. An Stellen, an denen Tiefen-Diskontinuitäten vorliegen (α_s nahe 1) erlaubt er den Dreiecken in G_s ihre eigenen Disparitäten zu behalten. Kombiniert ergibt sich eine zu minimierende Gesamtenergiefunktion

$$\begin{aligned} E_{ALL}(\mathbf{N}, \mathbf{D}, \boldsymbol{\alpha}) = E_{MatchingCost} + \lambda_{NS}E_{NormalSmooth} + \lambda_{AL}E_{Alignment} \\ +\lambda_{SP}E_{SplitPenalty} + \lambda_{SS}E_{SplitSmooth} \end{aligned} \tag{2.25}$$

wobei die $\lambda_{\times}$ Skalierungsfaktoren sind. Die Optimierung dieser Funktion erfolgt durch iteratives Aktualisieren von $\boldsymbol{\alpha}$, was in geschlossener Form gelöst werden kann, und $\mathbf{N}, \mathbf{D}$, was durch ein gebietsbasiertes Stereo-Modell mit quadratischer Relaxation optimiert wird.
MSE verwendet die Berechnung der *Matching Cost* des MC-CNN-Algorithmus von Zbontar und LeCun [24].

MSE liefert mit die besten Ergebnisse der Middlebury-Testsets [16], allerdings ist er mit einer Rechenzeit von durchschnittlich $121\,\mathrm{s/MP}$ nicht echtzeitfähig.

2.3.3. Semi-Global Matching

Semi-Global Matching (SGM) [2], [25] ist eine von Heiko Hirschmüller entwickelte Methode, die die Vorteile von globalen und lokalen Algorithmen verknüpfen

soll. Der Anspruch des Algorithmus ist es, annähernd an die Rekonstruktionsgenauigkeit globaler Algorithmen heranzukommen, bei für lokale Algorithmen typischen geringen Rechenzeiten.
SGM basiert auf der Idee von pixelweisem *Matching* mit *Mutual Information* [26] und der Approximierung einer globalen, 2D-Ebenheitsbedingung durch Kombinieren vieler 1D-Bedingungen.

Die *Matching Cost* für ein betrachtetes Pixel $\mathbf{p}$ wird aus seiner Intensität $I_{b\mathbf{p}}$ ($b\mathbf{p}$ für engl. base pixel) und der vermuteten Korrespondenz $I_{m\mathbf{q}}$ bei $\mathbf{q} = e_{bm}(\mathbf{p}, d)$ (m für engl. Match) des *Match*-Bildes. Die Funktion $e_{bm}(\mathbf{p}, d)$ symbolisiert die Epipolarlinie im *Match*-Bild für das betrachtete Pixel $\mathbf{p}$ mit dem Linienparameter d. Für *rectified images* gilt

$$e_{bm}(\mathbf{p}, d) = [\mathbf{p}_x - d, \mathbf{p}_y]^{\mathrm{T}} \tag{2.26}$$

mit d als Disparität. Ein wichtiger Aspekt des *Matching* ist die Größe und Form der Fläche, die für das *Matching* berücksichtigt wird. Bei größeren Flächen wird die Robustheit des *Matching* erhöht, allerdings stimmt die Annahme konstanter Disparität innerhalb der Fläche dadurch häufiger nicht mehr, weshalb verschwommene Objektkanten häufiger auftreten. SGM verwirft die Annahme konstanter Disparitäten in der Umgebung von $\mathbf{p}$, was bedeutet, dass nur die Intensitäten $I_{b\mathbf{p}}$ und $I_{m\mathbf{q}}$ zur Berechnung der *Matching Cost* verwendet werden können. Die Berechnung der *Matching Cost* besiert auf *Mutual Information*. *Mutual Information* zweier Pixel wird maximal, falls beide Pixel identisch, und minimal, falls beide Pixel vollständig unabhängig voneinander sind. *Mutual Information* ist definiert durch die Entropie H der Bilder, also durch ihren Informationsgehalt, als auch durch ihre bedingte Entropie.

$$MI_{I_1,I_2} = H_{I_1} + H_{I_2} - H_{I_1,I_2} \tag{2.27}$$

Die Entropien werden aus den Wahrscheinlichkeitsverteilungen der Intensitäten in den zugehörigen Bildern berechnet. Die in SGM verwendete Definition der *Matching Cost* lautet

$$MI_{I_1,I_2} = \sum_{\mathbf{p}} mi_{I_1,I_2}(I_{1\mathbf{p}}, I_{2\mathbf{p}}) \tag{2.28}$$

mit

$$mi_{I_1,I_2}(i,k) = h_{I_1}(i) + h_{I_2}(k) - h_{I_1,I_2}(i,k) \tag{2.29}$$

wobei die Datenterme h aus den Wahrscheinlichkeitsverteilungen der korrespondierenden Intensitäten errechnet werden. Dies führt zu der Definition einer *Matching Cost*

$$C_{MI}(\mathbf{p}, d) = -mi_{I_b, f_D(I-m)}(i_{b\mathbf{p}}, I_{m\mathbf{q}}) \tag{2.30}$$

mit $\mathbf{q} = e_{bm}(\mathbf{p}, d)$.

Nun verwendet der Algorithmus eine hierarchische Berechnung, die rekursiv das (hochskalierte) Disparitätsbild, das bei halber Auflösung des ursprünglichen Disparitätsbildes berechnet wurde, benutzt. Ist die ursprüngliche Komplexität des Algorithmus von der Ordnung $O(WHD)(width \times height \times disparityrange)$, so wird die Laufzeit bei halber Auflösung um einen Faktor von $2^3 = 8$ reduziert. Bei Start mit einer Auflösung von $\frac{1}{16}$ und 3 anfänglichen Iterationen verbessert sich die Laufzeit um den Gesamtfaktor

$$1 + \frac{1}{2^3} + \frac{1}{4^3} + \frac{1}{8^3} + 3\frac{1}{16^3} \approx 1,14 \tag{2.31}$$

Da eine pixelweise *Matching Cost* oft dazu führt, dass falsche *Matches* eine geringere *Cost* als das wahre *Match* haben, wird eine zusätzliche Bedingung eingeführt, die Gleichmäßigkeit durch das Bestrafen von Änderungen der Disparitäten nebenstehender Pixel unterstützt. Die pixelweise *Matching Cost* und die Ebenheitsbedingung werden durch die Energiefunktion $E(D)$, die vom Disparitätsbild D abhängt, definiert.

$$E(D) = \sum_{\mathbf{p}} C(\mathbf{p}, D_{\mathbf{p}}) + \sum_{\mathbf{q} \in N_{\mathbf{p}}} P_1 T[|D_{\mathbf{p}} - D_{\mathbf{q}}| = 1] + \sum_{\mathbf{q} \in N_{\mathbf{p}}} P_2 T[|D_{\mathbf{p}} - D_{\mathbf{q}}| > 1] \tag{2.32}$$

Der erste Term ist die Summe aller Pixel-*Matching Costs* für die Disparitäten von D. Der zweite Term fügt eine konstante Strafe $P1$ für alle Pixel $\mathbf{q}$ in der Umgebung $N_{\mathbf{p}}$ von $\mathbf{p}$ hinzu, bei denen sich die Disparität geringfügig ändert (z.B. 1 Pixel). Der dritte Term fügt eine größere konstante Strafe $P2$ für alle größeren Disparitätsänderungen hinzu. Die geringere Strafe für kleinere Änderungen lässt eine Anpassung an geneigte oder gekrümmte Flächen zu, die konstante Strafe für alle größeren Änderungen bewahrt Diskontinuitäten. Das Problem kann nun als das Finden eines Disparitätsbildes D formuliert werden, das die Energie $E(D)$ minimiert. Um die Minimierung der Energiefunktion in einem 2D-Bild zu bewältigen, werden *Matching Costs* in 1D aus allen Richtungen gleichwertig vereinigt. Die vereinigte *Matching Cost* $S(\mathbf{p}, d)$ (S für engl. smooth) für ein Pixel $\mathbf{p}$ und Disparität d wird durch die Summierung der Kosten aller 1D-Minimum-Kosten-Pfade, die in Pixel $\mathbf{p}$ an der Disparität d enden ermittelt. (vgl. Abb.2.2).

Die Gesamtkosten berechnen sich damit zu

$$S(\mathbf{p}, d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p}, d) \tag{2.33}$$

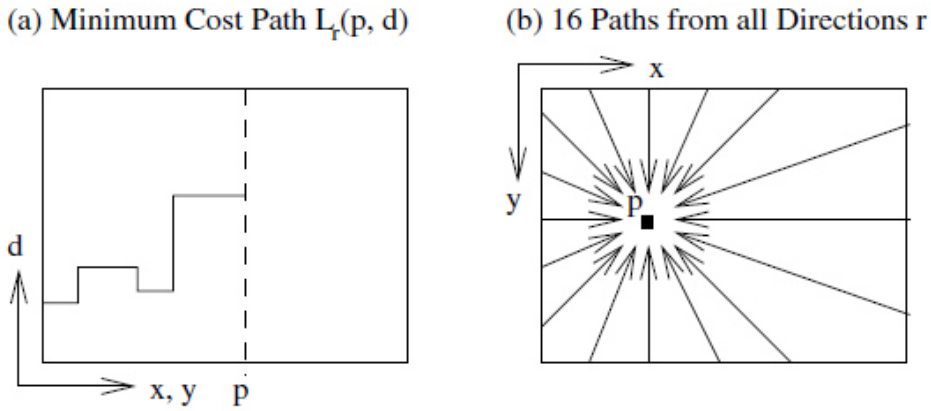

Abb. 2.2.: Vereinigung der Kosten [2]

wobei $L_{\mathbf{r}}(\mathbf{p}, d)$ die Minimum-Kosten-Pfade sind. Die Kosten $L_{\mathbf{r}}$ werden also über alle Richtungen $\mathbf{r}$ aufsummiert. Die Anzahl der Pfade muss zumindest 8 und sollte 16 für eine gute Abdeckung eines 2D-Bildes sein.

Das Dispartitätsbild D_b zur Basis-Ansicht I_b wird wie bei lokalen Stereomethoden durch Auswählen der Disparität d zu jedem Pixel $\mathbf{p}$, die zu den minimalen Kosten korrespondiert, also $min_d S(\mathbf{p}, d)$, bestimmt. Anschließend wird das Disparitätsbild D_m der *Match*-Ansicht berechnet. Der Vergleich beider Disparitätsbilder durch einen Konsistenzcheck erlaubt erlaubt die Bestimmung von Auslassungen und falschen *Matches*. Dabei wird jede Disparität von D_b mit der korrespondierenden Disparität von D_m verglichen und ugültig gesetzt (D_{inv}), falls sie unterschiedlich sind.

$$D_{\mathbf{p}} = \begin{cases} D_{b\mathbf{p}} & \text{für} |D_{b\mathbf{p}} - D_{m\mathbf{q}}| \leqslant 1, \mathbf{q} = e_{bm}(\mathbf{p}, D_{b\mathbf{p}}) \\ D_{inv} & \text{ansonsten} \end{cases} \tag{2.34}$$

Der Konsistenzcheck setzt die Eindeutigkeitsbedingung durch, indem er nur eins zu eins *Mapping* erlaubt.

SGM erzielt in den Middlebury-Testsets Ergebnisse im oberen Mittelfeld. Mit einer Laufzeit von $4,41\,\mathrm{s/MP}$ ist der Algorithmus zwar deutlich schneller als typische globale Algorithmen, allerdings ist er nicht schnell genug, um noch als echtzeitfähig eingestuft zu werden.

2.3.4. Semi-Global Block Matching

SGBM ist eine Modifikation des SGM-Algorithmus, der in der freien Programmbibliothek OpenCV implementiert ist. Die Unterschiede zu SGM sind [27]:

- SGBM ist ein *Single Pass* Algorithmus, was bedeutet, dass nur 5 anstatt 8 Richtungen berücksichtigt werden.

- Im Gegensatz zu SGM verlgeicht SGBM Blöcke anstatt einzelner Pixel
- *Mutual Information* ist nicht mehr als Kostenfunktion in SGBM implementiert. Stattdessen wird eine einfachere Birchfield-Tomasi Sub-Pixel-Metrik [28] angwandt. Farbbilder werden allerdings weiterhin unterstützt.
- Einige Vor- und Nachbearbeitungsschritte aus dem OpenCV-Algorithmus „StereoBM:operator()“ von K. Konolige sind in SGBM enthalten, z.B. *pre* und *post filtering*.

SGBM erzielt im Middlebury-Vergleich leicht schlechtere Ergebnisse als SGM und liegt damit insgesamt im Mittelfeld. Bei der Laufzeit liegt SGBM mit $0,68\,\mathrm{s/MP}$ allerdings deutlich vorne und kann mit diesem Wert als echtzeitfähig bezeichnet werden.

2.4. Auswahl und Ergebnisse

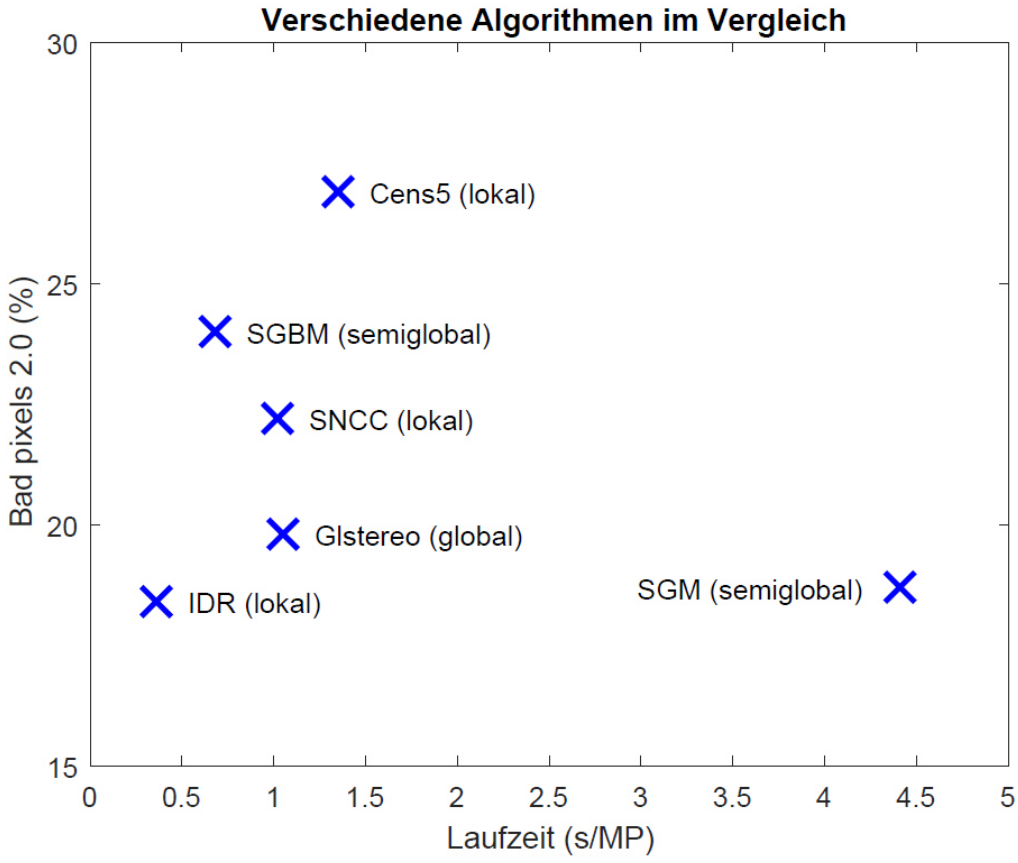

Abb. 2.3.: Vergleich verschiedener Stereoalgorithmen

Im Diagramm 2.3 ist die Standard-Qualitätsmetrik der Middlebury-Vergleichsseite *Bad Pixels 2.0*, also die Anzahl der Pixel in % mit einem Disparitätsfehler von 2.0 oder größer, gegenüber der Laufzeit in $\mathrm{s/MP}$ für die näher betrachteten

Stereoalgorithmen aufgetragen. Desweiteren sind die Algorithmen *Cens5* und *Glstereo* zum Vergleich aufgetragen. Der globale Algorithmus MSE ist nicht ansatzweise echtzeitfähig, weshalb er nicht die Anforderungen an den gesuchten Algorithmus erfüllt und nicht im Diagramm eingezeichnet ist. Die beste Kombination aus hoher *Matching*-Qualität und geringer Laufzeit liefert der IDR-Algorithmus. SNCC ist diesem als zweiter betrachteter lokaler Algorithmus sowohl in der *Matching*-Genauigkeit, als auch in der Laufzeit unterlegen. Auch die semiglobalen Algorithmen SGM und SGBM sind IDR sowohl in der Laufzeit, als auch in der *Matching*-Qualität unterlegen. Da bei der Auswahl ein hoher Fokus auf Laufzeit gelegt wird, scheinen IDR und SGBM die geeignetsten Algorithmen zu sein. Während IDR die besseren Benchmarks vorweisen kann, hat SGBM den großen Vorteil, dass es öffentlich in der Programmbibliothek OpenCV implementiert ist, wodurch die Anwendung in der RACOON-Umgebung deutlich vereinfacht wird. Weiterhin betont H. Hirschmüllers Paper [2] die Robustheit des Algorithmus gegenüber Beleuchtungsänderungen und Reflexionen, was bei RACOON eine wichtige Rolle spielt, da realitätsnahe Beleuchtungen im All simuliert werden sollen. Tab. 2.3 vergleicht den SGBM-Algorithmus mit den in Abschnitt 2.1. gestellten Anforderungen.

Tab. 2.3.: Vergleich der Anforderungen mit SGBM

Anforderungen	**SGBM**
Echtzeitfähigkeit	SGBM hat eine Laufzeit von $0,68\,\mathrm{s/MP}$, was bei einer Auflösung der Eingangsbilder von 640 x 480 Pixeln zu einer Frequenz von $4,79\,\mathrm{Hz}$ führt.
Eingangsdaten	RBG-Daten können in beliebiger Auflösung verarbeitet werden.
Lichtverhältnisse	SGBM ist robust gegenüber Beleuchtungsänderungen und Reflexionen [2].
Rechenleistung	SGBM stellt keine besonderen Anforderungen an die Rechenleistung.
Verfügbarkeit	Implementierung in der öffentlich zugänglichen Programmbibliothek OpenCV.

SGBM ist öffentlich verfügbar und erfüllt alle in Abschnitt 2.1. gestellten Anforderungen, weshalb es als Algorithmus zur Implementierung in RACOON ausgewählt wird.

3. Implementierung

3.1. Anforderungen

Die Anforderungen an die Implementierung des SGBM-Algorithmus in die RACOON-Umgebung sind in Tab. 3.1 definiert:

Tab. 3.1.: Anforderungen an die Implementierung

Anforderungen	**Spezifizierung**
Einbindung in bestehendes Framework	Es besteht eine vorgegebene Framework-Architektur zur 3D-Rekonstruktion aus einem Farb- und einem Tiefenbild, in die der Algorithmus eingepasst werden muss.
Schnittstelle zur Kamera	Es muss eine Schnittstelle zur Stereokamera im Simulator geschaffen werden.

3.2. Hardware

Die verwendete Stereokamera ist vom Modelltyp *Bumblebee 2* der Firma *Point Grey*. Die Kamera hat eine Bildauflösung von $640 \times 480\,\mathrm{px}$. Der Anschluss an einen Rechner erfolgt über *FireWire*. Die maximale *Framerate* beträgt $48\,\mathrm{Hz}$. Die Kamera ist an einem translatorisch und rotatorisch beweglichen Roboterarm mit Schrauben fixiert.

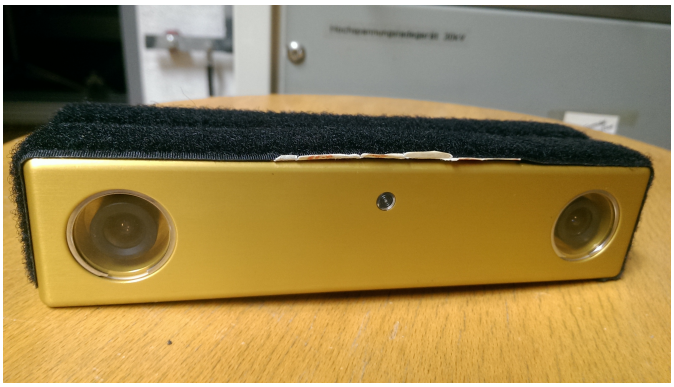

Abb. 3.1.: Bumblebee-Kamera

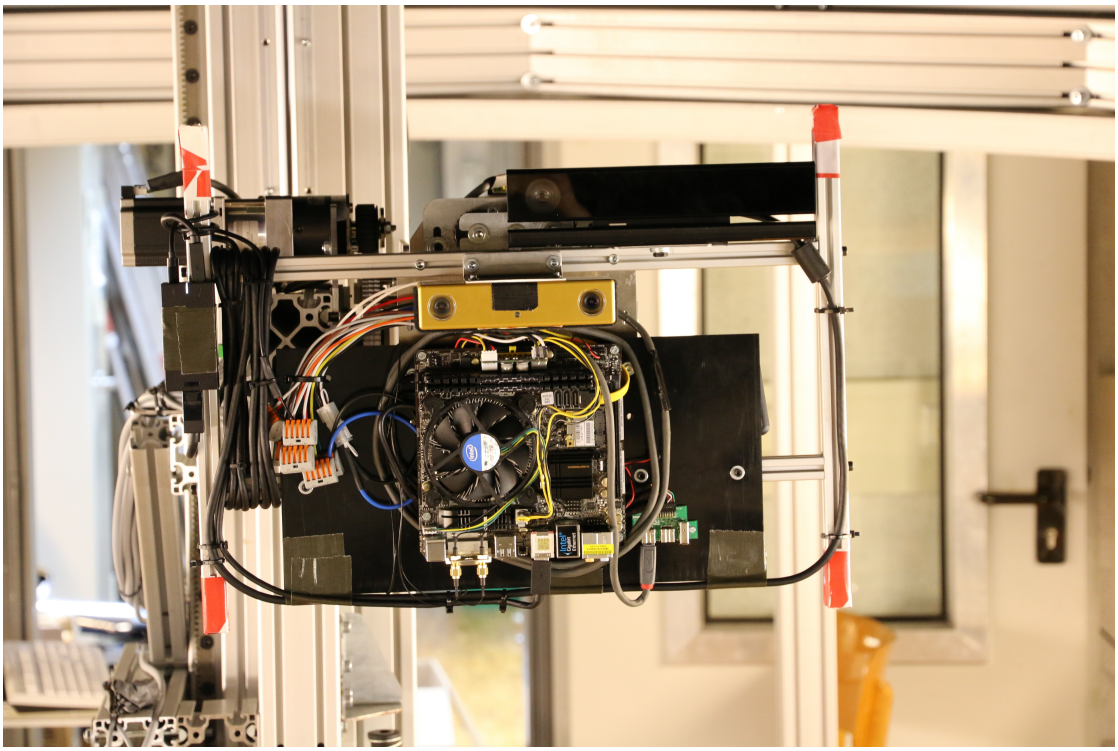

Abb. 3.2.: Vorrichtung zum Aufnehmen von Testdaten

3.3. Eingebundene Bibliotheken

Für die Implementierung ist die Einbindung von drei externen Bibliotheken notwendig. Die online frei verfügbare Bibliothek zur Bildverarbeitung OpenCV beinhaltet den SGBM-Algorithmus und ist daher zur Implementierung unverzichtbar. Die anderen beiden Bibliotheken *Flycapture2* und *Triclops* von *Point Grey* beinhalten Daten zur Weiterverarbeitung der Kamerabilder.

3.3.1. OpenCV

OpenCV [29] ist eine Bibliothek von Programmierfunktionen für maschinelles Sehen in Echtzeit, welche frei verfügbar für akademischen, als auch kommerziellen Nutzen ist. Sie liefert *Interfaces* für C++, C, Python und Java (Android). In dieser Arbeit wurde Version 3.1 verwendet.

3.3.2. FlyCapture2

FlyCapture2 ist ein *Software Development Kit* (SDK) der Firma *Point Grey*, das jeder *Point Grey*-Kamera beiliegt. Es liefert eine Software API (Programmierschnittstelle, engl. application programming interface)-Bibliothek, verschiedene

Demo-Programme und Quellcode-Beispiele zum Auslesen von Kamerabildern. In dieser Arbeit wurde Version 2.6 verwendet.

3.3.3. Triclops

Triclops ist wie *FlyCapture2* ein SDK der Firma *Point Grey*, das deren Kameras beiliegt. Im Gegensatz zu *FlyCapture2* beschäftigt sich *Triclops* mit der Weiterverarbeitung der ausgelesenen Bilder und hat bereits einige Anwendungen zum maschinellen Sehen implementiert. In dieser Arbeit wurde Version 3.4 verwendet.

3.4. Bestehende Architektur

Das bestehende *Framework* zur 3D-Rekonstruktion von Satellitenbildern wurde im Rahmen einer Semesterarbeit am LRT von Tim Wiese entwickelt. Dieses *Framework* bietet die Möglichkeit, verschiedene Algorithmen zur Lageschätzung und 3D-Rekonstruktion mit verschiedenen Sensoren zu integrieren. Die Eingangsdaten können sowohl von physischen Sensoren kommen und in Echtzeit verarbeitet werden, als auch von voraufgezeichneten Datensätzen stammen. Es gibt einen Echtzeitmodus, um die aktuelle Objektansicht „live“ darzustellen, und einen Gesamtdatenmodus, um den gesamten Datensatz zu verarbeiten und eine möglichst gute 3D-Rekonstruktion zu erzielen. Bei dem *Framework* handelt es sich um eine Windows-Anwendung mit objektorientierter Struktur, geschrieben in C++. Die Kernkomponenten des *Frameworks* sind in Abb. 3.3 aufgezeigt.

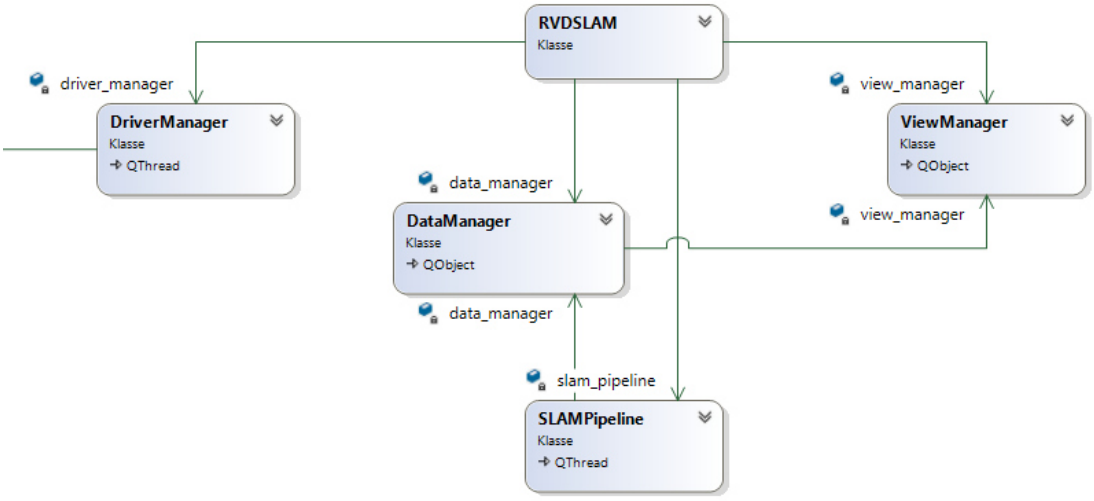

Abb. 3.3.: Klassendiagramm der Kernkomponenten (erstellt von Tim Wiese)

Das *Framework* ist in 4 größere Blöcke aufgeteilt.

- Dateneingabe
- SLAM-Pipeline
- Datenmanager
- Nutzerinterface

Der Dateineingabe-Teil umfasst die `DriverManager`- sowie die `DriverWrapper`-Klasse, die später noch genauer betrachtet wird. Dieser Teil ist für das Einlesen von Sensordaten zuständig. Die `DriverManager`-Klasse kann zwischen verschiedenen über `DriverWrapper` eingebundenen Sensoren oder aufgezeichneten Datensätzen wechseln.
Die *SLAM-Pipeline* ist für die Datenverarbeitung per eingebundener Algorithmen zuständig. SLAM steht hierbei für „Simultaneous Localization and Mapping“, also maschinelle Objekterkennung und Rekonstruktion in Echtzeit. Die Ausgabedaten der Berechnungen werden an den Daten-Manager übergeben.
Der Daten-Manager ist für die Verwaltung und Speicherung der eingelesenen und generierten Daten zuständig. Er stellt weiterhin die Schnittstelle zum Nutzerinterface dar.
Das Nutzerinterface wird vom Datenmanager mit Daten gespeist und stellt die 3D-Rekonstruktion für den Nutzer grafisch dar. Es können hier auch Softwareeinstellungen vorgenommen werden.

3.5. Codearchitektur

Im Rahmen dieser Arbeit wurde das bestehende *Framework* um die beiden Klassen `DriverWrapperBumblebee` und `FramePackageStereo` erweitert.

3.5.1. DriverWrapperBumblebee

Die `DriverWrapper`-Klassen binden die verschiedenen Sensoren in das *Framework* ein. Der *Driver Managger* kann zwischen verschiedenen *Driver Wrappern* wechseln und gibt die gesammelten Sensordaten in Form eines *Frame Package* (vgl. Abschnitt 3.5.2.) an den Datenmanager weiter. Bisher waren zwei Typen von `DriverWrapper`-Klassen vorhanden: Die Klasse `DriverWrapperKinectV1`, die für das Einlesen von *Red Green Blue Depth* (RGBD)-Daten von einem Kinect-Sensor zuständig ist und die Klasse `DriverWrapperRGBDFiles`, die einen voraufgezeichneten Datensatz von RGBD-Daten ausliest. Zum Auslesen der Stereo-Bilddaten der *Bumblebee*-Kamera wurde im Rahmen dieser Arbeit die Klasse `DriverWrapperBumblebee` als dritte `DriverWrapper`-Klasse hinzugefügt.

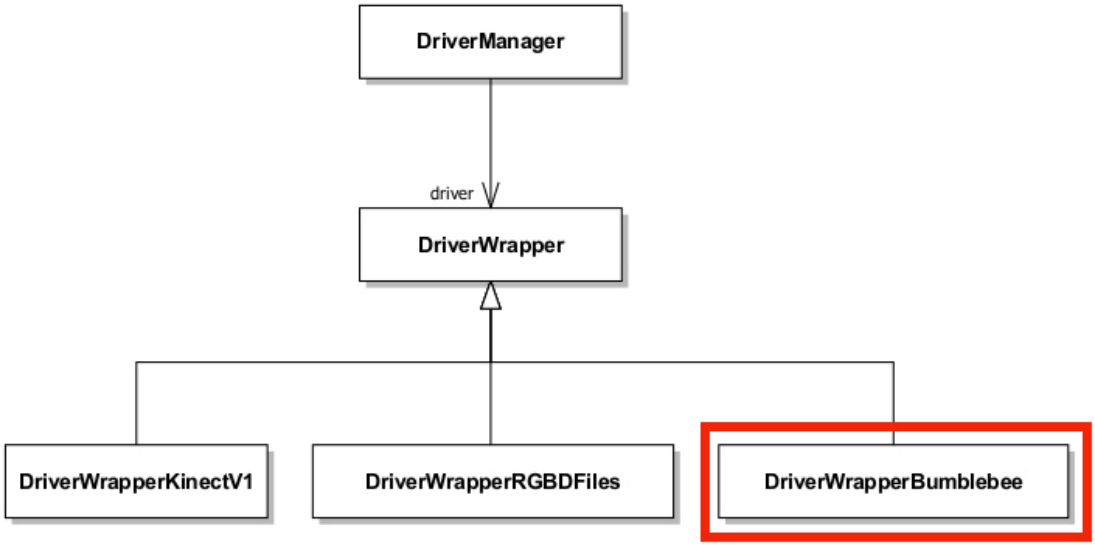

Abb. 3.4.: DriverWrapper-Klassen

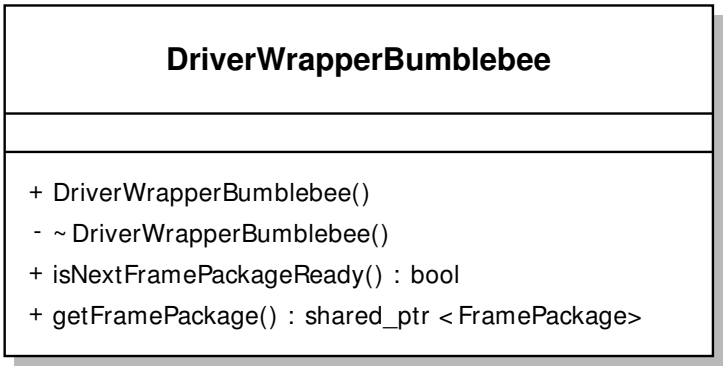

Abb. 3.5.: Methoden der Klasse `DriverWrapperBumblebee`

In der Klasse sind die beiden Methoden `isNextFramePackageReady()` und `getFramePackage()` implementiert, die von der Elternklasse `DriverWrapper` geerbt werden. Tab. 3.2 erläutert die Methoden genauer.

3.5.2. FramePackageStereo

Frame Packages beinhalten die zusammengefassten Sensor-Bilddaten sowie einige weitere Informationen, die zur Bildverarbeitung benötigt werden oder hilfreich sind.

Wie in Abb. 3.6 zu sehen beinhalten *Frame Packages* `CameraFrame`-Objekte. Es gibt zwei verschiedene `CameraFrame`-Klassen: `CameraFrameColor` und `CameraFrameDepth`. Die im Rahmen dieser Arbeit implementierte Klasse `FramePackageStereo` enthält zwei `CameraFrameColor`-Objekte, eines für jedes Kameraobjektiv der Stereokamera. Ein Objekt der Klasse `CameraFrameColor`

Tab. 3.2.: Nähere Betrachtung der Methoden der `DriverWrapperBumblebee`-Klasse

Methode	**Erläuterung**
`isNextFramePackageReady()`	Diese Methode ruft ein Bildpaar von der Bumblebee-Kamera ab und gibt (bei erfolgreichem Abruf) die Information, dass ein Bildpaar vorliegt, zurück.
`getFramePackage()`	Diese Methode verarbeitet das abgerufene Bildpaar zu einem `FramePackage` des Typs `FramepackageStereo` und gibt dieses in Form eines *shared pointers* zurück.

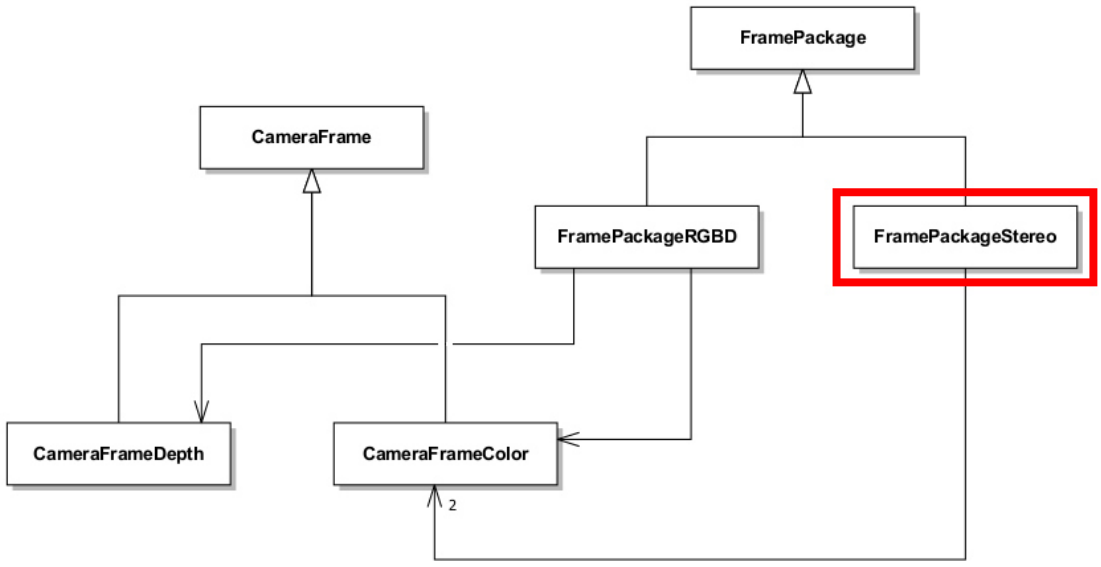

Abb. 3.6.: `FramePackage`-Klassen

beinhaltet die Daten eines Kamera-RGB-Bildes. Tab. 3.3 zeigt die Attribute eines `CameraFrameColor` auf.

Zur Weiterverarbeitung der Daten in der *SLAM-Pipeline* wird ein Farb- und ein Tiefenbild benötigt. Deshalb sind in jedem `FramePackage` die Methoden `getDepthImage()` und `getColorImage()` implementiert. Im Folgenden wird erläutert, wie diese Methoden in der Klasse `FramePackageStereo` umgesetzt sind.
`getColorImage()` gibt das linke Kamerabild zurück.
`getDepthImage()` erzeugt mit Hilfe des SGBM-Algorithmus ein Tiefenbild aus den eingegangenen Stereodaten und gibt dieses zurück. Hierbei werden sich einige in OpenCV integrierte Funktionen zunutze gemacht. Da SGBM *rectified images* benötigt, werdern die Bilder zuerst mit der Funktion `stereorectify()` bearbeitet. Anschließend wird der in OpenCV integrierte SGBM-Algorithmus auf die Bilder angewandt, der ein Disparitätsbild erstellt. Dieses wird schließ-

Tab. 3.3.: Attribute der `CameraFrameColor`-Klasse

Attribut	**Erläuterung**
`timestamp_secs`	der Zeitpunkt, zu dem das Bild aufgenommen wurde, angegeben in s
`width_px`	die Bildbreite in px
`height_px`	die Bildhöhe in px
`encoding_type`	Beschreibung, wie die Daten abgelegt sind
`intrinsics`	die intrinsischen Eigenschaften (Brennweite, Hauptpunkt) der Kamera (vgl. Abschnitt 3.6.)
`color_data`	die RGB-Farbdaten des Bildes

lich mit der Funktion `reprojectImageTo3D()` in ein Tiefenbild umgewandelt, welches ausgegeben wird. Die erforderlichen intrinsischen und extrinsischen Kameraparameter wurden vorher ermittelt (vgl. Abschnitt 3.5.) und werden im Code als konstante Werte übergeben. Innerhalb der Methode lassen sich einige Parameter ändern. So ist es z.B. möglich zwischen verschiedenen Algorithmen auszuwählen oder die Fenstergröße beim *Matching* zu ändern.

3.6. Kamerakalibrierung

Zur Stereorekonstruktion aus zwei Bildern werden die intrinsischen und extrinsischen Kameraparameter benötigt. Die intrinsischen Parameter lassen sich in einer Kameramatrix $\mathbf{P}$ darstellen. $\mathbf{P}$ beinhaltet den Hauptpunkt und die Brennweiten der Kamera. Die intrinsischen Parameter sind nicht von der betrachteten Szene abhängig und können daher beliebig oft wiederverwendet werden, solange die Brennweiten fixiert sind. Bei einer Stereokamera existieren 2 Kameramatrizen, eine für jedes Objektiv, die im Idealfall identisch sind. Die extrinsischen Parameter beschreiben die translatorische und rotatorische Bewegung der Kamera zwischen zwei Bildaufnahmen. Da für diese Arbeit eine Stereokamera mit fixierten Objektiven verwendet wurde, bleibt diese Bewegung ebenfalls stets gleich, weshalb auch die extrinsischen Parameter in unserem Fall konstant sind.

Um die intrinsischen und extrinsischen Parameter der *Bumblebee* zu ermitteln, wurde eine Kamerakalibrierung mit dem `stereo_calib`-*Sample* von OpenCV durchgeführt. Hierzu wurden Aufnahmen eines Schachbrettmusters in verschiedenen Entfernungen und Positionen aufgenommen, woraus der Kalibrierungsalgorithmus die intrinsischen und extrinsischen Kameraparameter berechnet.

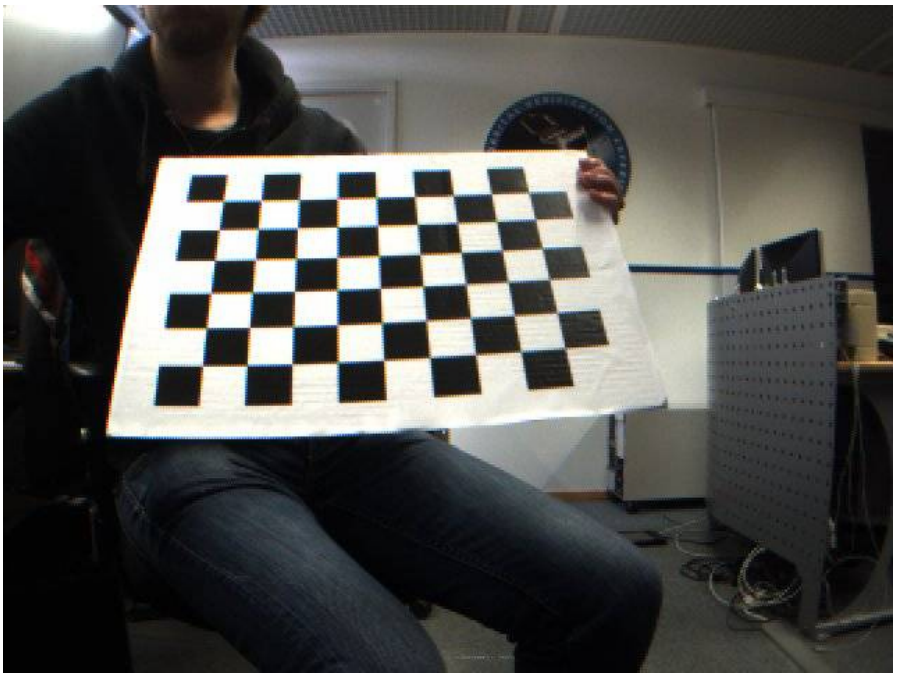

Abb. 3.7.: Kamerakalibrierung mit Schachbrettmuster

3.7. Ergebnisse

Die Anforderungen an die Implementierung waren die Einbindung des Algorithmus in das bestehende 3D-Rekonstruktions-*Framework* und die Bereitstellung einer Schnittstelle zwischen Algorithmus und Kamera. Die Einbindung des Algorithmus ist mit dem Erstellen der Klasse `FramePackageStereo`, welche den SGBM-Algorithmus zur Erstellung eines Tiefenbildes nutzt, erfolgt. Auch eine Kameraschnittstelle wurde in Form der Klasse `DriverWrapperBumblebee` geschaffen. Dieser `DriverWrapper` bindet die Kamera in das *Framework* ein und liest die benötigten Stereo-Bilddaten aus. Zum Zeitpunkt dieser Arbeit liegen noch Bugs in der Implementierung vor, die für eine sinnvolle Verwendung derselben behoben werden müssen.

4. Ergebnisse und Ausblick

4.1. Ergebnisse

Durch Betrachtung verschiedener Algorithmen und einen Vergleich mit Hilfe der *Middlebury Stereo Evalutation* [16] wurde SGBM als geeignetster Algorithmus zur Implementierung in RACOON ausgewählt. Zum Testen des Algorithmus wurden Stereo-Testbilder (vgl. Abb. (4.1)) des RACOON-Satelliten durch den Algorithmus geschickt, der daraus Disparitätsbilder erstellt. Ein Beispiel für ein entstandenes Disparitätsbild zeigt Abb. 4.2.

Abb. 4.1.: Testaufnahme eines Stereobildes

Abb. 4.2.: Disparitätsbild aus Testaufnahmen

Es lassen sich deutlich die Umrisse des Satelliten erkennen, allerdings sind noch einige Lücken vorhanden. Gerade die schlecht beleuchteten und stark reflektierenden Flächen, in diesem Fall die Solarpanele, sind nur sehr schwach besetzt. Außerdem existiert ein starkes Rauschen, das die 3D-Rekonstruktion weiter erschwert. Zum Vergleich zeigt Abb. 4.3 das entstehende Disparitätsbild des „Tsukuba“-Testbildpaares der Middlebury-Website [16] in einer Auflösung von $384 \times 288\,\mathrm{px}$. Man erkennt, dass das Rauschen deutlich geringer ist und kaum noch Lücken vorliegen. Die schlechte Beleuchtung im RACOON scheint also das Erstellen eines hochwertigen Disparitätsbildes deutlich zu erschweren.

Abb. 4.3.: Tsukuba-Testset

4.2. Ausblick

Durch den SGBM-Algorithmus lassen sich Tiefenbilder des Satellitenmodells in der RACOON-Umgebung erstellen. Diese Teifenbilder bieten einige Möglichkeiten zur weiteren Nutzung. Sonnenburg et al. stellen in ihrem Artikel [13] die Möglichkeit eines auf einem EKF basierenden SLAM-Algorithmus vor. Dieser Ansatz könnte in einer weiteren Forschungsarbeit in die RACOON-Umgebung integriert und getestet werden. Eine weitere interessante Alternative stellen Schnitzer et al. in ihrem Artikel [7] vor. Hierbei wird eine aus mehreren Tiefenbildern des betrachteten S/C erstellte Punktwolke mit dem *Random Sample Consensus* (RANSAC)-Algorithmus bearbeitet. Dieser Algorithmus dient zur Oberflächenrekonstruktion und ist laut Artikel auch bei dünn besetzten Punktwolken effektiv.

Eine Verbesserung der Qualität der Disparitätsbilder lässt sich wahrscheinlich noch durch eine genauere Kamerakalibrierung erreichen. Im Rahmen dieser Arbeit wurde zur Kalibrierung ein Schachbrettmuster auf Karton verwendet. Karton hat allerdings den Nachteil, dass er nicht völlig eben ist. Eventuell lässt sich die Kalibrierungsgüte durch ein ebeneres Schachbrettmuster noch weiter steigern.

Eine Alternative zur Stereorekonstruktion ist die Erfassung eines Farb- und Tiefenbildes über einen RGBD-Sensor. Diese Methode erzeugt ein Tiefenbild aus Infrarotdaten und ist daher nicht von Beleuchtungsbedingungen abhängig. Allerdings beträgt die Reichweite nur wenige Meter. Die 3D-Rekonstruktion mit

Hilfe eines RGBD-Sensors ist bereits in der RACOON-Umgebung implementiert und kann mit der Stereo-Rekonstruktion verglichen werden.

4.3. Zusammenfassung

Das Ziel der Arbeit war es, einen geeigneten echtzeitfähigen Algorithmus zur Stereorekonstruktion zu finden und diesen in der RACOON-Umgebung im Rahmen eines bestehenden *Frameworks* zu implementieren. Durch Einordnung moderner Stereoalgorithmen in globale, lokale und semiglobale Algorithmen und Vergleich der Vor- und Nachteile dieser Kategorien konnten globale Algorithmen aufgrund mangelnder Echtzeitfähigkeit aussortiert werden. SGBM wurde als leistungsfähiger, öffentlich verfügbarer Algorithmus zur Implementierung ausgewählt. Die Implementierung wurde durch Erweiterung des bestehenden *Frameworks* um die beiden Klassen `DriverWrapperBumblebee` und `FramePackageStereo` elegant umgesetzt.

A. Literaturverzeichnis

[1] Hartley R. and Zisserman A., Multiple View Geometry in Computer Vision. 2 ed., 2010.

[2] Hirschmüller H., "Accurate and efficient stereo processing by semi-global matching and mutual information," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005.

[3] Schnitzer F., Sonnenburg A., Janschek K., and Willich G., "Bildbasierte slam-relativnavigation und 3d rekonstruktion für das on-orbit-servicing von unbekannten und unkooperativen raumflugkörpern," Deutscher Luft- und Raumfahrtkongress 2012, 2012.

[4] Gallup D., Frahm J.-M., Mordohai P., Yang Q., and Pollefeys M., "Real-time plane-sweeping stereo with multiple sweeping directions," 2007 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8, 2007.

[5] de Wagter C., Tijmons S., Remes B., and Croon d. G., "Autonomous flight of a 20-gram flapping wing mav with a 4-gram onboard stereo vision system," 2014 IEEE International Conference on Robotics and Automation (ICRA), 2014.

[6] Krombach N., Droeschel D., and Behnke S., "Evaluation of stereo algorithms for obstacle detection with fisheye lenses," ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. II-1/W1, pp. 33–40, 2015.

[7] Schnitzer F., Janschek K., and Willich G., "Experimental results for image-based geometrical reconstruction for spacecraft rendezvous navigation with unknown and uncooperative target spacecraft," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

[8] Boykov Y., Veksler O., and Zabih R., "Fast approximate energy minimization via graph cuts," IEEE Transactions on PAMI, vol. 23, no. 11, pp. 1222–1239, 2001.

[9] Ohta Y. and Kanade T., "Stereo by intra- and inter-scanline search using dynamic programming," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-7, no. 2, pp. 139–154, 1985.

[10] Mei1 X., Sun X., Zhou Mingcai, Jiao S., Wang H., and Zhang Xiaopeng, "On building an accurate stereo matching system on graphics hardware," Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference, 2011.

[11] Felzenszwalb P. F. and Huttenlocher D. P., "Efficient belief propagation for early vision," International Journal of Computer Vision, vol. 70, no. 1, pp. 41–54, 2006.

[12] Feng Zhao, Qingming Huang, and Wen Gao, "Image matching by normalized cross-correlation," in 2006 IEEE International Conference on Acoustics Speed and Signal Processing, pp. II–729–II–732, 2006.

[13] Sonnenburg A., Tkocz M., and Janschek K., "Ekf-slam based approach for spacecraft rendezvous navigation with unknown target spacecraft," IFAC Proceedings Volumes, vol. 43, no. 15, pp. 339–344, 2010.

[14] Fleischner A., Wilde M., and Walter U., "Racoon - a hardware-in-the-loop simulation environment for teleoperated proximity operations," I-SAIRAS 2012, 2012.

[15] Kowalczuk J., Psota E. T., and Perez L. C., "Real-time stereo matching on cuda using an iterative refinement method for adaptive support-weight correspondences," IEEE Transactions on Circuits and Systems for Video Technology, vol. 23, no. 1, pp. 94–104, 2013.

[16] Scharstein D., Szeliski R., and Hirschmüller H., "Middlebury stereo evaluation: Version 3." `http://vision.middlebury.edu/stereo`.

[17] Unger C. and Navab N., "Stereo matching." `http://campar.in.tum.de/twiki/pub/Chair/TeachingWs10Cv2/3D_CV2_WS_2010_StereoMatching.pdf`.

[18] Kuk-Jin Yoon and In-So Kweon, "Locally adaptive support-weight approach for visual correspondence search," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), pp. 924–931, 2013.

[19] Fusiello A., Roberto V., and Emanuele T., "Efficient stereo with multiple windowing - computer vision and pattern recognition," 997. Proceedings: 1997 IEEE Computer Society Conference, pp. 858–863, 1997.

[20] Kang S. B., Szeliski R., and Chai J., "Handling occlusions in dense multi-view stereo," Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference, pp. I103–I110, 2001.

[21] Einecke N. and Eggert J., "A two-stage correlation method for stereoscopic depth estimation," Digital Image Computing: Techniques and Applications (DICTA), 2010 International Conference, pp. 227–234, 2010.

[22] Kim J., Kolmogorov V., and Zabih R., Visual Correspondence Using Energy Minimization and Mutual Information. PhD thesis, Cornell University, Ithaca, NY 14853, 2003.

[23] Chi Zhang, Zhiwei Li, Yanhua Cheng, Rui Cai, Hongyang Chao, Yong Rui, "Meshstereo: A global stereo model with mesh alignment regularization for view interpolation," ICCV, 2015.

[24] Žbontar J. and LeCun Y., "Stereo matching by training a convolutional neural network to compare image patches," 2016.

[25] Hirschmüller H., "Semi-global matching – motivation, developments and applications," Photogrammetric, vol. 11, 2011.

[26] Egnal G., Mutual Information as a Stereo Correspondence Measure. Technical reports (cis), University of Pennsylvania, Pennsylvania, 2000.

[27] `http://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#id5`.

[28] Birchfield S. and Tomas C., "A pixel dissimilarity measure that is insensitive to image sampling," IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, vol. 20, no. 4, 1998.

[29] `http://opencv.org`.