Fast Dense Depth Estimation from UAV-borne Aerial Imagery for the Assistance of Emergency Forces

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für
Bauingenieur-, Geo- und Umweltwissenschaften
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Boitumelo Ruf, M.Sc.
geb. in Palapye

Tag der mündlichen Prüfung: 26.01.2022

Hauptreferent: Prof. Dr.-Ing. Stefan Hinz
Institut für Photogrammetrie und Fernerkundung (IPF)
Karlsruher Institut für Technologie (KIT)

Korreferent: Assoc. Prof. Dr. Francesco Nex
Faculty of Geo-Information Science and Earth Observation (ITC)
Department of Earth Observation Science (EOS)
University of Twente

To Immanuel and my family

Abstract

This work addresses the use of commercial off-the-shelf (COTS) rotor-based unmanned aerial vehicles (UAVs) to facilitate emergency forces in the rapid structural assessment of a disaster site, by means of aerial image-based reconnaissance. It proposes a framework that consists of two parts and relies on the integrated stereo vision sensor and the visual payload camera of the UAV to execute three high-level applications that aim at facilitating first responders in disaster relief missions. The first part aims at (i) increasing the autonomous and safe use of the UAV, by relying on the image data from the integrated stereo vision sensor to perform real-time obstacle detection and collision avoidance. For this purpose, this work presents an approach for real-time and low-latency disparity estimation, which is optimized for the execution on an on-board embedded device and computes disparity maps that depict the environment in the flight-path of the UAV. These disparity maps, in turn, serve as input to a subsequent algorithm for real-time obstacle detection and reactive collision avoidance. The second part of the presented framework addresses (ii) fast 3D mapping and 3D change detection of the disaster site, based on image data captured by the visual payload camera of the UAV. In this, an approach for fast and dense multi-view stereo depth estimation from multiple aerial images captured by a single moving camera is presented. In addition, this work also presents an approach for single-view depth estimation from a single aerial image. While the former approach is based on conventional methods for dense image matching, the latter one is a data-driven approach relying on deep learning. The estimation of depth data from a single input image allows to also consider scenes that contain dynamic objects, such as moving cars. An important assumption in the second part of the framework is that during the flight of the UAV the image data from the main camera is transmitted to a ground control station, which is equipped with more hardware resources. However, details of the data-link are not covered by this work.

For the two-view disparity estimation from image data of the stereo vision sensor, a well-known and proven approach from literature is optimized for real-time processing on an embedded system equipped with a CPU and a GPU. Embedded systems with a built-in GPU are increasingly deployed on COTS UAVs, since they are more versatile than FPGA-based systems and better suited for running artificial intelligence (AI) applications. For the task of fast 3D mapping, an approach is presented that uses a hierarchical processing pipeline to compute dense depth maps from three or more images. The hierarchical processing scheme, as well as the improved method for surface-aware regularization, allows to efficiently and accurately reconstruct slanted surfaces structures which are particularly prominent in oblique aerial imagery. The third approach presented in this work uses a deep-learning-based approach to estimate a depth map from a single aerial image. In doing so, it is trained in a self-supervised manner, which does not require any special training data and thereby increases its flexibility with respect to its practical use.

In this work, the three methodological approaches for two-view, multi-view and single-view depth estimation are presented in detail and thoroughly evaluated. Their performance with respect to the considered use-cases is extensively discussed and demonstrated. This work shows (i) that the presented approach for real-time two-view disparity estimation on embedded devices achieves state-of-the-art results and is, at the same time, well-suited for real-time and low-latency obstacle detection and reactive collision avoidance on board the UAV. (ii) Furthermore, it is shown that the methodological extensions within the approach for multi-view depth

estimation allow for both fast and accurate reconstruction of the depicted scene. (iii) The experiments and investigations conducted with respect to the approach for single-view depth estimation by means of deep learning show, that even though it is not yet suited for a stand-alone use, it is well-suited for a complementary use in addition to approaches that perform depth estimation by means of conventional dense image matching. (iv) In conclusion, both the disparity maps computed from the stereo image pair, as well as the multi-view depth maps are well-suited for the considered use-cases, namely real-time obstacle detection and collision avoidance as well as 3D mapping and 3D change detection, respectively.

Kurzfassung

Diese Arbeit befasst sich mit dem Einsatz von handelsüblichen, drehflügel-basierten UAVs zur Unterstützung von zivilen Einsatzkräften bei der Lageerfassung und Übersichtsgewinnung an Einsatzorten. Besonders Ersthelfern kann ihre Arbeit durch eine rasche Bewertung der Lage am Einsatzort mit Hilfe von Luftbildern, die von handelsüblichen UAV erfasst werden, erleichtert werden. Dazu stellt diese Arbeit ein zweiteiliges Rahmenwerk vor. Mit dem ersten Teil des Rahmenwerkes soll (i) die Autonomie und Sicherheit des UAVs, durch einen Ansatz zur automatischen Hinderniserkennung und Kollisionsvermeidung anhand von Bilddaten einer Stereokamera, erhöht werden. Hierfür wird im Rahmen dieser Arbeit ein Verfahren zur Disparitätsschätzung anhand von Stereo-Bildpaaren vorgestellt, welches in Echtzeit auf einem eingebetteten System an Bord des UAVs ausgeführt werden kann. Dies nutzt die Bilddaten der im UAV eingebauten Stereokamera, um Disparitätskarten zu berechnen, die das Umfeld in Flugrichtung des UAVs abbilden und welche als Eingabe für ein nachgelagerten Algorithmus zur Hinderniserkennung und reaktiven Kollisionsvermeidung dienen. Der zweite Teil des vorgestellten Rahmenwerkes befasst sich mit der (ii) schnellen und echtzeitnahen 3D-Kartierung des Einsatzgebietes, sowie mit einer schnellen 3D-Änderungsdetektion, auf Basis der Bilddaten, welche von der Hauptkamera des UAVs erfasst werden. Hierfür wird zum einen ein Verfahren zur schnellen und dichten Tiefenschätzung aus mehreren Luftbildern, welche von einer sich um die Szene bewegenden Kamera erfasst werden, vorgestellt. Des Weiteren wird ein Verfahren für die Tiefenschätzung aus einem einzelnen Luftbild vorgestellt. Während Ersteres auf konventionelle Methoden des Bildmatchings beruht, verfolgt das zweite Verfahren einen datengetriebenen und lernbasierten Ansatz. Die Schätzung von Tiefendaten aus nur einem Bild erlaubt die Berücksichtigung von Szenen, die dynamische Objekte, wie bspw. fahrende Autos, enthält. Eine wichtige Annahme im zweiten Teil des Rahmenwerkes ist, dass die Bilddaten der Hauptkamera während des Fluges an eine Bodenkontrollstation, die mit mehr Hardwareressourcen ausgestattet ist, übermittelt werden. Details zur Datenverbindung sind aber nicht Gegenstand dieser Arbeit.

Für die Disparitätsschätzung aus Stereobildpaaren wird ein, in der Literatur bekanntes und bewährtes Verfahren für die Echtzeitverarbeitung auf einem eingebetteten System, welches mit einer CPU und einer GPU ausgestattet ist, optimiert. Eingebettete Systeme mit einer eingebauten GPU werden zunehmend auf handelsüblichen Drohnen verbaut, da sie gegenüber FPGA-basierten Systemen vielseitiger einsetzbar und für die Ausführung von KI-Verfahren besser geeignet sind. Für die schnelle 3D-Kartierung wird ein Verfahren vorgestellt, welches anhand einer hierarchisch aufgebauten Verarbeitungskette Tiefenkarten aus drei oder mehreren Bildern berechnet. Der hierarchische Ansatz, sowie eine verbesserte Regularisierung, welche auch Oberflächenstrukturen berücksichtig, erlaubt eine effiziente und genaue Rekonstruktion von geneigten Oberflächen, welche besonders in Luftbildern mit Schrägsicht dominant sind. Das dritte, in dieser Arbeit vorgestellte Verfahren, nutzt einen lernbasierten Ansatz zur Schätzung einer Tiefenkarte aus einem einzelnen Luftbild. Dabei wird es selbst-überwacht trainiert, was keine speziellen Trainingsdaten erfordert und damit die Flexibilität in der Anwendung des Verfahrens erhöht.

Die drei, in dieser Arbeit vorgestellten methodischen Verfahren zur Tiefenschätzung im Hinblick auf den Zwei-, Mehr- und Einzelbildfall, werden detailliert beschrieben und auf Basis umfangreicher Untersuchungen hinsichtlich ihrer Stärken und Schwächen beleuchtet. Dabei werden insbesondere ihre Leistungsfähigkeiten im

Zusammenhang der betrachteten Anwendungsfälle demonstriert und herausgearbeitet. Darin zeigt sich, (i) dass das vorgestellt Verfahren für die Echtzeit-Disparitätsschätzung aus Stereobildpaaren auf eingebetteten Systemen state-of-the-art Ergebnisse erzielt und gleichzeitig sehr gut für eine Hinderniserkennung und reaktive Kollisionsvermeidung an Bord des UAVs in Echtzeit geeignet ist. (ii) Des Weiteren wird gezeigt, dass die methodischen Erweiterungen innerhalb des Verfahrens für die Mehrbild-Tiefenschätzung sowohl eine schnelle als auch genau Erfassung der Geometrie erlauben. (iii) Die Untersuchungen zur Tiefenschätzung aus einem einzelnen Luftbild mittels eines lernbasierten Verfahrens zeigen, dass ein solches Verfahren zwar noch nicht für den alleinigen Betrieb geeignet ist, sich aber gut für den komplementären Gebrauch, zusätzlich zu den Methoden, die auf konventionelles Bildmatching beruhen, eignet. Da hierbei die Stärken beider Modalitäten kombiniert werden können. (iv) Abschließend zeigt sich, dass sich sowohl die Disparitätskarten, die anhand des Stereobildpaares berechnet werden, sowie die Tiefenkarten des Verfahrens für die Mehrbild-Tiefenschätzung, gut für die betrachteten Anwendungsfälle, nämlich Hinderniserkennung und Kollisionsvermeidung bzw. 3D-Kartierung und 3D-Änderungsdetektion, eignen.

Acknowledgements

Over the course of my Ph.D. journey, I have met many people, some of whom have gone with me for only a short period of time, others accompanied me from the beginning until now. I am very grateful for all the inspiring encounters and for all the support I have experienced along this journey, making this work and this thesis what it is today. I cannot acknowledge every person that as accompanied me on this journey, but I want to mention and express my gratitude to the most significant ones.

First and foremost, I would like to thank Prof. Dr.-Ing. Stefan Hinz from the Institute of Photogrammetry and Remote Sensing (IPF) at the Karlsruhe Institute of Technology (KIT) for accepting to supervise this work from the beginning and for the support throughout my time as a Ph.D. student. Thank you for always taking time out from your busy schedule as head of IPF to discuss my ideas and problems. I greatly treasure the ideas and feedback you gave throughout the years.

I would also like to thank my direct supervisor Norbert Heinze as well as Dr.-Ing. Markus Müller as head of the department of Video Exploitation Systems (VID) at the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (IOSB) in Karlsruhe, as this dissertation was conducted in close cooperation with the department VID at the Fraunhofer IOSB. Thank you for providing me with the opportunity to pursue my Ph.D. and for giving me the freedom to pursue my research interests.

Moreover, I would like to express my sincere gratitude to Martin Weinmann for his continuous guidance, the numerous valuable discussions as well as the comments and ideas he provided throughout my time as a Ph.D. student. I greatly appreciate your availability, the time you invested in this work, and the knowledge you shared with me.

Next, I want to thank Assoc. Prof. Dr. Francesco Nex for being my co-referee, despite his heavy workload. And I thank Boris Jutzi, Prof. Dr.-Ing. Rainer Stiefelhagen, Prof. Dr.-Ing. Uwe Sörgel and Prof. Dr.-Ing. Hansjörg Kutterer for being part of the examination board and for investing their time to review and examine this work.

I am also very grateful for all my co-authors who contributed to this work and to the publications which make up the core of this thesis. These co-authors are Martin Weinmann, Max Hermann, Prof. Dr.-Ing. Stefan Hinz, Prof. Dr.-Ing. Jürgen Beyerer, Thomas Pollok, Bastian Erdnüß, Jonas Mohrs, Laurenz Thiel, Sylvia Schmitz, Sebastian Monka, Michael Grinberg and Tobias Schuchert. I also thank Markus Hillemann and Markus Müller for their work in capturing the data of the TMB dataset.

Furthermore, I also greatly thank all my colleagues, ex-colleagues, and friends at the Fraunhofer IOSB and the IPF. Thank you for all the discussions and events we had together, your companionship throughout the years, as well as the milestones we have reached together. In this regard I particularly thank Thomas Pollok, Max Hermann, Raphael Senk, Thomas Golda, Aleksej Buller, Bastian Erdnüß, Lucas Deutschmann, Lucas Moser, Svea Krikau and Shreyas Manjunath from the Fraunhofer IOSB as well as Martin Weinmann, Markus Hillemann, Markus Müller, Boris Jutzi, Rebecca Ilehag and Jens Kern from the IPF. I also thank Tobias Schuchert for prompting me to pursue my Ph.D. and for his guidance and support during the first years of this journey.

Finally, I am immensely grateful to my family – my parents Ingrid and Max Ruf, my sisters Yvonne Ruf and Verena Rahn together with their families, my parents-in-law Andrea and Arnim Eglauer, and my sister-in-law Anja Kutter together with her family – for all their support. Last but not least, I am gratefully indebted to my wife Eva Ruf. Thank you, Eva, for your unlimited support and the love you have given me throughout all these years!

Boitumelo Ruf
Karlsruhe, May 2022

Contents

Terminology

This chapter includes material from the following publication:

Ruf, B.; Weinmann, M., and Hinz, S. (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821*. Reprinted with permission. It is cited as (Ruf et al. 2021a) and marked with a green sidebar.

This chapter introduces important terminologies which are used in this work in order to resolve ambiguous definitions and interpretations.

Commercial Off-the-Shelf Unmanned Aerial Vehicles

Remotely controlled aircrafts can be grouped into a number of different categories, such as fixed-wing or rotor-based aircrafts, high-altitude-long-endurance or medium-altitude-long-endurance aircrafts, those that can be bought off-the-shelf by anyone or those that are uniquely manufactured and flown by experts. This work focuses on the use of *commercial off-the-shelf (COTS) unmanned aerial vehicles (UAVs)*, meaning that the UAVs can be bought and flown off-the-shelf by anyone, without the need of special training. Only an appropriate license might be required, depending on the responsible aviation agency, e.g. the European Aviation Safety Agency (EASA). While this definition is not restricted to a single type of UAV, the most commonly available and most easy to use, are the rotor-based UAVs. Thus, in the scope of this work, unless explicitly specified, the terminology UAV stands for rotor-based remotely controlled aircrafts. Lastly, it is deliberately refrained from categorizing the considered UAVs by their costs, since depending on the application or use-case the category "low-cost" is defined differently.

Dense Image Matching and Depth Estimation

From two or more images that depict the same scene from different vantage points, the 3D scene structure and, in turn, the scene depth with respect to a selected reference view can be calculated. Here, the scene depth, perceived from a certain viewpoint with defined characteristics, e.g. field-of-view (FOV) and image resolution, is typically stored inside a 2D *depth map*, holding the pixel-wise orthogonal distance of the depicted object from the optical center of the camera. The process of calculating these depth maps is denoted as *depth estimation*. When performing depth estimation from two or more images, a dense correspondence field between the pixels of each considered image needs to be established. This is usually done by *dense image matching (DIM)*, which, in contrast to the matching of point features, uses direct pixel matching to find a correspondence for all pixels in the image.

Monocular and Single-Image

In literature, the two terminologies of "monocular" and "single-image" are often mixed and not used in a consistent manner. In the scope of this work, the term *monocular* is used when referring to the use of only one

camera. For example, this is the case for monocular multi-view depth estimation, where the scene depth is calculated based on two or more images captured by one camera from different viewpoints. Evidently, the capturing of images from different viewpoints by one camera can only be done at different moments in time. In contrast, the term *single-image* or *single-view* is used, when truly only one image is considered for depth estimation or other tasks.

On-Board and Off-Board Processing

On-board processing refers to the use of specialized and embedded hardware, that allows to execute algorithms directly on board the sensor platform. The use of the term "on-board" does not imply any details on the run-time of the algorithm. On-board processing can be done with or without time constraints. To denote the execution of algorithms on hardware that is not directly connected to the sensor platform or the UAV, the term *off-board* is used. This, for example, applies for the execution of algorithms on a ground control station (GCS).

Real-Time, Online and Offline Processing

The run-time requirement for an algorithm, and whether it is denoted as real-time or not, greatly depends on the rate of the input data and the actual application in which the algorithm is used. Thus, in literature, the use of the word "real-time" is not consistent and does not always mean the same thing. In the scope of this work, *real-time processing* is used to denote a fast and low-latency processing. This means that the processing rate of the algorithm is high enough, in order to allow an immediate reaction based on the calculated results. For example, in the case of reactive collision avoidance based on depth data from a stereo camera, the calculation of the scene depth needs to be fast enough to still be able to initiate an appropriate maneuver to avoid imminent collision. Real-time processing is deliberately not defined by a minimum processing rate, since the available reaction time depends on various factors, such as flight speed. In contrast, *online processing* is used to denote a fast processing of the algorithm without setting hard time constraints. Ideally, an algorithm for online processing should be able to keep up with the frame rate of the input data. At the same time, however, it is not critical if the algorithm has a high latency and if the results are only available a couple of frames after the input of corresponding reference frame. This, for example, is the case when performing depth estimation from two or more images that are captured by a single moving camera, i.e. monocular multi-view stereo. Here, enough input images with appropriate baseline have first to be collected before the actual processing can be started. Both real-time and online processing refer to a computation during the acquisition or receiving of the input data. Thus, corresponding algorithms only have a confined set of input data available during execution. *Offline* processing, on the other hand, is done fully disconnected from the actual acquisition of the input data and, thus, it is assumed that algorithms, which are executed offline, have access to all available input data. (Ruf et al. 2021a, Sec. 1.)

Structure-from-Motion, Multi-View Stereo and Simultaneous Localization and Mapping

In accordance with the definition of Schönberger et al. (2016), *structure-from-motion (SFM)* denotes the sparse modeling and camera pose computation from image data based on detecting and matching discriminative image features. *Multi-view stereo (MVS)*, on the other hand, names the process of dense modeling from image data with corresponding camera poses by means of dense image matching and dense depth estimation. Evidently, as instantiated by common software suites for offline photogrammetric 3D reconstruction, such as COLMAP (Schönberger and Frahm 2016, Schönberger et al. 2016), SFM is executed prior to MVS. *Simultaneous localization*

and mapping (SLAM) algorithms are considered as a special form of SFM, as they are intended for online and even real-time processing, whereas generic SFM does not have any implication on run-time or the availability of input data.

Notation

This chapter introduces the notation and symbols which are used in this work.

General Notation

Scalars	italic Roman and Greek lowercase letters	x, α
Sets	calligraphic uppercase letters and Greek uppercase letters	$\mathcal{M}, \Theta$
2D Points/Vectors	Roman lowercase letters	m
3D Points/Vectors	Roman uppercase letters	M
Matrices	bold Roman uppercase letters	$\mathbf{R}$
Functions	italic Roman uppercase letters	T

Dense Image Matching

b	stereo baseline, given in a metric scale
$\mathrm{C} \in \mathbb{R}^3$	3D extrinsic translation vector, denoting the location of the camera center with respect to a reference coordinate system
$C(\cdot)$	cost function to quantify the similarity between two pixels
$CT(\cdot)$	census transform
δ	orthogonal distance of the scene plane Π from the optical center of the camera
d	depth, i.e. orthogonal distance of a scene point from the optical center of the camera
$\mathcal{D}$	2D depth map
$\hat{f}$	actual focal length of the camera optics, given in a metric scale
$f_u = \frac{\hat{f}}{h_u}, f_v = \frac{\hat{f}}{h_v}$	camera constant, i.e. mathematical focal length of the pinhole camera model, in u- and v-direction
$\mathbf{F}$	fundamental matrix between two calibrated stereo images
$\Gamma_u = [\Delta u_{\min}, \Delta u_{\max}]$	disparity range in which a dense image matching is performed
$\Gamma_d = [d_{\min}, d_{\max}]$	depth range used to sample the scene-space on the highest pyramid level
h_u, h_v	actual size of a sensor element, i.e. pixel, in u- and v-direction, given in a metric scale
$\mathbf{H}(\Pi, \mathbf{P}_{\text{ref}}, \mathbf{P}_k)$	homography induced by the scene plane Π between a reference camera with projection matrix $\mathbf{P}_{\text{ref}}$ and a neighboring camera with projection matrix $\mathbf{P}_k$

$\mathcal{I}_\mathrm{L}, \mathcal{I}_\mathrm{R}$	images of the left and right camera of a stereo camera rig
$\mathcal{I}_k$	input image k within the multi-image setup
$\mathcal{I}_\mathrm{ref}$	reference image within the multi-image setup
$\mathbf{K}$	intrinsic camera calibration matrix
$\mathrm{l_m}$	epipolar line corresponding to the image point m
m	projection of the scene point M in image $\mathcal{I}$
M	scene point in the camera coordinate system
N_Π	3D normal vector of the scene plane Π
$NCC(\cdot)$	normalized cross correlation
$\mathcal{P}$	image patch around the pixel p
$\Pi = (\mathrm{N}_\Pi, \delta)$	scene plane with the normal vector N_Π and located at the orthogonal distance δ from the camera
$\mathrm{p}^\mathrm{L}, \mathrm{p}^\mathrm{R}$	pixel in the reference image $\mathcal{I}_\mathrm{L}$ and the matching image $\mathcal{I}_\mathrm{R}$ respectively
$\mathrm{p}^\mathrm{ref}, \mathrm{p}^k$	pixel in the reference image $\mathcal{I}_\mathrm{ref}$ and the matching image $\mathcal{I}_k$ respectively
$\mathbf{P} = \mathbf{K}\,[\mathbf{R}^\intercal \text{ -}\mathbf{R}^\intercal \mathrm{C}]$	full camera projection matrix, made up of camera calibration matrix $\mathbf{K}$, extrinsic rotation $\mathbf{R}$ and translation C
$\mathbf{R} \in SO(3)$	extrinsic rotation matrix, with the row vectors holding the normalized coordinate axes of the camera coordinate system with respect to the reference coordinate system
$\mathcal{S}$	three-dimensional cost volume, holding the matching score s for each pixel p and the disparity $\Delta u \in \Gamma_u$ or the plane Π, depending on the stereo setup used
Δu	disparity, i.e. horizontal displacement in the image
$\mathcal{U}$	2D disparity map
u, v	2D image coordinates

Semi-Global Matching

E	energy function of the Markov random field
V	smoothness constraint within E
L_r	1D aggregation path, as part of the dynamic programming to approximate the solution of E
r	direction of the aggregation path L
φ_1, φ_2	smoothness penalties, penalizing deviations in the disparity between neighboring pixels
$\bar{\mathcal{S}}$	optimized cost volume, holding the aggregated matching score s for each pixel p and disparity $\Delta u \in \Gamma_u$ after path aggregation

Geometric Consistency Filtering

ϵ_h	hit count of reprojections, for which $\epsilon_\mathrm{r} < \eta_\mathrm{r}$, within the scope of a multi-view geometric consistency check
ϵ_r	reprojection error within the scope of a multi-view geometric consistency check
η_h	threshold for consistent hits within the scope of a multi-view geometric consistency check
η_r	threshold for the reprojection error within the scope of a multi-view geometric consistency check

Fast Multi-View Stereo with Surface-Aware Semi-Global Matching

$\mathcal{C}$	estimated confidence map corresponding to $\mathcal{I}_\mathrm{ref} \in \Omega$, holding confidence values with respect to the estimates of $\mathcal{D}$ and $\mathcal{N}$
$\mathrm{e}_\mathrm{ref}^k$	epipole of the reference camera, i.e. projection of C_ref in the image $\mathcal{I}_k$
Δi_pg	discrete index jump through a set of sampling planes to adjust zero-cost transition within $V_{\Pi\text{-pg}}$
Δi_sn	discrete index jump through a set of sampling planes to adjust zero-cost transition within $V_{\Pi\text{-sn}}$
$I(\cdot)$	indexing function to return the index of a certain plane Π within the set of sampling planes
l	pyramid level of the Gaussian image pyramid within the hierarchical processing scheme
n	height of the Gaussian image pyramid
$\mathcal{N}$	estimated normal map corresponding to $\mathcal{I}_\mathrm{ref} \in \Omega$
Ω	input bundle, consisting of input images $\mathcal{I}_k$ and projection matrices $\mathbf{P}_k$
$\Pi_\mathrm{min}, \Pi_\mathrm{max}$	scene planes at minimum and maximum distance, bounding sampling space
$Q(\cdot)$	perspective-invariant cross-ratio
$\nabla \mathrm{r}$	running gradient of the minimum cost path along the aggregation path r
$s^\mathrm{L}, s^\mathrm{R}$	matching cost computed by C and aggregated for left and right subset of matching images
V_m	ray casted from C through the 2D image point m
V_Π	plane-wise smoothness constraint, adopting the E to depth hypotheses produced by plane-sweep multi-image matching
$V_{\Pi\text{-sn}}$	surface normal smoothness constraint, extending V_Π with an adjusted zero-cost transition based on the surface normal in order to account for non-fronto-parallel surface structures

$V_{\Pi\text{-pg}}$	path gradient smoothness constraint, extending V_{Π} with an adjusted zero-cost transition based on the gradient of the minimum cost path in order to account for non-fronto-parallel surface structures

Self-Supervised Single-View Depth Estimation

$\mathbf{E}_{\text{ref}\to k} = [\mathbf{R}\ \mathrm{T}]$	relative extrinsic transformation between the reference image $\mathcal{I}_{\text{ref}}$ and the neighboring image $\mathcal{I}_k$
L	loss function
P	projective transformation, projecting the matching image $\mathcal{I}_k$ into the view of the reference image $\mathcal{I}_{\text{ref}}$, given $\mathbf{K}$, $\mathcal{D}$ and $\mathbf{E}_{\text{ref}\to k}$
s	image reconstruction error
$SSIM(\cdot)$	structural similarity score

1 Introduction

Even though remotely controlled aircrafts have been around for quite a long time now, until recently they were niche products, mostly used by military or by model aircraft enthusiasts. However, in the last decade, the availability of *commercial off-the-shelf (COTS)* rotor-based *unmanned aerial vehicles (UAVs)*, that can be purchased by anyone and are easy to fly, has increased tremendously. According to a recent report by Markets-and-Markets (2021), it is estimated that the global UAV market, including all different sectors, is to reach 58.4 billion USD by 2026, which is a compound annual growth rate (CAGR) of 16.4 % compared to global market in 2021. In this, with a relative CAGR of 28 %, the commercial segment is expected to have the highest growth rate in the forecast period, which is attributed to the advancements of COTS UAVs (Markets-and-Markets 2021). In recent years, these developments and advancements have led to an increasing use of COTS UAVs to facilitate a wide range of commercial applications, such as aerial video and photography, civil construction and industrial inspection (Chen et al. 2014, Sankarasrinivasan et al. 2015), precision agriculture (Perz and Wronowski 2019, Tsouros et al. 2019), security monitoring (Finn and Wright 2012), disaster relief (Estrada and Ndoma 2019, Restas 2015, Kerle et al. 2020, Furutani and Minami 2021), as well as search-and-rescue (SAR) missions (Silvagni et al. 2017, Doherty and Rudol 2007). This work focuses on the use of COTS UAVs by emergency forces in order to facilitate disaster relief as well as SAR missions. In the following, a short review on the ability and the use of modern COTS UAVs to execute specific tasks that assist emergency forces in their mission is provided (Section 1.1), before giving an overview on the proposed framework and the structure of this work (Section 1.2).

1.1 The Use of Commercial Off-the-Shelf UAVs for Disaster Relief and Search-and-Rescue Missions

The commercial UAV market offers a great variety of different UAVs, ranging from smaller low-cost models, which are primarily intended for the private use, up to high-end models for professional use. The latter ones allow for a customization with a number of different sensors and equipment, meeting the special needs of the application and task that is to be executed. As illustrated by Figure 1.1, the DJI Matrice 210v2 RTK, for example, allows the use of different types of sensors which, in turn, are designed to execute a variety of tasks for facilitating disaster relief and SAR missions. For one, it is equipped with localization sensors like a global navigation satellite system (GNSS) and an inertial measurement unit (IMU). These provide a precise positioning and localization of the UAV, needed for navigation and execution of a planned flight path as well as providing a geographical position in case of finding missing persons. Moreover, in order to ensure safety during flight, different close vicinity sensors like a stereo vision camera and ultrasonic sensor are deployed on-board modern UAVs. In case of autonomous flight, which is especially important if the UAV is to be used beyond visual line-of-sight (BVLOS), these close vicinity sensors provide the opportunity to perform an obstacle detection and collision avoidance. In this, a direct on-board high-performance processing is crucial, which is why UAVs from the DJI Matrice series can be equipped with an on-board processing unit (PU), e.g. the DJI Manifold 2-G. Due to the increasing demand for deep-learning-based artificial intelligence (AI), most of these on-board PUs of modern UAVs are equipped with graphic processing units (GPUs), just as the DJI Manifold 2-G which is

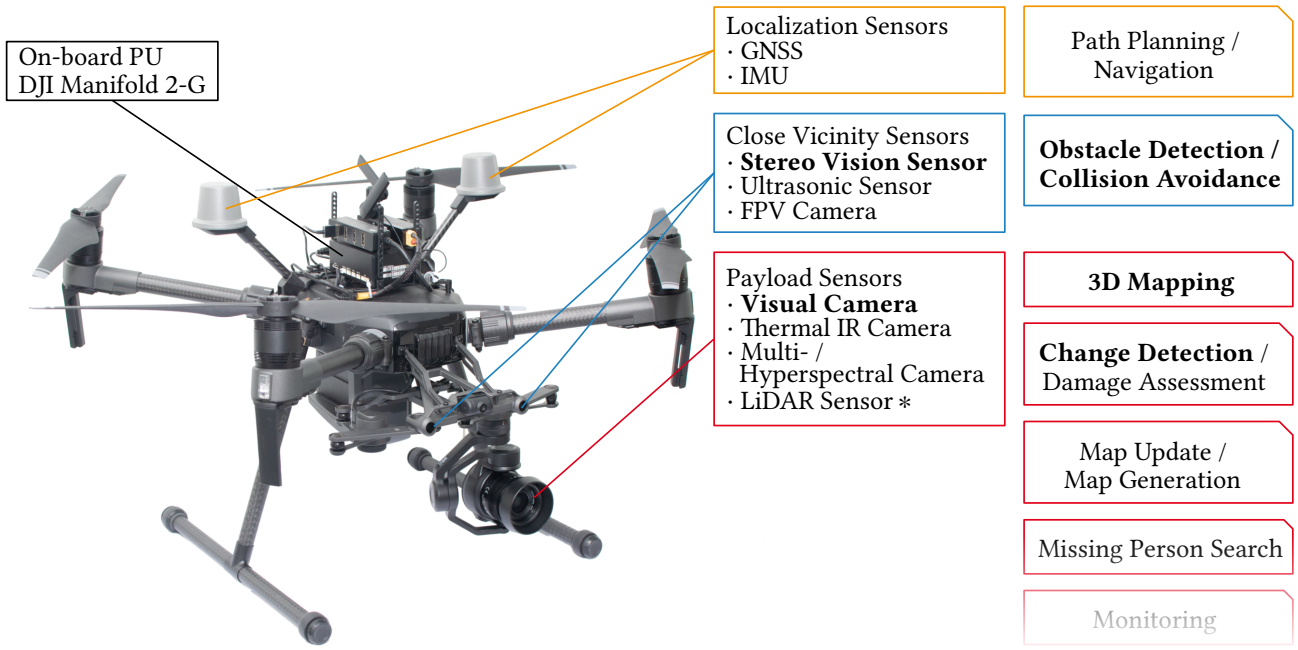


Figure 1.1: Overview of the equipment of modern high-end COTS UAVs and how it can be used to execute a variety of tasks and, in turn, facilitate disaster relief and search-and-rescue (SAR) missions. The depicted DJI Matrice 210v2 RTK is equipped with localization sensors like a GNSS and an inertial measurement unit (IMU) for precise navigation and geographical localization. Close vicinity sensors ensure a safe autonomous flight, by allowing to perform obstacle detection and collision avoidance on a high-performance on-board processing unit (PU), such as the DJI Manifold 2-G. More high-level applications, such as 3D mapping, damage assessment and missing person search can be achieved by a variety of payload sensors. This work focuses on the applications of obstacle detection and collision avoidance, 3D mapping and change detection based on visual images provided by the stereo vision sensor as well as a visual payload camera (printed in bold). ∗ The LiDAR sensor is not available for the DJI Matrice 210.

based on the NVIDIA Jetson TX2 board. To facilitate more high-level applications, the UAVs are equipped with appropriate payload sensors. UAVs from a lower-price segment typically only provide a single payload sensor that can, if at all, only be replaced by the manufacturer. High-end UAVs like the DJI Matrice Series provide a range of different plug-and-play sensors that can easily be attached to the UAV by the user. Examples of such sensors are visual cameras that provide high-resolution imagery, thermal infrared (IR) cameras, multi- or hyperspectral cameras and recently even light detection and ranging (LiDAR) sensors.

There are a large number of tasks and applications that can be executed with such sensors to facilitate disaster relief and SAR missions. For example, 3D mapping can be achieved from the camera images by means of photogrammetry or directly from the data provided by the LiDAR sensor. This is crucial for an effective assessment of the disaster site and for an appropriate planning and deployment of the emergency forces (Furutani and Minami 2021). In comparison to relying on data provided by manned aircrafts or satellites for the task of 3D mapping, the use of UAVs has a great advantage in terms of cost efficiency and rapid deployment (Eastern Kentucky University 2017, Furutani and Minami 2021). By Ferris-Rotman (2015) and DroneDeploy (2018), it is reported how COTS UAVs were used, together with a photogrammetric software for off-board 3D mapping, to help with assessing the aftermath of the 2015 Nepal and 2018 Indonesia earthquake. The images from the visual camera, paired with 3D data, are used to perform change detection and damage assessment as well as to generate a current map of the disaster site. This is helpful for the assessment of the situation by the emergency forces. In other situations, the thermal IR camera allows to efficiently search for missing persons. Silvagni et al. (2017), for example, propose to use a UAV equipped with a thermal IR camera to find missing persons in mountainous areas, that are buried in snow after an avalanche. If a multi- or hyperspectral camera is deployed,

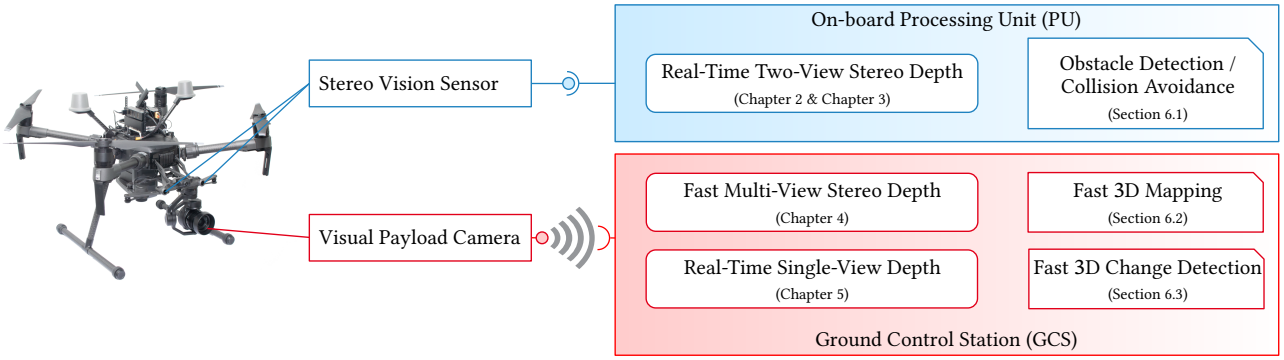


Figure 1.2: Overview of the proposed framework for the support of first responders in the rapid structural assessment of the disaster site by means of COTS UAVs. The framework is divided into two parts: (i) The blue part uses the image data from the stereo vision sensor and performs two-view stereo depth estimation in real-time, that serves as input to an obstacle detection and collision avoidance algorithm. This is done directly on board the UAV, utilizing the high-performance on-board PU, facilitating an autonomous operation of the UAV. (ii) In the red part, the image data from the payload camera is streamed down to a ground control station (GCS) (details on the data-link are not covered in the scope of this work), where it is processed by more powerful hardware. Here, an approach on fast multi-view stereo (MVS) depth estimation and single-view depth estimation (SDE) is proposed to facilitate rapid 3D mapping and change detection. Details on the individual processing steps can be found in the cross-referenced chapters and sections.

surface structures and soil moisture can be monitored in order to reason on the structural integrity of collapsed buildings (Ferris-Rotman 2015) or a possible dam failure after flooding or intense rainfall. Apart from these applications, there are a lot more use-cases for both fixed-wing and rotor-based UAVs to assist disaster relief and SAR missions. They can, for example, also be used to deliver goods, such as medical supplies, to areas which are difficult to access or even build up radio relays to facilitate communication (Shakhatreh et al. 2019).

As illustrated by the bold print in Figure 1.1, the aim of this work is to facilitate three of the above-mentioned applications, namely obstacle detection and collision avoidance, 3D mapping and change detection, by means of image-based depth estimation from data provided by the stereo vision sensor and the visual payload camera. In particular, the empowerment and support of first responders is of great interest, with focus on fire fighters and medical emergency forces. This increases the need for fast and reliable processing in order to ensure a rapid assessment. The proposed framework relies solely on data from the visual camera as they are typically deployed on modern UAVs by default and are cheaper than other sensors, such as LiDAR sensors. In the following, the proposed framework for the support of first responders in the rapid structural assessment of the disaster site by means of COTS UAVs together with the associated challenges and the contributions of this work is presented.

1.2 Proposed Framework, Challenges and Contributions

In this work, a framework for the support of first responders, e.g. fire fighters and medical emergency forces, in the rapid structural assessment of the disaster site by means of COTS UAV is proposed. The idea of the covered use-case is that in case of a disaster, which can be of natural cause, e.g. earthquake, or induced by humans, e.g. road accident, specialized first responders can use a COTS UAV in combination with a ground control station (GCS) to rapidly assess the situation by aerial reconnaissance. In particular, fast 3D and orthographic mapping is of great benefit to rapidly acquire an overview of the disaster site and in turn perform route and mission planning for the emergency forces. Moreover, given the computed 3D data of the area, change detection can be performed to assess the caused damage. Apart from the high-level applications, a save operation of the deployed UAV is to be ensured, especially if it is operated autonomously, which further increases its benefit as the operating personnel can concentrate on the data produced by the system rather than on flying the UAV

itself. While the actual flight can be realized by providing geographical waypoints to the flight controller, a dynamic obstacle detection and collision avoidance needs to be implemented for a safe autonomous flight. As discussed above, there exist numerous other applications, such as path planning or person detection, that can be executed in order to assist first responders. They are, however, not covered in the scope of this work.

With this use-case and these applications in mind, a framework that consists of two parts is proposed, as illustrated in Figure 1.2. (i) The blue part uses the image data from the stereo vision sensor and performs two-view stereo depth estimation in real-time, that serves as input to an obstacle detection and collision avoidance algorithm. This is done directly on board the UAV, utilizing the high-performance on-board PU. (ii) In the red part, the image data from the payload camera is streamed down to a GCS, where it is processed by more powerful hardware. Here, an approach on fast multi-view stereo (MVS) depth estimation and single-view depth estimation (SDE) is proposed to facilitate rapid 3D mapping and change detection. Details and realization on the data-link are not covered in the scope of this work. In the following two sections, the challenges associated with each of the outlined component as well as the contributions of this work to tackle these challenges are discussed.

1.2.1 Two-View Stereo Depth Estimation for Obstacle Detection and Collision Avoidance

As outlined above, the first part of the presented framework performs obstacle detection and collision avoidance based on dense depth maps which are estimated from a two-view camera setup. In this, the complete processing is done in real-time and on an embedded hardware which poses a number of challenges and restrictions. The main contributions of this work to tackle these challenges are presented in Chapter 2, Chapter 3 and Section 6.1 and are summarized in the following:

Reliable and high-accurate dense depth estimation: For safety-critical applications, such as obstacle detection and collision avoidance, a reliable and deterministic processing is of great importance. Moreover, in order to confidently detect obstacles and derive their location with respect to the UAV by means of stereo image processing, the depth maps must have a high completeness and be of metric scale. Here, the visual appearance of the depth map and the subpixel accuracy of the estimates play a minor role. Thus, this work proposes ReS^{2}tAC (Ruf et al. 2021b), an approach which adopts a widely used stereo algorithm for high-performance depth estimation based on stereo images. It reliably computes dense depth maps that achieve state-of-the-art accuracies on public stereo benchmarks, with an error rate as low as 4 % and a completeness of 88 % and higher.

Computationally efficient obstacle detection and collision avoidance: The proposed framework relies on the stereo vision sensor to perceive the space in front of the UAV and, in turn, perform obstacle detection and collision avoidance. In this, dense depth maps are estimated by means of stereo image processing, which has a high computational complexity since depth hypotheses are to be estimated for a great majority of pixels in order to reliably detect obstacles. This, in turn, requires the subsequent process of obstacle detection and collision avoidance to be computationally efficient in order to not exceed given run-time and latency constraints. Thus, this work demonstrates the suitability of a simple and yet effective approach (Ruf et al. 2018b) for obstacle detection and collision avoidance. This reprojects the depth maps into a top and side view of the environment in front of the UAV, allowing to detect possible obstacles by means of simple clustering and thresholding operations.

Real-time & low-latency processing: Despite the computational high complexity of estimating dense depth maps, the processing has to be done in real-time and with low-latency in order for the flight controller to still be able to initiate an appropriate avoidance maneuver. In this, the processing entails the estimation

of the depth map, as well as the detection of obstacles and the calculation of the evasion maneuver. This maneuver can be as complex as recalculating the flight path in order to evade the obstacle, or as simple as initiating an emergency stop before calculating a new route. This work shows that both the approach for dense depth estimation (Ruf et al. 2021b), as well as the approach for obstacle detection and collision avoidance (Ruf et al. 2018b), allow real-time and low-latency processing by utilizing general-purpose computation on a GPU (GPGPU) and vectorized single-instruction-multiple-data (SIMD) processing on a CPU.

On-board embedded processing: Lastly, the most challenging aspect in this is that all processing is to be done on an embedded device on board the UAV in order to ensure low latency. A communication with a GCS to allow an off-board processing would induce unnecessary latency and be subjected to additional sources of unreliability due to a possible data-link failure. The use of on-board embedded devices, however, is subjected to a reduced availability of processing capabilities. This is because a trade-off between low-power and high-performance processing is to be found in order to not excessively reduce the flight time while at the same time enabling fast processing. With ReS^{2}tAC (Ruf et al. 2021b) a real-time execution on embedded CUDA and ARM devices, such as the NVIDIA Jetson boards is demonstrated. These embedded systems are increasingly deployed for on-board processing, as the development of appropriate algorithms is less cumbersome and time-consuming compared to the development for field-programmable gate array (FPGA) technology.

The contributions presented in these chapters have been published in:

- Ruf, B.; Mohrs, J.; Weinmann, M.; Hinz, S., and Beyerer, J. (2021b): "ReS^{2}tAC – UAV-borne real-time SGM stereo optimized for embedded ARM and CUDA devices". In: *MDPI Sensors* 21.11, 3938. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2021b).
- Ruf, B.; Monka, S.; Kollmann, M., and Grinberg, M. (2018b): "Real-time on-board obstacle avoidance for UAVs based on embedded stereo vision". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1, pp. 363–370. Cited as (Ruf et al. 2018b).

1.2.2 Multi-View Stereo and Single-View Depth Estimation for Fast 3D Mapping and 3D Change Detection

The second part of the proposed framework aims at using multi-view stereo and single-view depth estimation for fast and online 3D mapping and 3D change detection. Even though it is assumed that the processing is to be done on a GCS, equipped with enough hardware resources, the requirements and the nature of the input data impose a number of other challenges. These are tackled by the contributions presented Chapter 4, Chapter 5, as well as in Section 6.2 and Section 6.3, and are summarized in the following:

Accurate dense 3D mapping and rapid 3D change detection: In order to facilitate a good assessment of a disaster site, as outlined by the considered use-case, an accurate and dense 3D mapping is essential. In this, for the actual representation of the area and, in turn, the assessment of the situation, it is not critical if the 3D map misses a high level-of-detail. Yet, it is of great importance that it is densely and thoroughly reconstructed without big holes, i.e. areas with missing data. Just as in case of software suites for offline reconstruction, dense 3D mapping requires the computation of dense depth maps by means of dense image matching (DIM) with a subsequent fusion to a 3D point cloud or 3D model. With FaSS-MVS (Ruf et al. 2021a), this work proposes an approach that efficiently estimates accurate dense depth maps which

have an absolute error of approximately only 1 % with respect to the maximum depth of the reconstructed scene. Furthermore, this work presents how appropriate algorithms for dense depth estimation can be utilized in a processing pipeline for fast and online 3D mapping from aerial imagery captured by COTS UAVs (Hermann et al. 2021a) as well as for rapid 3D change detection (Ruf and Schuchert 2016).

Efficient handling of large scene depth and oblique imagery: A large scene depth poses a great challenge to the estimation of dense depth maps, especially if the processing is subjected to run-time constraints. While images captured from a nadir viewpoint only have a confined depth range, which typically depends on the flight altitude, a straight flyover is not always possible due to numerous reasons, for example, because of fire and smoke or other air traffic. In such cases, the scene can only be captured from oblique viewpoints which, in turn, increases the scene depth and with it the computational complexity, as more depth hypotheses have to be generated and evaluated. Moreover, due to the oblique vantage points, the scenes depicted by such imagery are mostly comprised of non-fronto-parallel surface structures. This requires the consideration of slanted surfaces in the estimation of dense depth maps in order to not induce stair-casing artifacts in the resulting reconstruction. The approach for multi-view stereo (Ruf et al. 2019, Ruf et al. 2021a) presented in this work, i.e. FaSS-MVS, employs a hierarchical processing scheme in the estimation of dense depth maps and thus allows to efficiently handle large scene depths of oblique imagery by making use of a coarse-to-fine strategy. Furthermore, this work proposes a surface-aware optimization scheme for the presented multi-view stereo approach in order to account for non-fronto-parallel surfaces and, in turn, reduce stair-casing artifacts in the reconstruction of slanted surfaces.

Handling the existence of dynamic scene objects: When performing monocular 3D modeling, a key assumption is that the scene remains static while the camera moves around it during data acquisition. During the operation of first responders, this can seldom be ensured, as the emergency forces themselves or/and other persons, e.g. casualties, might be moving around on the disaster site. On the other hand, depending on the application, it might also be of benefit and of interest to model the disaster site in 4D. This means that moving objects should be correctly placed inside the model. A possible application would be to monitor the operation of the emergency forces in order to ensure their safety. To account for the inability of monocular multi-view stereo approaches to reconstruct dynamic objects, this work presents an approach for single-view depth estimation (Hermann et al. 2020), i.e. depth estimation from a single aerial image. This approach learns to predict depth maps from a single input image by means of a deep convolutional neural network (CNN) and is thus able to also predict depth maps for scenes with dynamic objects. Furthermore, it is trained in a self-supervised manner, not requiring any special training data apart from a large number of camera images, making it suitable for a great variety of domains. This allows to execute the presented approach for single-view depth estimation in parallel to a monocular multi-view stereo approach, enabling the fusion of the depth maps from both approaches and, thus, filling missing estimates in multi-view stereo depth maps with the results of the depth map from the single-view depth estimation approach.

Fast and online processing: Again, in all this, the time it takes to process the data and to generate the desired output is not irrelevant. As the outlined use-case aims at supporting first responders, the processing is to be done fast and online. This means that the 3D mapping is to be done while the UAV is acquiring image data and streaming it to the GCS. In contrast to the embedded processing for obstacle detection and collision avoidance, however, there is little restriction with respect to the available processing resources. Nonetheless, fast and online dense depth estimation by means of multi-view stereo and a subsequent 3D mapping is still challenging, not only due to the reasons mentioned above, but also because not all

data is available at the time of processing. In particular with respect to handling oblique imagery with a large scene depth, this might inhibit the exploitation of an appropriate baseline between the images, which is sufficient for an accurate reconstruction. Again, by employing optimizations for massively parallelized GPGPU, the presented approach for multi-view stereo, i.e. FaSS-MVS (Ruf et al. 2021a), achieves a processing rate of 1-2 Hz, which is well-suited for online processing of a monocular image sequence. This work further shows, that, depending on the image size, the presented approach for single-view depth estimation (Hermann et al. 2020) even reaches theoretical frame rates of up to 250 FPS, making it capable of real-time and low-latency processing.

The contributions presented in these chapters have been published in:

- Ruf, B.; Weinmann, M., and Hinz, S. (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821*. Published as preprint. Cited as (Ruf et al. 2021a).
- Hermann, M.; Ruf, B., and Weinmann, M. (2021a): "Real-time dense 3D reconstruction from monocular video data captured by low-cost UAVs". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2021, pp. 361–368. Cited as (Hermann et al. 2021a).
- Hermann, M.; Ruf, B.; Weinmann, M., and Hinz, S. (2020): "Self-supervised learning for monocular depth estimation from aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-2-2020, pp. 357–364. **Peer-reviewed** on the basis of the full paper. Cited as (Hermann et al. 2020).
- Ruf, B.; Pollok, T., and Weinmann, M. (2019): "Efficient surface-aware semi-global matching with multi-view plane-sweep sampling". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W7, pp. 137–144. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2019).
- Ruf, B.; Thiel, L., and Weinmann, M. (2018a): "Deep cross-domain building extraction for selective depth estimation from oblique aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-1, pp. 125–132. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2018a).
- Ruf, B.; Erdnüß, B., and Weinmann, M. (2017): "Determining plane-sweep sampling points in image space using the cross-ratio for image-based depth estimation". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W6, pp. 325–332. Cited as (Ruf et al. 2017).
- Ruf, B. and Schuchert, T. (2016): "Towards real-time change detection in videos based on existing 3D models". In: *Proceedings of the SPIE Image and Signal Processing for Remote Sensing XXII*, pp. 489 –502. Cited as (Ruf and Schuchert 2016).

2 Dense Depth Estimation by Two-View and Multi-View Stereo

This chapter includes material from the following publications:

Ruf, B.; Mohrs, J.; Weinmann, M.; Hinz, S., and Beyerer, J. (2021b): "ReS^{2}tAC – UAV-borne real-time SGM stereo optimized for embedded ARM and CUDA devices". In: *MDPI Sensors* 21.11, 3938. Reprinted with permission. It is cited as (Ruf et al. 2021b) and marked with a blue sidebar.

Ruf, B.; Weinmann, M., and Hinz, S. (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821*. Reprinted with permission. It is cited as (Ruf et al. 2021a) and marked with a green sidebar.

The ultimate goal of computer vision is to enable computers and robotic systems to fully perceive and understand their surrounding by the interpretation of camera images, just like the human brain does with the visual information it receives from the eyes. Moreover, to be able to freely navigate through and interact with the three-dimensional world, it is of vital importance to be able to perceive and map the surrounding not only in 2D but also in 3D. In the process of image acquisition, however, the three-dimensional world gets projected onto a two-dimensional plane, discarding one dimension and loosing vital information on the depth of the scene, which is important to be able to reconstruct the world or objects therein in 3D. Thus, an essential task for robotic vision is to estimate the depth of the scene and, in turn, to reason about the 3D geometry. If, due

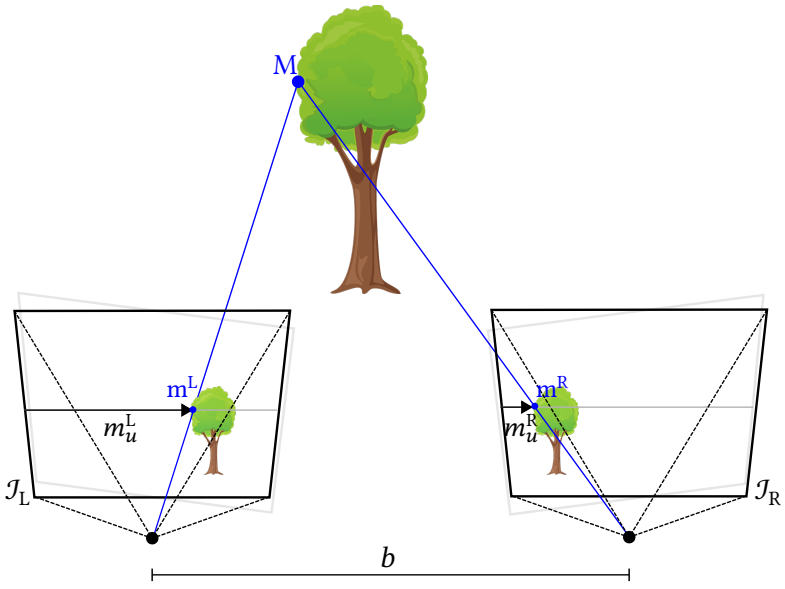


Figure 2.1: Illustration of a stereo camera setup. Two cameras placed apart from each other with a distance b (baseline) observe the same scene from two different vantage points. A scene point M is projected onto different locations m^L and m^R in the two camera images $\mathcal{I}_L$ and $\mathcal{I}_R$. If the image pair is rectified, the difference between m^L and m^R is reduced to a horizontal shift, the so-called disparity $\Delta u = |m_u^L - m_u^R|$. (Ruf et al. 2021b, Fig. A1.)

to different constraints, no active sensing (e.g. LiDAR systems) is available, this has to be done by solely relying on 2D camera vision. With enough training, us humans, and also modern machine learning algorithms, are able to learn how to predict a sense of scene depth from single images. This prediction, however, is only based on the experience and knowledge gained in life or during training, respectively. It can fail, if there is not enough contextual information available in the image, or if the scene that is depicted is unknown or difficult to grasp. Moreover, in such a case, the predictions on sizes and distances that are made are only relative, i.e. how big is one object with respect to another, or how far away is a certain scene point relative to a reference.

However, the scene depth can be accurately reconstructed if not one, but at least two cameras are available and both are positioned in such a way that they observe the same scene at the same time from two slightly different vantage points. Just like with the two eyes of us humans, the slight difference in the arrangement of the scene observed in the images of the two cameras can be used to reason about the distance of the depicted object with respect to the camera setup. This is referred to as *stereoscopic* vision, or in short *stereo*. As illustrated in Figure 2.1, two cameras placed apart from each other with a distance b (baseline) observe a scene point M from two different vantage points. The scene point will appear at different locations m^{L} and m^{R} in the two camera images $\mathcal{I}_{\mathrm{L}}$ and $\mathcal{I}_{\mathrm{R}}$. Finding such correspondences is the essential task in the process of stereo processing, and in turn dense depth estimation. (Ruf et al. 2021b, Appendix A.)

Generally speaking, a correspondence search between an image point in the one camera image and the image projection of the same scene point in the other camera image has to be conducted along the epipolar curve, which is the viewing ray going through the image point of the first camera projected into the image of the second camera. This correlation is formulated by the epipolar constraints (cf. (Hartley and Zisserman 2004, Sec. 9.1.)). If the camera rig is not calibrated, the two image planes are not rectified, i.e. the image planes are not co-planar. This is illustrated by the light gray boxes in Figure 2.1. When the intrinsic parameters of the cameras are known, it is possible to account for image distortions, transforming the curve into an epipolar line. Furthermore, if the relative position and orientation between the two cameras of the stereo rig are known, the camera images can be rectified, i.e. they can be transformed onto a common image plane, and in turn the epipolar lines can be transformed to coincide with the image row of the image points. Thus, the difference between the image positions of m^{L} and m^{R} is reduced to a horizontal shift, the so-called *disparity* $\Delta u = |m_u^{\mathrm{L}} - m_u^{\mathrm{R}}|$. (Ruf et al. 2021b, Appendix A.)

From this, the *depth* d of the scene point, i.e. the orthogonal distance from the optical center of the reference camera, can be deduced according to:

$$d = \frac{b \cdot \hat{f}}{\Delta u \cdot h_u}. \tag{2.1}$$

In this equation, $\hat{f}$ denotes the actual focal length of the optics, while h_u states the width of a pixel on the sensor chip of the camera, both given in millimeters (mm). Together with the baseline b, which is typically also given in a metric system, these additional parameters allow to convert a disparity map, given in pixels, into a depth map with a metric scale. In case the intrinsic parameters of a camera are not known a priori and thus are calibrated with images of a calibration pattern, the focal length and the size of the sensor element are often combined into a single, two-dimensional parameter, i.e. f_u and f_v. These are denoted as the camera constant in u- and v-direction. They are located in the diagonal of the intrinsic camera matrix and are usually denoted as the mathematical focal length of the pinhole camera system.

Figure 2.2: General processing pipeline for the task of dense disparity or depth estimation based on two-view stereo. Given two input images $\mathcal{I}_L$ and $\mathcal{I}_R$, a dense image matching and cost computation is first executed, yielding a three-dimensional cost volume $\mathcal{S}$. Then, a cost optimization and regularization are performed, transforming the raw cost volume $\mathcal{S}$ into a regularized cost volume $\bar{\mathcal{S}}$. In the last stage, a disparity map $\mathcal{U}$ or depth map $\mathcal{D}$ is extracted from $\bar{\mathcal{S}}$ as well as refinement and post-processing operations are executed.

In their work, Scharstein and Szeliski (2002) have studied and categorized a number of stereo algorithms for dense disparity or depth estimation based on their processing steps. From their observations, they have derived a general processing pipeline, which provides a basic blueprint for most modern stereo algorithms, as illustrated in Figure 2.2. In this, the estimation of dense disparity maps can be subdivided into three subsequent steps. Given two input images $\mathcal{I}_L$ and $\mathcal{I}_R$, a dense image matching and cost computation is first executed, yielding a three-dimensional cost volume $\mathcal{S}$. Then, a cost optimization and regularization are performed, transforming the raw cost volume $\mathcal{S}$ into a regularized cost volume $\bar{\mathcal{S}}$. In the last stage, a disparity map $\mathcal{U}$ or depth map $\mathcal{D}$ is extracted from $\bar{\mathcal{S}}$ as well as refinement and post-processing operations are executed. (Ruf et al. 2021b, Appendix A.)

The following sections give a brief overview on each individual step and its specific task within the processing chain. While the processing pipeline is proposed for the disparity and depth estimation from two images, it can also be adopted for the use of three or more input images. The most significant change in this adaptation is done with respect to the dense image matching and cost computation, which is discussed in the next section.

2.1 Dense Image Matching and Cost Computation

Finding two corresponding image points, that depict the same scene point in both images of a stereo camera setup, means to match the pixels of the reference image, typically the image of the left camera, against each pixel in the matching image within a certain disparity range $\Delta u \in \Gamma_u = [\Delta u_{\min}, \Delta u_{\max}]$. In this, the similarity between two pixels is modeled by a similarity measure from which a cost function $C(\cdot)$ can be deduced, which typically is at its minimum if both pixels coincide. This so-called matching cost between a pixel $\mathrm{p}^L = (u, v)$ in the left image and a corresponding pixel p^R, located according to a disparity shift Δu in the right image, is stored in the three-dimensional cost volume $\mathcal{S}$:

$$\mathcal{S}(\mathrm{p}^L, \Delta u) = C\left(\mathcal{I}_L(u, v), \mathcal{I}_R(u - \Delta u, v)\right). \tag{2.2}$$

Thus, the objective in finding two corresponding image points is to minimize the matching cost computed by $C(\cdot)$. (Ruf et al. 2021b, Sec. 2.1.1.)

When relying on distinctive image features like SIFT (Lowe 2004), SURF (Bay et al. 2006) or ORB (Rublee et al. 2011) features, a unique matching between two image points can be found. However, such image features can only be calculated in descriptive image regions, resulting in a sparse correspondence field. Thus, in the case of *dense image matching (DIM)*, Equation (2.2) is evaluated for each pixel in the reference image, computing a pixel-wise matching cost s according to $C(\cdot)$, which indicates the similarity between the pixel pair. This cost

is then stored inside a three-dimensional cost volume $\mathcal{S}$, from which the disparity map is later extracted. (Ruf et al. 2021b, Sec. 2.1.1.)

Since the disparity is only evaluated along the same pixel row, it is assumed that the input images $\mathcal{I}_L$ and $\mathcal{I}_R$ are rectified prior to the process of image matching. The OpenCV library[1] provides a standard calibration routine (Zhang 2000), which can be used to calculate two rectification maps that allow to efficiently resample the input images such that the epipolar lines lie horizontally on the image rows. (Ruf et al. 2021b, Sec. 2.1.1.)

Two prominent and widely used similarity measures for DIM are the census transform and the normalized cross correlation. Both use a support region of fixed size around a reference pixel, which is typically situated in the center of the support region. Thus, the support region typically has an uneven side length. The computation of the census transform and normalized cross correlation as well as the corresponding cost functions are described in Section 2.1.1 and Section 2.1.2, respectively.

2.1.1 Census Transform and its Hamming Distance

First proposed by Zabih and Woodfill (1994), the *census transform (CT)* is often used as a similarity measure for efficient and real-time DIM, due to its low computational complexity. Given a support region of fixed size of $m \times n$ pixels, each pixel of a grayscale image within the support region is converted into a bit-string of length $m \cdot n$. This is done by comparing the intensities of each neighbor pixel q to the intensity of the reference pixel p at the center of the support region and setting the according bit to 1, if the intensity of q is less than that of p:

$$CT(\mathrm{p}, \mathrm{q}) = \begin{cases} 0, \text{ if } \mathcal{I}(\mathrm{q}) \geq \mathcal{I}(\mathrm{p}) \\ 1, \text{ if } \mathcal{I}(\mathrm{q}) < \mathcal{I}(\mathrm{p}) \end{cases} \tag{2.3}$$

The similarity between two image patches is then calculated by computing the *Hamming distance* between the corresponding bit-strings, i.e. by counting how many bits differ between the two strings. In this, it is assumed that the lower the Hamming distance is, the more similar the two corresponding image patches are. While the CT is very robust to local illumination changes, it is not very descriptive, since it reduces the actual image content to a bit-string, which only encodes the relative intensities within a confined support region.

2.1.2 Truncated and Scaled Normalized Cross Correlation

Compared to the CT and its Hamming distance, the *normalized cross correlation (NCC)* is much more descriptive and unique in the matching of two patches with a size of $m \times n$ pixels. However, the computation of the NCC is significantly more complex than the computation of the CT. The NCC between the image patches around two pixels p^L and p^R is computed according to:

$$NCC\left(\mathrm{p}^L, \mathrm{p}^R\right) = \frac{\sum_{\mathrm{q}_i \in \mathcal{P}^L, \mathrm{q}_k \in \mathcal{P}^R} \left(\left(\mathcal{I}_L(\mathrm{q}_i) - \bar{\mathcal{P}}^L\right) \cdot \left(\mathcal{I}_R(\mathrm{q}_k) - \bar{\mathcal{P}}^R\right)\right)}{\sqrt{\sum_{\mathrm{q}_i \in \mathcal{P}^L} \left(\mathcal{I}_L(\mathrm{q}_i) - \bar{\mathcal{P}}^L\right)^2} \cdot \sqrt{\sum_{\mathrm{q}_k \in \mathcal{P}^R} \left(\mathcal{I}_R(\mathrm{q}_k) - \bar{\mathcal{P}}^R\right)^2}}, \tag{2.4}$$

with $\mathcal{P}^L$ and $\mathcal{P}^R$ being the image patch around p^L and p^R, respectively, and with $\bar{\mathcal{P}}^L$ and $\bar{\mathcal{P}}^R$ being the mean intensity values of the corresponding image patches. The NCC can geometrically be interpreted as the dot product between the vectors of the intensity values corresponding to the pixels' zero-mean image patches.

[1] www.opencv.org

Thus, it is at its maximum, i.e. 1, if both patches are equal. Since the range of the NCC is [-1,1], it is typically truncated and scaled when used as a cost function in the process of DIM (Sinha et al. 2014, Scharstein et al. 2017):

$$C_{\mathrm{NCC}}\left(\mathrm{p}^{\mathrm{L}}, \mathrm{p}^{\mathrm{R}}\right) = \left(1 - \max\left(0, NCC(\mathrm{p}^{\mathrm{L}}, \mathrm{p}^{\mathrm{R}})\right)\right) \cdot 255. \tag{2.5}$$

2.1.3 True Multi-Image Matching by Means of Plane-Sweep Sampling

As discussed above, the epipolar constraint formulates the relationship between an image point m^{L} in the one camera image and the epipolar line $\mathrm{l}^{\mathrm{R}}_{\mathrm{m}}$, on which the corresponding image point m^{R} in the other camera image lies. Here, both image points depict the same scene point. This correlation is manifested in the so-called *fundamental matrix* $\mathbf{F}$:

$$\mathrm{l}^{\mathrm{R}}_{\mathrm{m}} = \mathbf{F} \cdot \mathrm{m}^{\mathrm{L}}. \tag{2.6}$$

A similar characteristic exists for the three- and four-view geometry. Analogously to the fundamental matrix, the trifocal and quadrifocal tensors allow to establish point-line correspondences between three and four views, respectively. Due to the increasing complexity in the computation of these tensors with an increasing number of views, their practical use does not go beyond four views (Hartley and Zisserman 2004). (Ruf et al. 2021a, Sec. 2.2.)

Besides the fundamental matrix, there exists a second projective relationship between two views, which can be extended equally to an arbitrary number of views. Again, the case of a two-camera setup with known intrinsic and extrinsic parameters is considered. This time, however, an additional scene plane $\Pi = (\mathrm{N}_{\Pi}, \delta)$ is positioned in the field-of-view (FOV) of both cameras. Here, the scene plane Π is parameterized by its normal vector N_{Π} and its distance δ from the first camera, i.e. the reference camera. Given this setup, the image point m^{L} in one camera image can directly be mapped onto the image point m^{R} in the second camera image via the homography $\mathbf{H}$ induced by the plane Π according to

$$\begin{aligned} \mathrm{m}^{\mathrm{R}} &= \mathbf{H}\left(\Pi, \mathbf{P}_{\mathrm{L}}, \mathbf{P}_{\mathrm{R}}\right) \cdot \mathrm{m}^{\mathrm{L}}\,, \text{ with} \\ \mathbf{H}\left(\Pi, \mathbf{P}_{\mathrm{L}}, \mathbf{P}_{\mathrm{R}}\right) &= \mathbf{K}_{\mathrm{R}} \cdot \frac{\mathbf{R} - \mathrm{T} \cdot \mathrm{N}^{\mathrm{T}}_{\Pi}}{\delta} \cdot \mathbf{K}^{-1}_{\mathrm{L}}. \end{aligned} \tag{2.7}$$

Here, $\mathbf{K}_{\mathrm{L}}$ and $\mathbf{K}_{\mathrm{R}}$ denote the intrinsic matrices of both cameras, and $[\mathbf{R}\ \mathrm{T}]$ denotes the relative transformation matrix of the pose $\mathbf{P}_{\mathrm{R}}$ of the second camera with respect to the pose $\mathbf{P}_{\mathrm{L}}$ of the first camera. Equation (2.7) is geometrically interpreted by casting a viewing ray through the image point m^{L} and intersecting it with the scene plane Π, yielding a scene point M, which is then projected into the second camera, resulting in the image point m^{R} (Hartley and Zisserman 2004). (Ruf et al. 2021a, Sec. 2.2.)

Based on the relationship between two cameras and a scene plane, Collins (1996) proposed an algorithm for true multi-image matching (cf. Algorithm 2.1). This algorithm samples the scene-space between two bounding planes Π_{max} and Π_{min}, located at δ_{max} and δ_{min}, by sweeping a plane along its normal vector N_{Π} through space and matching the pixels inside the input images according to Equation (2.7) for each distance $\delta \in [\delta_{\mathrm{min}}, \delta_{\mathrm{max}}]$ of the plane relative to the reference camera. For each position of the plane, an arbitrary number of matching images are warped by the plane-induced homography $\mathbf{H}^{-1}_{\mathrm{ref}\rightarrow k}$ into the view of the reference camera, where they are matched against the reference image. If the scene plane is close to a three-dimensional structure, then the corresponding image regions of the warped matching images overlap with those in the reference image,

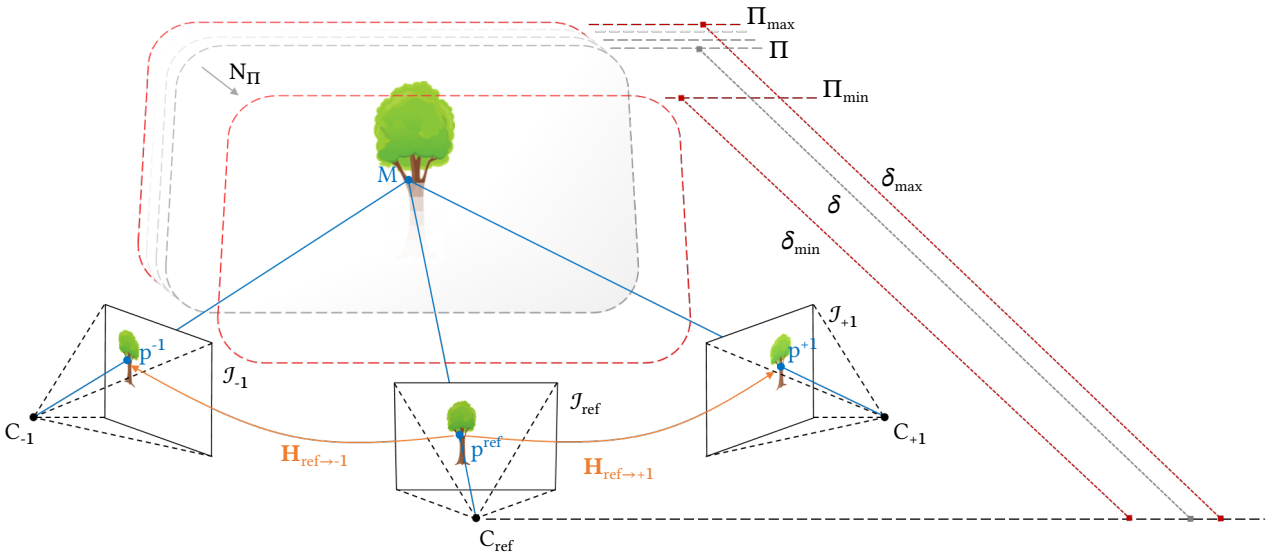


Figure 2.3: Illustration of the plane-sweep algorithm for multi-image matching. A scene is sampled by a plane $\Pi = (\mathrm{N}_\Pi, \delta)$, with N_Π being the normal vector of the plane and δ being the orthogonal distance of the plane from $\mathrm{C}_{\mathrm{ref}}$, that is swept along its normal vector between two bounding planes Π_{max} and Π_{min} through space. For each distance δ of Π the reference pixel $\mathrm{p}^{\mathrm{ref}}$ is projected by the plane induced homography $\mathbf{H}_{\mathrm{ref}\to k}$ into arbitrary number of viewpoints, where it is matched against the corresponding pixel in $\mathcal{I}_k$. (Ruf et al. 2021a, Fig. 2.)

allowing to deduce the scene depth of the corresponding object from the parameterization of the corresponding plane. Initially denoted as space-sweep algorithm, it was adopted by numerous studies on multi-image matching and MVS (Gallup et al. 2007, Pollefeys et al. 2008, Sinha et al. 2014), eventually denoting it as *plane-sweep* algorithm. (Ruf et al. 2021a, Sec. 2.2.1.)

An overview on the basic plane-sweep algorithm for true multi-image matching is given in Figure 2.3 and Algorithm 2.1. Moving from a two-camera setup to an arbitrary number of cameras, it is not distinguished between left and right image anymore, but between the reference image $\mathcal{I}_{\mathrm{ref}}$ and the matching images $\mathcal{I}_k \setminus \mathcal{I}_{\mathrm{ref}}$. Moreover, when using the plane-sweep algorithm, the disparity Δu in the image matching is substituted by the plane distance δ. Given a plane parameterization, the depth can be deduced by intersecting the viewing ray through pixel $\mathrm{p} = (u\ v)^\intercal$ with the corresponding plane:

$$d_{\mathrm{p}} = \frac{-\delta}{\mathrm{N}_\Pi^\intercal \cdot \mathbf{K}^{-1} \cdot (u\ v\ 1)^\intercal}. \tag{2.8}$$

Algorithm 2.1: The basic plane-sweep algorithm for true multi-image matching.

Input Data: multiple images $\mathcal{I}_k$ with a pre-selected reference image $\mathcal{I}_{\text{ref}}$, corresponding camera calibration matrices $\mathbf{K}_k$ and camera poses $\mathbf{P}_k$, as well as a set of planes Π_{N} with a normal vector N_Π and distances $\delta \in [\delta_{\text{min}}, \delta_{\text{max}}]$.

Result: cost volume $\mathcal{S}$, holding the matching cost for all pixels $\text{p} \in \mathcal{I}_{\text{ref}}$, matched according to $\Pi \in \Pi_{\text{N}}$.

foreach *plane* $\Pi \in \Pi_{\text{N}}$ **do**

· warp all matching images $\mathcal{I}_k \setminus \mathcal{I}_{\text{ref}}$ according to $\mathbf{H}\left(\Pi, \mathbf{P}_{\text{ref}}, \mathbf{P}_k\right)^{-1}$ into the view of the reference camera, yielding $\tilde{\mathcal{I}}_k$.

foreach *pixel* $\text{p}^{\text{ref}} = (u, v) \in \mathcal{I}_{\text{ref}}$ **do**

· compute matching costs between reference image and warped matching images:

$$\mathcal{S}(\text{p}^{\text{ref}}, \Pi) = \sum_k C\left(\mathcal{I}_{\text{ref}}(u, v), \tilde{\mathcal{I}}_k(u, v)\right).$$

end

end

2.2 Cost Optimization and Regularization

Given the previously computed three-dimensional cost volume $\mathcal{S}$, the next step consists of extracting a plausible disparity map $\mathcal{U}$. Since each voxel of $\mathcal{S}$ holds the matching cost of a particular pixel p of the reference image, associated with a certain disparity Δu within the studied disparity range Γ_u, a straight-forward approach to compute a disparity map from the cost volume is to take the *winner-takes-it-all (WTA)* solution for all pixels inside the reference image:

$$\mathcal{U}(\text{p}) = \underset{\Delta u \in \Gamma_u}{\arg\min}\, \mathcal{S}(\text{p}, \Delta u). \tag{2.9}$$

However, due to the limited descriptiveness of the cost functions and resulting ambiguities in the cost volume, this would lead to a noisy and unsuitable disparity map. Thus, it is important to perform a cost optimization and, in turn, regularization of the cost volume. (Ruf et al. 2021b, Sec. 2.1.2)

Scharstein and Szeliski (2002) have categorized the stereo methods according to their cost optimization strategies into *local* and *global* methods. While the first group of algorithms only optimize the cost volume in a locally confined window and make implicit smoothness assumptions, global methods explicitly state their regularization scheme and perform an optimization within the whole image domain. Global methods will thus produce more accurate disparity maps compared to those estimated by local methods. Yet, at the same time, not all global methods are feasible for fast and embedded processing due to their complexity. (Ruf et al. 2021b, Sec. 2.1.2.)

2.2.1 Semi-Global Matching

In their taxonomy, Scharstein and Szeliski (2002) additionally propose a third group of algorithms, which utilize dynamic programming to compute a disparity map. Algorithms in this group usually state an explicit smoothness assumption and approximate a global optimization scheme, which is why this group can be considered as a subgroup of the global methods. The most prominent algorithm, especially for real-time processing, is the so-called *Semi-Global Matching (SGM)* algorithm (Hirschmüller 2005, Hirschmüller 2008). Here, the optimization

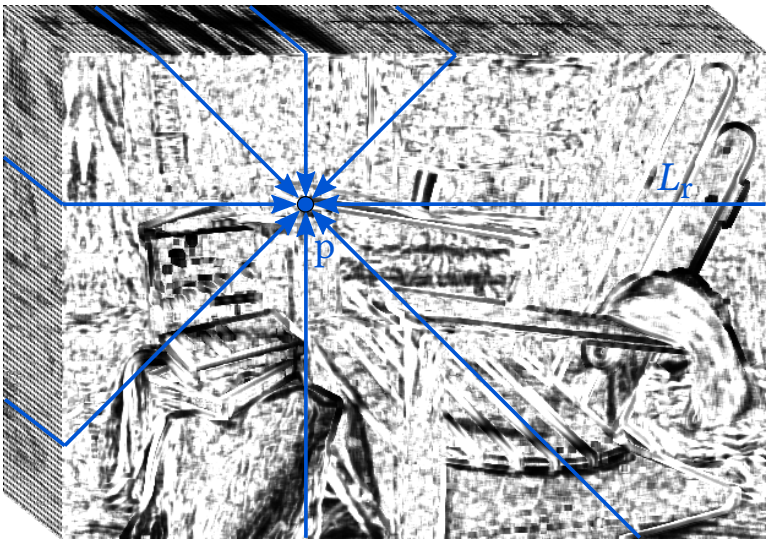


Figure 2.4: Illustration of the path aggregation within SGM to regularize the three-dimensional cost volume. For each pixel p, the cost volume $\mathcal{S}$ is optimized along eight concentric paths L_r.

scheme is formulated as a 2D Markov random field (MRF) and the minimization of the energy function E:

$$E(\mathcal{U}) = \sum_{\mathrm{p}} \left(\mathcal{S}(\mathrm{p}, \mathcal{U}(\mathrm{p})) + \sum_{\mathrm{q} \in \mathcal{R}_\mathrm{p}} \varphi_1 \cdot [|\mathcal{U}(\mathrm{p}) - \mathcal{U}(\mathrm{q})| = 1] + \sum_{\mathrm{q} \in \mathcal{R}_\mathrm{p}} \varphi_2 \cdot [|\mathcal{U}(\mathrm{p}) - \mathcal{U}(\mathrm{q})| > 1] \right), \tag{2.10}$$

with an explicit smoothness assumption, which penalizes the disparity deviation within the support region $\mathcal{R}_\mathrm{p}$ of pixel p by φ_1 and φ_2 according to the magnitude of disparity differences, with $[\cdot]$ denoting the Iverson bracket. As illustrated by Figure 2.4, the minimization of E is approximated by aggregating the matching costs along a number of concentric paths for each pixel p within the image domain:

$$\begin{aligned} L_\mathrm{r}(\mathrm{p}, \Delta u) &= \mathcal{S}(\mathrm{p}, \Delta u) + \min_{\Delta u'} \left(L_\mathrm{r}(\mathrm{p} - \mathrm{r}, \Delta u') + V(\Delta u, \Delta u') \right), \text{ with} \\ V(\Delta u, \Delta u') &= \begin{cases} 0, & \text{if } \Delta u = \Delta u' \\ \varphi_1, & \text{if } |\Delta u - \Delta u'| = 1 \\ \varphi_2, & \text{if } |\Delta u - \Delta u'| > 1. \end{cases} \end{aligned} \tag{2.11}$$

In L_r, for each pixel p and disparity Δu inside the cost volume $\mathcal{S}$, the matching costs are recursively summed up while moving along the path with the direction r. Within the smoothness term $V(\Delta u, \Delta u')$, the matching costs of the previously considered pixel with respect to all evaluated disparities u' are penalized with φ_1 or φ_2 according to the disparity difference between Δu and $\Delta u'$. Finally, for each pixel, all path costs are summed up and stored inside the aggregated cost volume $\bar{\mathcal{S}}$:

$$\bar{\mathcal{S}}(\mathrm{p}, \Delta u) = \sum_{\mathrm{r}} L_\mathrm{r}(\mathrm{p}, \Delta u), \tag{2.12}$$

before extracting the WTA disparity map according to Equation (2.9) from $\bar{\mathcal{S}}$. (Ruf et al. 2021b, Sec. 2.1.2.)

The use of dynamic programming, i.e. breaking down the minimization problem of the energy function into the aggregation of independent one-dimensional paths, makes the SGM approach well-suited for massively parallel computing and vector processing, and in turn for real-time and even embedded processing. At the

same time, many studies have shown that the results of the SGM algorithm are very accurate, making it one of the most widely used algorithms for real-time and accurate stereo processing. (Ruf et al. 2021b, Sec. 2.1.2.)

2.3 Refinement and Post-Processing

There are a number of post-processing steps, i.e. filtering, regularization and further optimizations, which can be applied to the initial disparity map or depth map in the final stage of the processing pipeline. Typically implemented are the subpixel disparity or depth refinement, outlier filters, as well as geometric consistency checks for occlusion detection and final outlier filtering. (Ruf et al. 2021b, Sec. 2.1.3.)

2.3.1 Disparity or Depth Refinement

The initial disparity map $\mathcal{U}$ computed by the SGM optimization is made up of discrete disparity values, which is sufficient for a diversity of robotic applications such as the perception of the surrounding and obstacle detection. However, when the aim is to accurately reconstruct the scene, it is important to also account for slanted surfaces and thus incorporate a subpixel refinement of the disparity or the depth. A simple and yet effective way to implement such a refinement is to use the minimum matching cost for each pixel, i.e. the matching cost corresponding to the WTA disparity $\Delta\hat{u}$, as well as the matching costs of the two neighboring disparities in front and behind of $\Delta\hat{u}$, and fit a parabola through these three matching costs. The location of the minimum of this parabola with respect to $\Delta\hat{u}$ is then considered as the subpixel refinement and added to $\Delta\hat{u}$. This optimization has only a minor computational overhead and is thus well-suited for real-time processing. However, it requires to work with floating point arithmetic, which might be of restriction to some embedded hardware. (Ruf et al. 2021b, Sec. 2.1.3.)

2.3.2 Image-Based Filtering

Before a disparity or depth map is finalized for further processing, a simple filtering is typically performed in order to regularize the measurements in a local neighborhood or remove still remaining outliers. In the scope of this work, a median filter (Section 2.3.2.1) as well as a difference-of-Gaussian filter (Section 2.3.2.2) is employed.

2.3.2.1 Median Filter

A commonly used approach for noise reduction in disparity or depth maps is the employment of a local median filter in order to remove small outliers within a confined support region. In this, for each pixel p within the disparity or depth map, all estimates inside a local support region around p are first sorted according to their estimated value. Then, the estimate that is located at the center of the sorted vector, i.e. the median, is assigned to p. The median filter is computationally very efficient. The computationally most demanding part is the sorting of the estimates, which only poses a problem when data cannot be randomly accessed and should be processed in order, e.g. in case the filter is optimized for vectorized processing on CPUs or FPGAs. However, this can be remedied when utilizing the concept of sorting networks as described in Section 3.3.2.4.

Algorithm 2.2: The difference-of-Gaussian filter to invalidate all image pixels belonging to weakly-textured areas. (Ruf et al. 2021a, Alg. 3.)

Input Data: unfiltered disparity or depth map ($\mathcal{U}$ or $\mathcal{D}$), as well as corresponding reference image $\mathcal{I}_{\text{ref}}$.
Result: filtered $\mathcal{U}$ or $\mathcal{D}$, in which all estimates corresponding to weakly-textured areas in $\mathcal{I}_{\text{ref}}$ are removed.

· use a Gaussian filter with a kernel of $m \times m$ pixels to smooth the reference frame $\mathcal{I}_{\text{ref}}$, yielding $\mathcal{I}_{\text{ref}}^{\text{smooth}}$.
· compute the DOG image depicting local image gradients according to: $\mathcal{I}_{\text{ref}}^{\text{DoG}} = \mathcal{I}_{\text{ref}} - \mathcal{I}_{\text{ref}}^{\text{smooth}}$.
· apply a binary threshold to compute the DOG mask $\mathcal{M}^{\text{DoG}}$ marking all image areas in which the intensity change is greater than 0.5.
· remove activation areas smaller than α pixels in $\mathcal{M}^{\text{DoG}}$ by applying a speckle filter.
· dilate $\mathcal{M}^{\text{DoG}}$ with a kernel size of $n \times n$ pixels to fill small holes in activation areas.
· remove deactivation areas smaller than β pixels by applying a speckle filter to the inverted DOG mask $\mathcal{M}^{\text{DoG-inv}} = 1 - \mathcal{M}^{\text{DoG}}$.
· invalidate pixels in $\mathcal{U}$ or $\mathcal{D}$ for which $\mathcal{M}^{\text{DoG}} = 1$.

2.3.2.2 Difference-of-Gaussian Filter

As proposed by Wenzel (2016), the *difference-of-Gaussian (DOG)* filter allows to remove estimates by masking out pixels in image regions that only provide little textural information (e.g. unsharp or overexposed areas). Here, it is assumed that in such regions the image matching is ambiguous and that it leads to less accurate results. The DOG filter is used to detect weakly-textured areas inside the reference image $\mathcal{I}_{\text{ref}}$ and construct a binary image mask, which, in turn, is used to remove the estimates from the corresponding maps. Algorithm 2.2 provides an overview on the implementation of the DOG filter used in the scope of this work, which is similar to the one proposed in (Wenzel 2016). (Ruf et al. 2021a, Sec. 2.6.1)

2.3.3 Geometric Consistency Check and Occlusion Detection

Given two or more disparity or depth maps, a consistency check can be performed in order to filter out estimates that are geometrically inconsistent or occluded, and thus not reliable. Section 2.3.3.1 gives an overview on the left-right consistency check, which is typically employed when estimating a disparity map from a two-view stereo setup and allows to detect occluded areas. In Section 2.3.3.2, a geometric consistency check is introduced, which relies on multiple depth maps from different views and allows to filter estimates which are inconsistent between the different views or between different time steps, if an image sequence is considered as input.

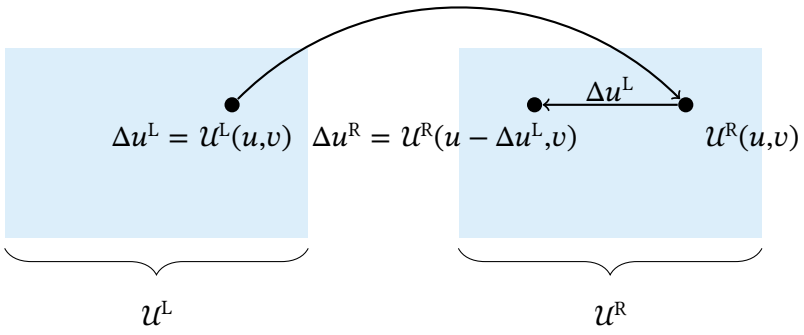


Figure 2.5: Illustration of the left-right consistency check to find occluded image regions when estimating a disparity map from a two-view stereo setup. Given the disparity maps $\mathcal{U}^{\mathrm{L}}$ and $\mathcal{U}^{\mathrm{R}}$ for the reference and matching image, respectively, a consistency check according to Equation (2.13) can be performed in order to find occlusions. (Ruf et al. 2021b, Fig. A2.)

2.3.3.1 Left-Right Consistency Check

The existence of occluded pixels that arise from areas, which are observed by one camera and yet occluded in the field-of-view of the other camera, is inherent to disparity and depth maps that are computed from a conventional stereo setup consisting of only two cameras. A typical approach to detect and filter such pixels is the *left-right consistency check*. As illustrated by Figure 2.5, the disparities that are stored in the disparity map of the left image $\mathcal{U}^{\mathrm{L}}$ are compared with the corresponding disparities in the right disparity map $\mathcal{U}^{\mathrm{R}}$ and invalidated if they differ by a certain threshold, usually 1:

$$\mathcal{U}(\mathrm{p}) = \begin{cases} \mathcal{U}^{\mathrm{L}}(\mathrm{p}), & \text{if } |\mathcal{U}^{\mathrm{L}}(\mathrm{p}) - \mathcal{U}^{\mathrm{R}}(p_u - \mathcal{U}^{\mathrm{L}}(\mathrm{p}), p_v)| \leq 1 \\ 0, & \text{otherwise.} \end{cases} \tag{2.13}$$

(Ruf et al. 2021b, Appendix B.)

While the occlusion detection using the left-right consistency check is very effective, it requires the computation of a second disparity map $\mathcal{U}^{\mathrm{R}}$, which corresponds to the right image of the stereo pair. A straight-forward approach to compute $\mathcal{U}^{\mathrm{R}}$ would be to swap and horizontally flip the input images and repeat the image matching, cost optimization and disparity computation as described above. This, however, would mean to execute the first and computationally most expensive steps of the processing pipeline twice for each image pair. Yet, the computation of $\mathcal{U}^{\mathrm{R}}$ can be efficiently approximated by reusing the aggregated cost volume $\bar{\mathcal{S}}$ from the cost optimization step:

$$\mathcal{U}^{\mathrm{R}}(\mathrm{p}) = \underset{\Delta u \in \Gamma_u}{\arg\min}\, \bar{\mathcal{S}}((p_u + \Delta u, p_v), \Delta u). \tag{2.14}$$

(Ruf et al. 2021b, Sec. 2.1.3.)

2.3.3.2 Multi-View Consistency Check

When multiple depth maps $\mathcal{D}_k$ with corresponding projection matrices $\mathbf{P}_k$ are available, for example when performing reconstruction by MVS or when considering an image sequence as input and a temporal consistency is to be established, a geometric consistency check can be performed by relying on the mutual reprojection error. In this, each pixel $\mathrm{p}^{\mathrm{ref}}$ of a selected reference depth map $\mathcal{D}_{\mathrm{ref}}$, having a depth estimate $d_{\mathrm{p}}^{\mathrm{ref}}$, is projected into the view another depth map $\mathcal{D}_k$, according to $d_{\mathrm{p}}^{\mathrm{ref}}$ and the corresponding projection matrices $\mathbf{P}_{\mathrm{ref}}$ and

$\mathbf{P}_k$, yielding the image point p^k. Given p^k and the corresponding depth d_p^k from $\mathcal{D}_k$, the image point p^k is projected back into the view of $\mathcal{D}_\mathrm{ref}$, resulting in $\tilde{\mathrm{p}}^\mathrm{ref}$. Finally, if the Euclidean distance between p^ref and $\tilde{\mathrm{p}}^\mathrm{ref}$ exceeds a given threshold η_r, the estimate at p^ref is invalidated. (Ruf et al. 2021a, Sec. 2.6.2.)

Schönberger et al. (2016) formulate this reprojection error for pixel p in a reference view and a neighboring map k as $\epsilon_\mathrm{r}^k(\mathrm{p}) = \left|\mathrm{p} - \mathbf{H}_\mathrm{p}^k \cdot \mathbf{H}_\mathrm{p} \cdot \mathrm{p}\right|$, with $\mathbf{H}_\mathrm{p}$ denoting the forward projection into the view k according to $d_\mathrm{p}^\mathrm{ref}$, and $\mathbf{H}_\mathrm{p}^k$ representing the corresponding backward projection according to d_p^k (cf. Equation (2.7)). In both cases, a fronto-parallel plane orientation with $\mathrm{N}_\Pi = (0\ 0\ {-1})^\intercal$ is assumed. Apart from thresholding the reprojection error ϵ_r^k, the number of neighboring views for which the reprojection error is within the threshold, i.e. the number of hits, are counted: $\epsilon_\mathrm{h}(\mathrm{p}) = \sum_k [\epsilon_\mathrm{r}^k(\mathrm{p}) < \eta_\mathrm{r}]$, with $[\cdot]$ being the Iverson bracket. This approach can be adopted to perform a geometry-based filtering between a number of depth maps within a sliding window Ψ. Algorithm 2.3 gives an overview on the geometric consistency and the according filtering of the depth map. (Ruf et al. 2021a, Sec. 2.6.2.)

Algorithm 2.3: The geometric consistency check for multiple depth maps from different views. (Ruf et al. 2021a, Alg. 4.)

Input Data: depth maps $\mathcal{D}_k$ within a sliding window Ψ, as well as corresponding projection matrices $\mathbf{P}_k$.
Result: filtered $\mathcal{D}_\mathrm{ref}$ of the reference view, in which all estimates that are geometrically not consistent are removed.

· select $\mathcal{D}_\mathrm{ref}$ corresponding to the center-most view within the sliding window Ψ.
foreach *pixel* $\mathrm{p}^\mathrm{ref} \in \mathcal{D}_\mathrm{ref}$ **and** *neighboring view* $k \in \Psi$ **do**
  · calculate the number of hits for which the reprojection error is below a threshold η_r:

$$\epsilon_\mathrm{h}(\mathrm{p}) = \sum_k [\epsilon_\mathrm{r}^k(\mathrm{p}) < \eta_\mathrm{r}], \text{ with } \epsilon_\mathrm{r}^k(\mathrm{p}) = \left|\mathrm{p} - \mathbf{H}_\mathrm{p}^k \cdot \mathbf{H}_\mathrm{p} \cdot \mathrm{p}\right|.$$

  · if the hit-count is below a threshold η_h, i.e. $\epsilon_\mathrm{h} < \eta_\mathrm{h}$, invalidate pixel p in $\mathcal{D}_\mathrm{ref}$, by setting it to 0.
end

3 Real-Time Stereo Disparity Estimation on Embedded On-Board Devices

This chapter includes material from the following publication:

Ruf, B.; Mohrs, J.; Weinmann, M.; Hinz, S., and Beyerer, J. (2021b): "ReS^{2}tAC – UAV-borne real-time SGM stereo optimized for embedded ARM and CUDA devices". In: *MDPI Sensors* 21.11, 3938. Reprinted with permission. It is cited as (Ruf et al. 2021b) and marked with a blue sidebar.

As outlined in Section 1.2, the first part of the proposed framework aims at facilitating a safe and autonomous use of COTS UAVs by means of real-time on-board obstacle detection and collision avoidance. In order to perceive their surroundings, modern UAVs are often equipped with a range of different sensors, e.g. ultrasonic and stereo vision sensors. In the scope of this work, the stereo vision sensor is used to compute a dense disparity map that serves as an input to the algorithm for obstacle detection and collision avoidance (cf. Section 6.1). Compared to active sensors like LiDAR scanners, camera systems in combination with state-of-the-art algorithms are typically more practical in performing these tasks, especially in terms of costs, weight and power consumption. Moreover, such stereo vision sensors are often already integrated in COTS UAVs. On the other hand, while a LiDAR sensor directly provides data on the 3D geometry of the scene, using a stereo camera for the same task requires to process the stereo image data and perform a disparity or depth estimation (Nex and Rinaudo 2011). This, in turn, requires a high-performance embedded processing on-board the UAV. (Ruf et al. 2021b, Sec. 1.)

For a long time, so-called FPGAs were the only processing hardware, that were capable of high-performance computing, while at the same time preserving a low power consumption, essential for embedded systems. In recent years, however, the availability of embedded GPUs, such as the NVIDIA Tegra, allows for massively parallel embedded computing on graphics hardware, which is typically more flexible than FPGAs and less cumbersome to implement. Furthermore, with the increasing use of deep learning for a wide range of applications, the importance and availability of embedded GPUs have grown even more. With the Jetson boards, comprised of an embedded ARM CPU and an embedded Tegra GPU, NVIDIA provides a suitable alternative to FPGAs for embedded high-performance computing. Especially, since these systems are recently also being integrated in COTS UAVs, such as the DJI Matrice series in combination with the DJI Manifold or the UVify IFO-S UAV. (Ruf et al. 2021b, Sec. 1.)

3.1 Contributions and Outline

In this chapter, an approach for real-time embedded stereo processing on ARM- and CUDA-enabled devices is presented. It is based on the well-known and widely used SGM algorithm (cf. Section 2.2.1) first proposed by Hirschmüller (2005) and Hirschmüller (2008), and it is further denoted as *Real-Time SGM Stereo Optimized for Embedded ARM and CUDA Devices (ReS^{2}tAC)*. Its main contributions are:

- the optimization of the SGM algorithm for embedded CUDA GPUs, such as the NVIDIA Tegra, by utilizing massively parallel computing,
- the use of the NEON intrinsics to optimize the algorithm for vectorized SIMD processing on embedded ARM CPUs, and
- the deployment on the DJI Manifold 2-G attached to a DJI Matrice 210v2 RTK UAV and a use-case-specific evaluation with respect to accuracy, processing speed and power consumption.

Even though ReS^{2}tAC is deployed and tested for real-time processing on board a UAV, it is also suitable for other embedded systems, such as those deployed on ground-based robots or those used in an advanced driver assistance system (ADAS). (Ruf et al. 2021b, Sec. 1.)

The contributions presented in this chapters have been published in:

- Ruf, B.; Mohrs, J.; Weinmann, M.; Hinz, S., and Beyerer, J. (2021b): "ReS^{2}tAC – UAV-borne real-time SGM stereo optimized for embedded ARM and CUDA devices". In: *MDPI Sensors* 21.11, 3938. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2021b).

In the following, the related work on embedded stereo processing utilizing embedded FPGA, GPU or CPU hardware is briefly summarized in Section 3.2. Here, it is also pointed out how ReS^{2}tAC differs from the approaches found in literature. In Section 3.3, the processing pipeline of ReS^{2}tAC is outlined with detailed descriptions on the optimization for parallel processing on CUDA-enabled GPUs and on the vectorized SIMD processing with NEON intrinsics for ARM CPUs in Section 3.3.1 and Section 3.3.2, respectively. The experimental results are presented and discussed in Section 3.4 and Section 3.5, respectively. A summary and concluding remarks are finally provided in Section 3.6.

3.2 Related Work

The following sections summarize the related work on embedded stereo processing. Section 3.2.1 looks into studies that have deployed stereo algorithms on FPGA hardware. This is followed by an overview on the emergence of embedded GPU hardware for real-time stereo processing in Section 3.2.2. Lastly, in Section 3.2.3, the related work on deploying real-time stereo processing on CPU hardware, both for high-end desktop and embedded environments, is discussed. In addition, it is pointed out how ReS^{2}tAC differs from the related work utilizing embedded GPU and CPU hardware for real-time stereo processing. A relevant excerpt of the related work on real-time SGM stereo processing on FPGA, GPU and CPU hardware is summarized in Table 3.1. (Ruf et al. 2021b, Sec. 1.2.)

3.2.1 Embedded Stereo Processing on FPGAs

The use of FPGAs is key to achieve high-performance image processing with minimal power consumption, especially when relying on computationally expensive algorithms. Thus, most implementations of stereo algorithms for embedded systems, in particular of the SGM algorithm (Hirschmüller 2005, Hirschmüller 2008), are based on FPGA technology. First optimizations of the SGM algorithm, such as those presented in (Gehrig et al. 2009, Banz et al. 2010), were deployed on a PCIe-FPGA card inside a conventional PC or on a separate development kit, achieving real-time frame rates of 27 FPS and 30 FPS on low-resolution imagery, i.e. images with a size of 320×240 pixels and 640×480 pixels, respectively. Due to ongoing technological advancements,

the implementation of Wang et al. (2015), deployed on an Altera Stratix-IV FPGA-Board, already achieved a frame rate of 67 FPS on images with a size of 1024×768 pixels in 2015. However, besides the dedicated and specialized processing of a specific task, typical characteristics of embedded systems are a small form-factor and the integration in larger systems or cooperative environments. (Ruf et al. 2021b, Sec. 1.2.1.)

In their work, Schmid et al. (2013) have deployed the implementation of Gehrig and Rabe (2010) on a small rotor-based UAV for stereo-vision-based navigation achieving 14.6 FPS on a Spartan 6 FPGA. Further system-on-a-chip (SOC) developments with respect to size and performance allowed to deploy computationally expensive algorithms on increasingly smaller systems with higher performance. Honegger et al. (2014) implemented the SGM algorithm on a small SO-DIMM sized SOC equipped with a Xilinx Artix7 FPGA and reaching 60 FPS with a frame size of 753×480 pixels. By reducing the frame size to 320×240 pixels, the implementation of Barry et al. (2015) reached 120 FPS and was used to navigate a small and fast flying fixed-wing UAV around obstacles. (Ruf et al. 2021b, Sec. 1.2.1.)

A number of recent studies (Hofmann et al. 2016, Rahnama et al. 2018a, Zhao et al. 2020) have shown, that further optimizations, such as reducing the number of processing paths in the SGM optimization or increasing parallelization by splitting the input images in independent stripes, as well as the technological advancements, allow to reach frame rates of over 100 FPS, while at the same time increasing the accuracy of the stereo algorithm by using a higher image resolution and reducing the form-factor, leading to a reduced power consumption of the SOC. Yet, the use of FPGAs for real-time embedded image processing involves a cumbersome and time-consuming development, optimization and deployment process. In order to reduce development costs of such systems, substantial effort is done to enhance the process of high-level synthesis (HLS) and, in turn, facilitate the development of algorithms for FPGAs with more high-level languages such as C/C++ (Ruf et al. 2018b, Kalb et al. 2019, Zhao et al. 2020). (Ruf et al. 2021b, Sec. 1.2.1.)

3.2.2 On the Emergence of Embedded Processing on GPUs

The development cycles for implementing and optimizing image processing algorithms for massively parallel processing on GPUs, on the other hand, are much shorter and thus less expensive. In addition, GPUs provide a much higher processing power, ideal for algorithms with high computational effort, such as stereo image processing. Early works (Rosenberg et al. 2006, Ernst and Hirschmüller 2008) have utilized the rendering pipeline of OpenGL to deploy the SGM stereo algorithm on graphics hardware and reached frame rates of up to 8 FPS and 4 FPS on VGA image resolution, respectively. With the introduction of the CUDA-API in 2007, the development costs for general-purpose computation on a GPU (GPGPU) have dropped even more. And so, a lot of implementations of the SGM algorithm for real-time stereo processing on hardware without embedded constraints have been optimized and deployed on graphics hardware (Banz et al. 2011, Michael et al. 2013, Hernandez-Juarez et al. 2016). Hernandez-Juarez et al. (2016) show that, with increasing computational power, the use of GPGPU on modern, high-performing graphics hardware, such as the NVIDIA Titan X, allows to reach frame rates of up to 237 FPS on VGA image resolution with the conventional use of eight optimization paths inside the SGM optimization. Even higher frame rates of up to 475 FPS and 886 FPS are possible, if the number of optimization paths are reduced to four or two, respectively. (Ruf et al. 2021b, Sec. 1.2.2.)

While GPUs provide great computational performance, a major drawback is given by their high power consumption. The deployment of the SGM algorithm on a high-end GPU only achieves 1.90 FPS/W on VGA image resolution (Hernandez-Juarez et al. 2016), while a comparable configuration of the algorithm deployed on a

Table 3.1: Overview of numerous studies that have developed and deployed an SGM-based stereo algorithm on different embedded hardware architectures. The highest frame rates, with respect to the corresponding power consumption, are achieved by FPGA-based implementation. ∗: Power consumption stated for the whole system, e.g. including image capture. †: Studies state measurements for different configurations, however, the configuration listed provides a good trade-off between image resolution and frame rate. (Ruf et al. 2021b, Tab. 1.)

Reference	HW Device	Embedded SOC	Resolution	Disp. Range	FPS	Power
Gehrig et al. (2009)	FPGA		320 × 240	64	27	< 3 W
Banz et al. (2010)	FPGA		640 × 480	128	30	n/a
Honegger et al. (2014)	FPGA	✓	753 × 480	32	60	< 5 W
Wang et al. (2015)†	FPGA		1024 × 768	96	67	n/a
Barry et al. (2015)	FPGA	✓	320 × 240	32	120	< 5 W ∗
Hofmann et al. (2016)†	FPGA	✓	640 × 480	64	140	n/a
Ruf et al. (2018b)	FPGA	✓	640 × 360	64	29	n/a
Rahnama et al. (2018a)†	FPGA	✓	640 × 480	128	109	< 3 W
Zhao et al. (2020)†	FPGA	✓	1242 × 374	128	161	6.6 W
Banz et al. (2011)†	GPU		1024 × 768	128	25	n/a
Michael et al. (2013)	GPU		640 × 480	64	11.7	n/a
Hernandez-Juarez et al. (2016)†	GPU	✓	640 × 480	128	42	< 10 W
ReS^{2}tAC-CUDA (Ruf et al. 2021b)	GPU	✓	640 × 480	128	24	~ 20 W ∗
Gehrig and Rabe (2010)†	CPU		640 × 320	16	14	n/a
Arndt et al. (2013)†	CPU	✓	640 × 480	64	0.5	n/a
Spangenberg et al. (2014)†	CPU		640 × 480	128	16	n/a
ReS^{2}tAC-NEON (Ruf et al. 2021b)	CPU	✓	640 × 480	128	7.2	~ 18 W ∗

state-of-the-art embedded FPGA achieves 15 FPS/W on a larger image resolution (Zhao et al. 2020). However, due to the increasing importance of deep learning algorithms for robotic applications, more and more embedded CPU-GPU-based SOCs are released and integrated into robotic systems. Recently, such SOCs with embedded GPUs have even been integrated onto COTS UAVs, and thus been made available for the mainstream user. In their work, Hernandez-Juarez et al. (2016) have also deployed their implementation on the NVIDIA Jetson TX1, encapsulating the embedded Tegra X1 GPU, reaching 42 FPS (4.19 FPS/W) on VGA resolution and a 4-path SGM optimization. Chang et al. (2020) further optimized the computation of the normalized cross correlation (NCC) matching cost for the use on GPUs and deployed their SGM-based stereo algorithm on the NVIDIA Jetson TX2 reaching 28 FPS on images with a size of 1242 × 375 pixels. (Ruf et al. 2021b, Sec. 1.2.2.)

In the scope of ReS^{2}tAC, the SGM algorithm is also optimized for real-time processing on CUDA devices and deployed on the NVIDIA Jetson TX2 and the more powerful NVIDIA Jetson Xavier AGX. In this, the performance of different configurations and optimization strategies with respect to performance and power consumption is evaluated. The computationally most expensive part of the SGM algorithm is the aggregation along the different scanlines. At the same time, due to the nature of dynamic programming, this is also the part which can be parallelized most effectively, since the computation of each scanline can be done fully independently, without the need of synchronization. In terms of GPGPU, this is typically done by instantiating one thread on the graphics hardware for each scanline, resulting in a massively parallel processing of the SGM path aggregation. This is also adopted by ReS^{2}tAC as described in Section 3.3.1. (Ruf et al. 2021b, Sec. 1.2.2.)

3.2.3 Are Embedded CPUs Suitable for Stereo Processing?

In contrast to FPGAs and GPUs, which are designed to be less flexible and yet very powerful in processing specific tasks on a large amount of data, CPUs are designed to do more general and versatile processing, needed to allow computers to instantly react to new sensor input. Even though they have much higher clock frequencies, CPUs are often not capable to keep up with the performance achieved by their more specialized counterparts, due to their small number of cores and, in turn, limited ability of parallelization. (Ruf et al. 2021b, Sec. 1.2.3.)

One of the first deployments of the SGM algorithm on a conventional CPU was done by Gehrig and Rabe (2010). They have implemented a number of different parallelization techniques, among others splitting the 8-path optimization scheme into two independent scans. With this, they achieve 14 FPS on images with a size of 640×320 pixels, but they only considered a range of 16 disparities. They have deployed their algorithm on an Intel Core i7 with four cores and a clock frequency of 3.3 GHz. A few years later, Spangenberg et al. (2014) have achieved 16 FPS on VGA resolution and 128 disparities, also running their implementation on a conventional Intel Core i7 with four cores. Apart from a number of algorithmic optimizations, e.g. disparity space compression and striped computation, they have parallelized the processing by utilizing single-instruction-multiple-data (SIMD) vectorization with the SSE instruction set from Intel combined with multi-threading. (Ruf et al. 2021b, Sec. 1.2.3.)

The work of Arndt et al. (2013) is one of the first to deploy an implementation of the SGM algorithm on an embedded CPU, namely the Freescale P4080, reaching a frame rate of only 0.5 FPS on VGA image resolution. When considering that this was done around the same time as the works of Gehrig and Rabe (2010) and Spangenberg et al. (2014), it illustrates the gap between conventional and embedded CPUs in terms of performance. However, the embedded CPU technology has also evolved and gained performance. Most of the modern high-end embedded SOCs typically consist of an ARM CPU and a FPGA or a GPU, for example the Xilinx Ultrascale series or the NVIDIA Jetson series. Rahnama et al. (2018b) implemented the ELAS stereo algorithm (Geiger et al. 2011) on a Xilinx ZC706 SOC made up of an ARM CPU and a FPGA. In this, they have deployed the computationally most expensive stages of the algorithm onto the FPGA (if possible) and used the ARM CPU to process the stages with unpredictable memory access patterns. Saidi et al. (2020) accelerated a simple stereo algorithm on an ARM CPU, achieving frame rates of up to 59 FPS on an image resolution of 320×240 pixels. In this, they have parallelized the algorithm by using multi-threading as well as SIMD vectorization on ARM with the NEON instruction set (ARM 2013). (Ruf et al. 2021b, Sec. 1.2.3.)

Just as the dynamic programming in the SGM path aggregation is well-suited for massively parallel computing on graphics hardware, its computation can also be effectively vectorized by SIMD processing as Spangenberg et al. (2014) have shown. With this in mind, this work investigates the ability of embedded CPUs in performing high-accuracy stereo processing in real-time based on the SGM algorithm. So far, with ReS^{2}tAC, this work is the first to implement and optimize the Semi-Global Matching stereo algorithm on an embedded ARM CPU, leveraging multi-threading parallelization and SIMD vectorization with NEON intrinsics. (Ruf et al. 2021b, Sec. 1.2.3.)

3.3 ReS^{2}tAC – Real-Time SGM Stereo Optimized for Embedded ARM and CUDA Devices

As illustrated in Figure 3.1, the processing pipeline of ReS^{2}tAC is divided into three subsequent steps, which in turn are made of smaller building blocks. It follows the general pipeline for two-view stereo disparity and depth estimation as outlined by Figure 2.2. Given the images of a calibrated stereo camera system, it first rectifies the two input images according to the transformation maps that are pre-computed in the scope of calibrating the stereo setup. With the two rectified input images, dense image matching (DIM) and cost computation are performed (cf. Section 2.1), followed by a cost optimization by means of SGM (cf. Section 2.2.1). Lastly, disparity refinement, left-right consistency check and median filtering are executed in a post-processing (cf. Section 2.3) to produce the final output. In the following, detailed descriptions on the optimization of this processing pipeline for GPGPU on CUDA-enabled GPUs and on the vectorized SIMD processing with NEON intrinsics for ARM CPUs are given in Section 3.3.1 and Section 3.3.2, respectively.

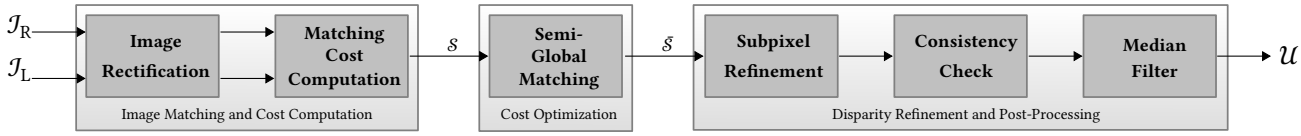


Figure 3.1: Overview of the processing pipeline of Real-Time SGM Stereo Optimized for Embedded ARM and CUDA Devices (ReS^{2}tAC). Given two input images from a stereo camera, ReS^{2}tAC computes a dense disparity map by means of dense image matching (DIM) and Semi-Global Matching (SGM), followed by a series of post-processing steps. (Ruf et al. 2021b, Fig. 1.)

3.3.1 Real-Time Processing by Massively Parallel Computing on CUDA-Enabled GPUs

Graphic processing units are designed for massively parallel processing. The number of processing units that are integrated into modern GPU hardware exceeds the number of cores available on a conventional CPU by far. Even embedded GPUs, like the one built into the NVIDIA Jetson Xavier AGX, have up to 100× more cores than high-end desktop CPUs. However, the processing units on a GPU are less powerful and flexible than those of a CPU. While the CPU is designed to do arbitrary processing tasks, the cores of the GPUs are intended for the simultaneous and parallel processing of small and dedicated instructions on a large amount of data. (Ruf et al. 2021b, Appendix C.)

Massively parallel general-purpose processing on NVIDIA GPUs is alleviated by the CUDA-API, which allows the deployment of routines and functions for data processing in the form of so-called CUDA kernels on the GPU. When deployed, such a kernel is instantiated inside a high number of threads, which are being distributed among the available processing units and are each processing a different subset of the data. For a better abstraction and handling, the threads are logically grouped into thread blocks, which share a local memory space and can thus exchange data between each other. Furthermore, the execution of all threads within one thread block can be halted and synchronized. All thread blocks are grouped into a grid. The grid size and the number of threads inside a grid are used to parameterize the instantiation of the kernel. The actual execution of the threads on a processing unit is always done in groups of 32, which is referred to as a CUDA warp. For an efficient GPGPU, the developer is compelled to consider some design guidelines:

- Reduction of global memory access due to higher latency. Instead, data which gets processed multiple times by threads in a thread block should be cached inside the shared memory space.

- Pooling of global memory access and reduction of non-contiguous data storage.
- Efficient and maximum utilization of hardware resources.

(Ruf et al. 2021b, Appendix C.)

In the scope of ReS²tAC, each step of the stereo algorithm (cf. Figure 3.1) is implemented in separate CUDA kernels in order to optimize the stereo algorithm for embedded NVIDIA GPUs. Since each kernel execution is aimed to achieve a high utilization of the GPU, a parallel execution of the CUDA kernel methods with CUDA streams is not implemented. The following sections provide a detailed description on how each step of the algorithm for an efficient GPGPU is optimized and instantiated. (Ruf et al. 2021b, Sec. 2.2.)

3.3.1.1 Matching Cost Computation

As already mentioned in Section 2.1, the two most commonly used matching cost functions are the Hamming distance of the census transform (CT) as well as an inverted and truncated version of the NCC, which, in turn, are also used by ReS²tAC. A detailed description on the implementation and the optimization of these two cost functions for execution on a GPU with CUDA is given in the following. (Ruf et al. 2021b, Sec. 2.2.1.)

The Census Transformation and its Hamming Distance

For the parallel calculation of the CT on the GPU, different image regions are assigned to each instantiation of the corresponding CUDA kernel. Before the actual CT is calculated, each kernel instantiation copies the pixel data of the considered image region into shared memory in order to achieve a higher access speed. The computation of the CT is then performed in parallel by each thread of the thread block for a specific pixel in the assigned image region. To account for pixels at the image border, where a part of the neighborhood lies outside the image, a zero-valued margin with half the size of the neighborhood is assigned to the image for the calculation of the CT. Furthermore, the calculation of the Hamming distance is separated from the calculation of the CT, since the two kernel methods of these two steps are instantiated with different parameters. For the parallelization on the GPU, a CT with a neighborhood size of 5×5 pixels and 9×7 pixels is implemented, the latter being the largest neighborhood that fits into a 64-bit integer. The choice of the former neighborhood size is justified by the limitations of the optimized implementation for the CPU (cf. Section 3.3.2.1). An overview of the implementation is illustrated by Figure 3.2 (top). (Ruf et al. 2021b, Sec. 2.2.1.)

In the calculation of the Hamming distance, the reference and matching image are divided into stripes and each stripe is assigned to different CUDA warps. Each thread of the CUDA warps then calculates the Hamming distance from the corresponding census descriptors at a certain pixel position and disparity. The dimensions of the CUDA warps are chosen in such a way that for 16 different pixels half the disparities can be calculated simultaneously (Figure 3.2 (bottom)). In this, the census descriptor of the reference pixels is first loaded by all threads of a thread block into the shared memory. Then, all threads load the census descriptors of the matching pixel given a certain disparity. This means that the 32 threads of a thread block load the census descriptors $CT(\mathcal{I}_R(u-i,v)), \ldots, CT(\mathcal{I}_R(u-i-31,v))$ from the matching image into the shared memory. This is repeated until for all disparities $[0, \Delta u_{max}]$ the census descriptors of the matching image are loaded into the shared memory. Given all the census descriptors, the threads of a thread block compute the Hamming distance simultaneously for different pixels and store the result at the corresponding position in the cost volume. Since the matching costs of the different disparities for one pixel lie directly next to each other in the cost volume,

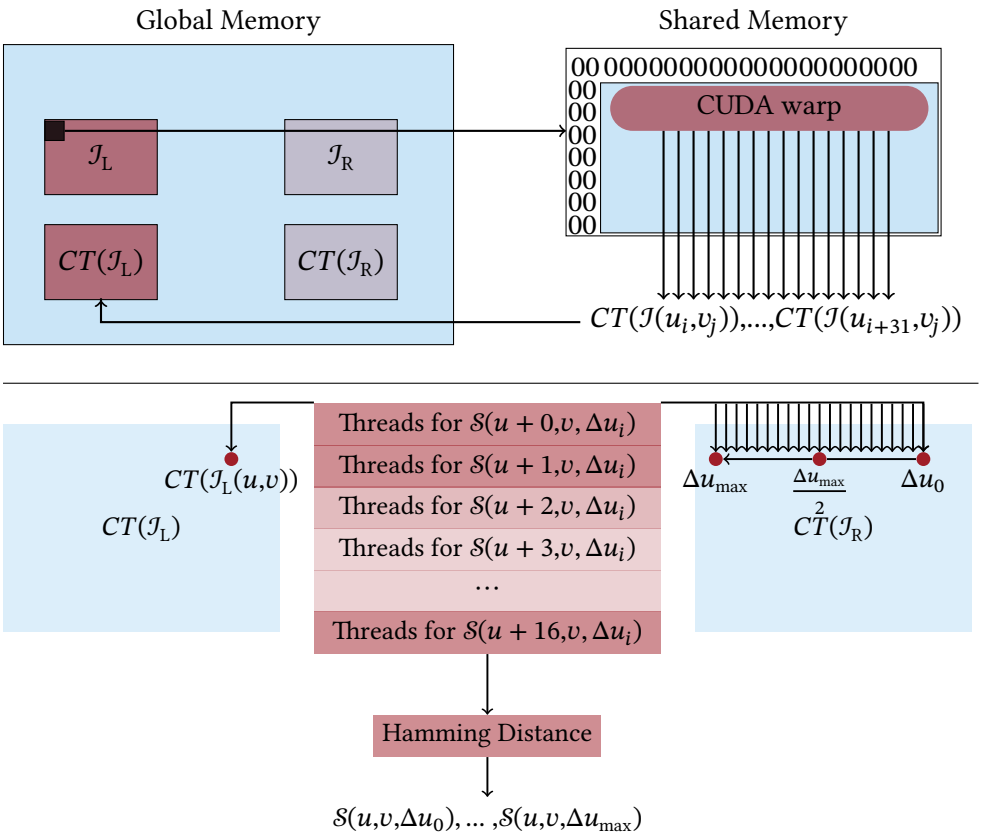


Figure 3.2: **Top:** To calculate the CT by a CUDA warp for a specific image region, the image data of this region is first copied from the global memory to the shared memory, the latter having higher access speeds. In the second step, each thread of the CUDA warp calculates the CT for one pixel inside this region. **Bottom:** The threads of a CUDA warp calculate the Hamming distance for 16 disparities simultaneously. (Ruf et al. 2021b, Fig. 2.)

the dimension of the CUDA warp is chosen in such a way that the memory access from the GPU can be pooled together. (Ruf et al. 2021b, Sec. 2.2.1.)

Inverted and Truncated Version of the Normalized Cross Correlation

Using the NCC as a cost function is computationally more expensive than relying on the Hamming distance of the CT. While the computation of the CT and the subsequent Hamming distance only requires some comparative and bit-level operations, the computation of the NCC needs the calculation of a mean and variance of the two input patches. Since the mean and the variance of all possible patches inside an image can be precomputed and then reused, the calculation of the NCC and the process of image matching is divided into two separate stages, similar to the calculation of the CT and the Hamming distance. (Ruf et al. 2021b, Sec. 2.2.1.)

In the first step, the mean and variance for all patches in the left and right input image are calculated. In this, a kernel with the same configuration as when calculating the CT is instantiated, iterating over all pixels in the left and right image, and the necessary data for a patch of a given size, centered around the current pixel, is computed. Similar to the process of computing the CT, the input images are thus first enhanced with the independent patch information, storing the patch-mean and patch-variance together with the pixel value of the center pixel in a special struct for each pixel of the input image. Just as when calculating the Hamming distance of the census transform, the pixel and the patch data are then used to perform the image matching based on the inverted and truncated normalized cross correlation and the resulting cost volume is filled in the second stage.

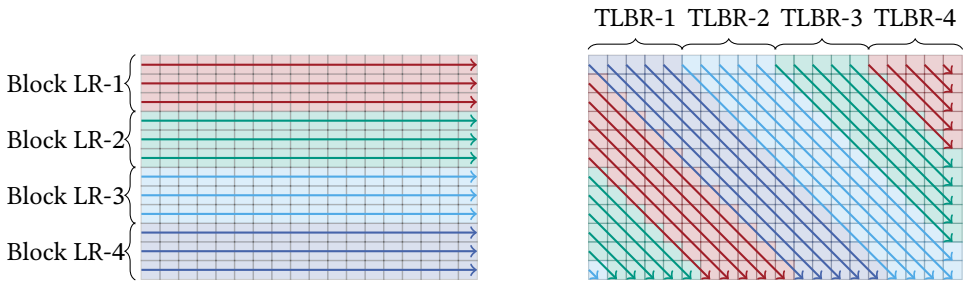


Figure 3.3: Illustration of the SGM path aggregation optimized for massively parallel processing on CUDA-enabled GPUs. Multiple CUDA blocks processing the SGM path aggregation. Each block calculates 16 different lines along one path direction. If a diagonal line reaches the image border, the values are reset and the calculation is resumed on the other side of the image. (Ruf et al. 2021b, Fig. 3.)

Again, a kernel with the same parameters as for the image matching with the Hamming distance is instantiated. The NCC is implemented for a patch size of 5×5 pixels and 9×9 pixels. (Ruf et al. 2021b, Sec. 2.2.1.)

3.3.1.2 Semi-Global Matching Optimization

The calculation of the eight different SGM path costs is done sequentially on CUDA hardware. The parallelization of the cost aggregation on one path direction is realized on two different levels. First, each CUDA block calculates the costs for 16 different lines along one path direction (Figure 3.3). If a diagonal line reaches the image border, the values are reset and the calculation is resumed on the other side of the image. This ensures that all calculations of one path direction take the same time. Additionally, for each image point, the costs for the disparities $\Delta u_0, \dots, (\frac{\Delta u_{\max}}{2} - 1)$ and $\frac{\Delta u_{\max}}{2}, \dots, \Delta u_{\max}$ are calculated in parallel by two iterations. This ensures that all threads within one warp access a contiguous area in the memory, allowing the memory transactions to be more efficient. However, this requires a synchronization of the threads within a CUDA warp after the

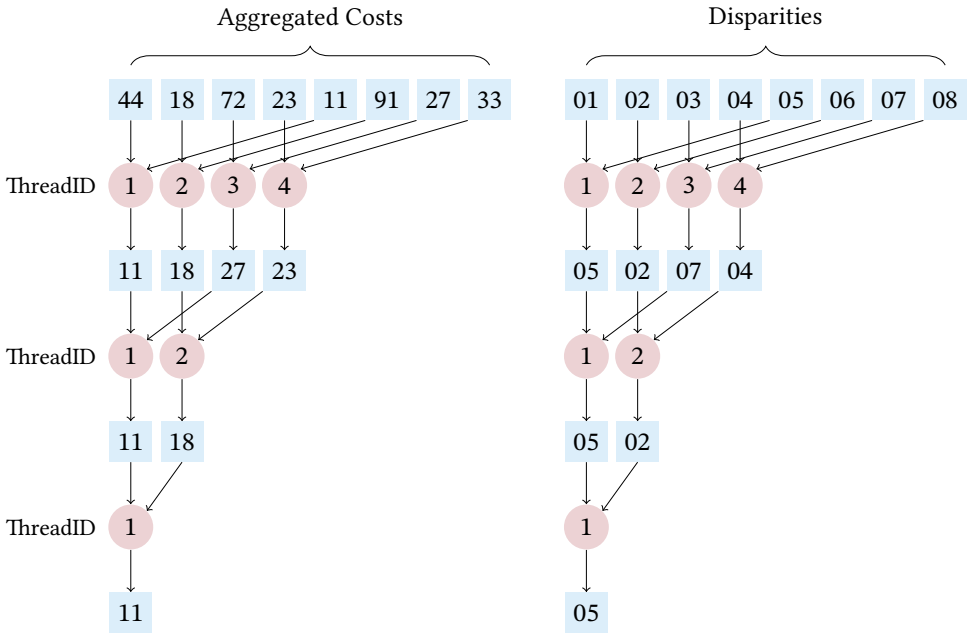


Figure 3.4: Illustration of the MapReduce method to find the WTA disparity with the minimum aggregated cost. Each active thread performs a comparison between two elements and stores the smaller element in a designated memory space. In each iteration, the number of aggregated costs that are being processed and the corresponding disparities are halved. (Ruf et al. 2021b, Fig. A3.)

costs for all disparities have been calculated, in order to find the minimum path cost, which is necessary for further processing. (Ruf et al. 2021b, Sec. 2.2.2.)

To find the minimum of the aggregated costs, the MapReduce (Dean and Ghemawat 2008) method is utilized. In this, each active thread performs a comparison between two elements and stores the smaller element in a designated memory space. The number of active threads as well as the number of elements that are to be processed get halved by each iteration as illustrated by Figure 3.4. Thus, the search for the minimum cost requires $\log_2(\Delta u_{\max})$ iterations. Ideally, the dimensions of the CUDA blocks are chosen in such a way, that there are 32 active threads at the beginning. This allows that the MapReduce algorithm can be processed in only one warp. After the calculation and aggregation of the different SGM path costs, the WTA disparity with the minimum aggregated costs is to be found. This is done by assigning a specific image region to each CUDA warp and again utilizing the MapReduce method mentioned above. (Ruf et al. 2021b, Appendix D.)

3.3.1.3 Consistency Check

The key aspect in the consistency check is the calculation of the approximated disparity map $\mathcal{U}^{\mathrm{R}}$ corresponding to the matching image. This is approximated from the calculated aggregated cost volume $\bar{\mathcal{S}}$ of the reference image. In this, each entry of $\mathcal{U}^{\mathrm{R}}$ is calculated according to Equation (2.14). The difficulty, that arises in this process, is the access of non-adjacent areas in $\bar{\mathcal{S}}$, as illustrated by Figure 3.5. Between all entries of the aggregated cost volume $\bar{\mathcal{S}}$, that are to be used for the calculation of $\mathcal{U}^{\mathrm{R}}$, always lie $\Delta u_{\max} + 1$ entries, which are of no interest. (Ruf et al. 2021b, Sec. 2.2.3.)

In the GPU implementation for the consistency check, each instantiation of a CUDA kernel computes a specified region in the approximated disparity map $\mathcal{U}^{\mathrm{R}}$. Here, the data of the aggregated cost volume is first copied into shared memory for quicker access. Since the data does not lie next to each other, the access to the global memory by the different CUDA threads cannot be pooled together. When all data is available in the shared memory, the MapReduce method is again used to find the minimum. After the disparity map $\mathcal{U}^{\mathrm{R}}$ for the matching image is approximated, each thread performs the consistency check according to Equation (2.13) for a specific pixel in the final disparity map $\mathcal{U}^{\mathrm{L}}$. (Ruf et al. 2021b, Sec. 2.2.3.)

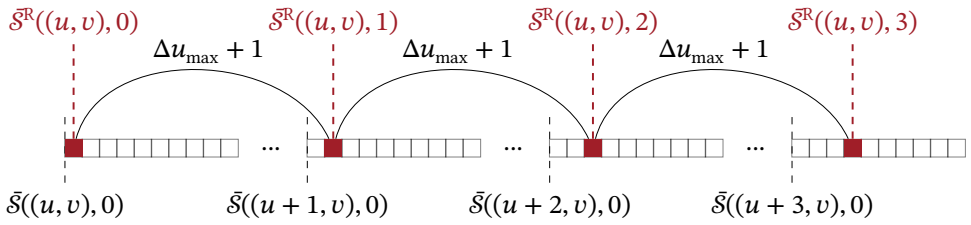


Figure 3.5: In the consistency check, an additional disparity map, which corresponds to the matching image, is approximated from the aggregated cost volume $\bar{\mathcal{S}}$ (cf. Equation (2.14)). The required entries are not situated directly next to each other, which hinders an efficient memory access. (Ruf et al. 2021b, Fig. 4.)

3.3.1.4 Median Filter

The execution of the median filter on the GPU is straight-forward. To each CUDA thread block a region in the final disparity map is assigned for which the filter is to be processed. In this, the necessary data is first copied to the shared memory. Pixels that lie outside the image get a disparity value of `0xFFFF` assigned, making them irrelevant for the filtering. With all the data in the shared memory, each thread of the thread block calculates the median for a pixel. Here, the first five iterations of the bubble sort algorithm are performed to sort the values in the 3×3 pixels neighborhood. After the fifth iteration, the five highest values are correctly sorted and the median can be extracted. (Ruf et al. 2021b, Sec. 2.2.4.)

3.3.2 Vectorized SIMD Processing with NEON Intrinsic Set on ARM CPUs

While conventional single-instruction-single-data (SISD) processing on the CPU requires one instruction for each data transfer between the memory and the registers of the CPU, as well as for each processing of the data stored inside these registers, the additional use of vector-processors allows to perform one instruction on a set of data simultaneously, the so-called *single-instruction-multiple-data (SIMD)* processing. Each vector-processor is divided into multiple vector-lanes, which in turn hold one datum each. The memory unit as well as the arithmetic and logical unit of the vector-processor allow to simultaneously transfer multiple data into and from the lanes of the vector-registers, as well as to simultaneously combine the lanes of two registers and store the results into the lanes of a third register. The vector-processors of the ARMv8 architecture contain 32 vector-registers with a size of 128 bits each (ARM 2017). The number of vector-lanes inside each register differs, depending on the data type which is to be stored inside the register. Thus, each register can have 16 lanes when data with a size of 8 bits each is to be stored, or only two lanes, if each lane holds a datum with the size of 64 bits. In particular, image processing is well-suited for the SIMD parallelization on vector-processors, since all pixels in an image are processed in the same manner, only with different data. (Ruf et al. 2021b, Appendix E.)

For an efficient use of vectorized SIMD processing, the developer is compelled to consider some design guidelines (ARM 2013):

- Reduction of the dependencies between conventional CPU and vectorized SIMD processing, in order to minimize the latency induced by copying data between the SISD and SIMD pipeline.
- Exploitation of cache coherence, to speed up the data transfer between the memory and the vector-registers.
- Dependencies in the data of vector-instructions trigger pipeline-stalls, in which the SIMD pipeline is stopped until the dependencies are resolved, slowing down the processing.
- Minimal use of conditional branching, since if the branch prediction unit (BPU) of the CPU predicts the wrong branch, the pipeline has to be recursively cleared until the point of branching and then restarted.

(Ruf et al. 2021b, Appendix E.)

In order to efficiently deploy the real-time processing pipeline for the estimation of dense disparity maps on an embedded CPU, such as the 8-core ARMv8.2 on the NVIDIA Jetson Xavier AGX, two strategies of parallelization are utilized in the scope of this work as illustrated in Figure 3.6, namely:

(i) a thread-level parallelization, and

(ii) a vectorized data processing with the SIMD NEON intrinsics.

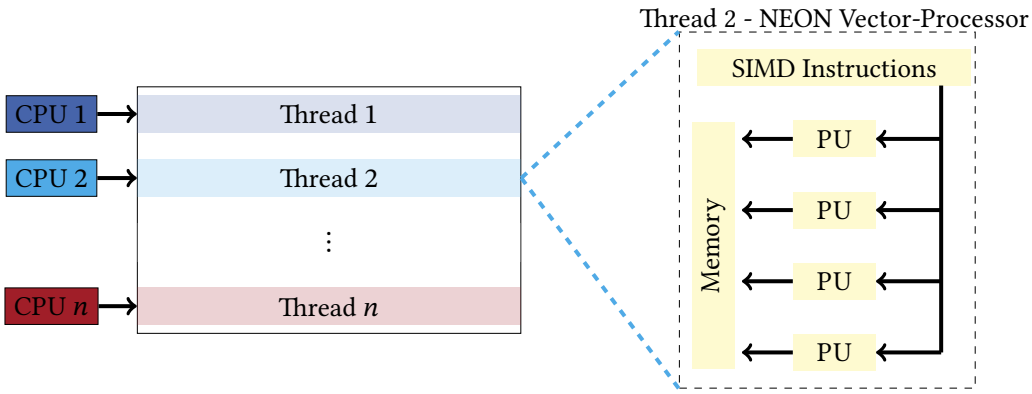


Figure 3.6: In the optimization of the SGM stereo algorithm for the execution on embedded CPUs, two different parallelization strategies are utilized. The implementation uses multiple threads to evenly distribute the processing on the available CPU cores. Each thread uses the AMD NEON instruction set to perform a vectorized SIMD processing by using the NEON processing units (PU). (Ruf et al. 2021b, Fig. 5.)

The implementation uses eight concurrent threads to efficiently utilize the available CPU cores. In each step of the processing pipeline (cf. Figure 2.2), in exception to the SGM optimization step, each thread operates on a different image stripe, thus operating isolated and independently from the other threads. At two locations in the processing pipeline of the stereo algorithm, i.e. before and after the SGM optimization, the concurrent threads need to be synchronized, since the SGM optimization relies on a different thread barrier than the other steps of the pipeline. In the other steps of the pipeline, the threads do not require any synchronization and can thus be processed fully concurrently. As part of the second parallelization strategy, each thread uses the ARM NEON instruction set (ARM 2013) to perform a vectorized SIMD processing on the vector-processors of the CPU. (Ruf et al. 2021b, Sec. 2.3.)

In the following sections, the optimization of each step of the processing pipeline for execution on an ARMv8 CPU, utilizing thread parallelism and SIMD processing, is discussed. Since the use of the NCC as a cost function for the image matching is computationally more expensive, and the frame rates achieved by the use of the CPU are anyway much lower than those achieved on the GPU, the NCC is not implemented for a vectorized SIMD processing on the CPU. (Ruf et al. 2021b, Sec. 2.3.)

3.3.2.1 Calculating the Census Transformation and its Hamming Distance

As illustrated by Figure 3.7, in each thread, the CT is calculated for 16 pixels simultaneously, using the SIMD vector-registers. In this, the 16 reference pixels with 8 bits each are loaded into one vector-register. In addition, the corresponding 24×16 neighbor pixels are loaded into one vector-register each, resulting in a total use of 25 vector-registers with 16 lanes. The full image is processed by sliding the illustrated window from left-to-right and top-to-bottom over the image. In doing so, there is a good chance that the image data, which is needed by the next iteration, is already cached. In the calculation of the CT, the comparison of the reference pixel with itself is omitted. This allows to represent the 24 bits of the resulting CT bit-string by three bytes and thus store the census descriptors of all 16 pixels in three vector-registers. This is also why only a support region of 5×5 pixels is considered in the optimized implementation of the CT for the ARM CPU. In order to also calculate the CT at the image border, where a part of the neighborhood lies outside the image, a conditional statement would need to be introduced, which is not recommended when using SIMD vectorization. Thus, the CT is only calculated up to two pixels with respect to the image border, reducing the image size by four pixels in each dimension for the subsequent processing. (Ruf et al. 2021b, Sec. 2.3.1)

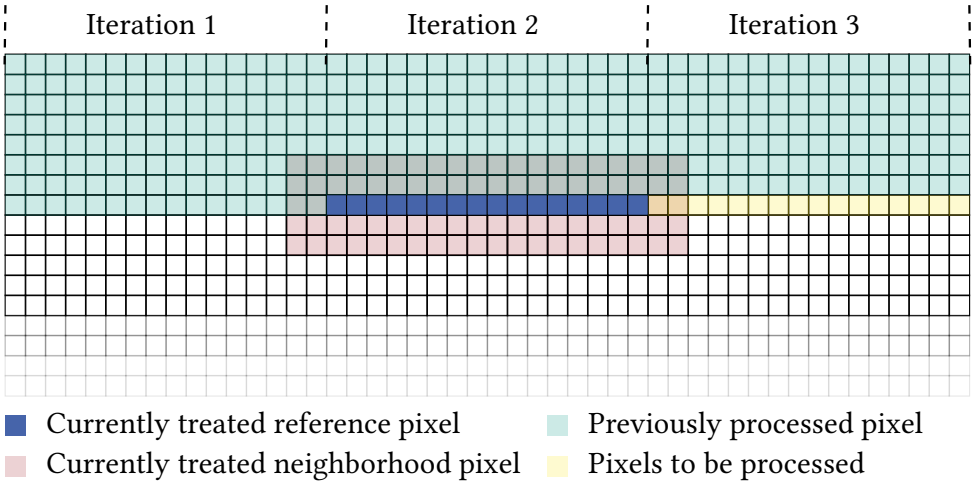


Figure 3.7: For the optimized CPU implementation, the census transform is processed for multiple pixels simultaneously by utilizing the NEON vector processing units. In this, a sliding window is used, which processes the image data from left-to-right and from top-to-bottom. (Ruf et al. 2021b, Fig. 6.)

To calculate the Hamming distance between two census descriptors, namely the one from the pixel of the reference image and the corresponding pixel in the matching image, the XOR operator is applied followed by the counting of how many bits are set to 1 in the final output. To count the number of bits which are set, the NEON instruction set offers a population count (VCNT) which can be applied to each vector-lane. As illustrated by Figure 3.8, the resulting matching cost, i.e. the Hamming distance of the CT, is then stored into the three-dimensional cost volume. By utilizing the SIMD instructions and the 32 vector-register, a maximum of 64

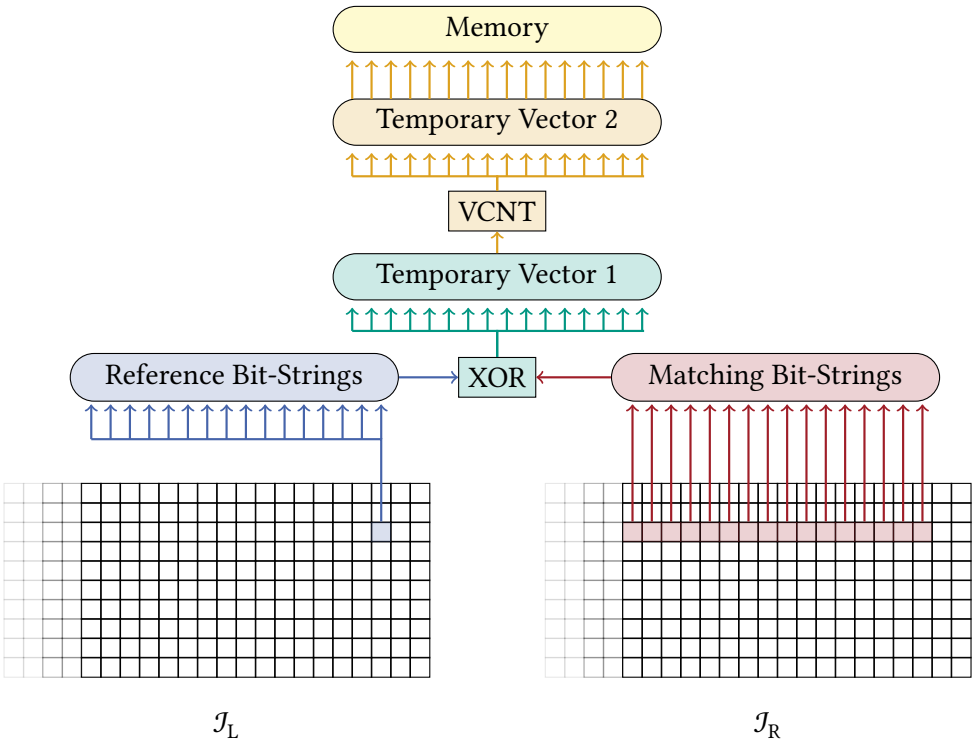


Figure 3.8: Illustration of the calculation of the Hamming distance with NEON intrinsics. Each census descriptor is loaded from $\mathcal{I}_L$ and $\mathcal{I}_R$ in separate vector-registers, on which a XOR operation is applied. The number of set bits inside a vector-register is counted by using the NEON hardware instruction VCNT (Vector Count Set Bits). (Ruf et al. 2021b, Fig. 7.)

matching costs can be calculated simultaneously in each thread. Thus, each thread processes 16 disparities and four lines simultaneously in one iteration. (Ruf et al. 2021b, Sec. 2.3.1)

In case the currently calculated disparity is bigger than the u-coordinate of the reference pixel, the corresponding matching pixel lies outside the image. In order to efficiently handle this case, the disparity for which the matching costs are currently being calculated as well as the u-coordinate of the currently processed pixel are additionally stored in two additional vector-registers. In each iteration, both of the above-mentioned registers are compared against each other and the result is stored in a third register. If the disparity is greater than the u-coordinate of the pixel, all bits inside the vector-lanes will be set. Finally, if an OR operation between the register with the matching cost and the register with the comparison result is applied, the matching cost of each disparity that spans over the image boundary will be set to `0xFF` and thus will not contribute in the subsequent search for the optimum. (Ruf et al. 2021b, Sec. 2.3.1)

3.3.2.2 Semi-Global Matching Optimization

The optimized implementation of the SGM algorithm can be divided into two separate steps. First, each thread calculates the SGM path costs for each path that is assigned to it according to Equation (2.11). As illustrated by Figure 3.9, the four vector-registers are first filled with the results of the previous iteration, so that the vector-register $L_\mathrm{r}(\mathrm{p}-\mathrm{r},\Delta u)$ will hold the previous path costs at the same disparity level, the vector-registers $L_\mathrm{r}(\mathrm{p}-\mathrm{r},\Delta u-1)$ and $L_\mathrm{r}(\mathrm{p}-\mathrm{r},\Delta u+1)$ will hold the previous path costs at the disparity level ± 1, and the vector-register $\min_{\Delta u} L_\mathrm{r}(\mathrm{p}-\mathrm{r},\Delta u)$ will hold the minimum path costs over all disparity levels at the considered pixel. According to Equation (2.11), the current matching costs from the cost volume as well as the penalties are added to the different vector-registers. Again, the NEON instruction set provides a method to get the minimum from the four vector-registers. The result is stored in the allocated memory and is compared to the path costs of the other disparities in order to get the minimum path cost for the next iteration. (Ruf et al. 2021b, Sec. 2.3.2)

In each thread, the costs for the disparities of two pixels are calculated simultaneously. In this, the implementation of the horizontal and vertical paths (L_LR, L_RL, L_TB, L_BT) differ from the implementation of the diagonal paths (L_TLBR, L_TRBL, L_BLTR, L_BRTL). On the straight paths, each thread processes two pixels on two neighboring rows or columns (Figure 3.10a). However, due to the different lengths of the diagonal paths, this cannot be applied to the processing of the same. Instead, on the diagonal paths, each thread processes two pixels lying on opposite sides of the image. This is illustrated by Figure 3.10b. (Ruf et al. 2021b, Sec. 2.3.2)

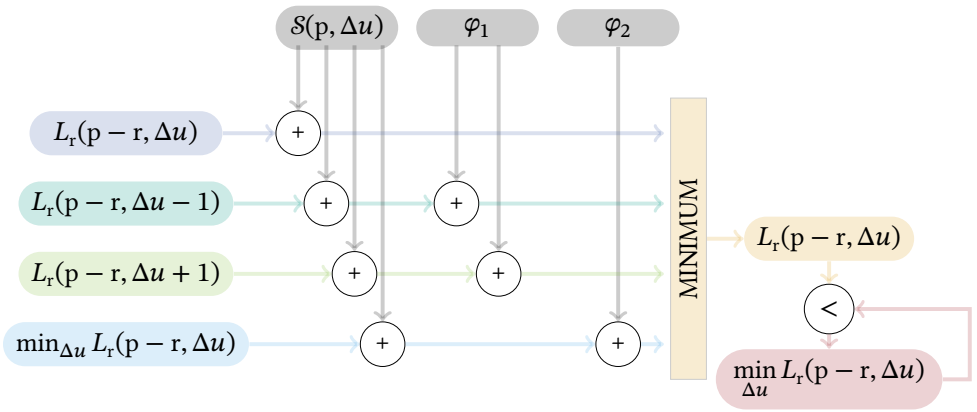
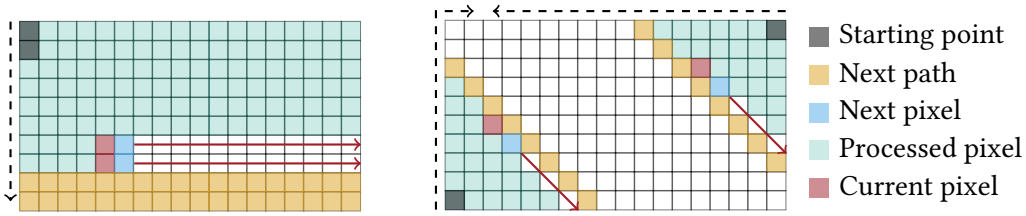


Figure 3.9: Schematic overview of the implementation of the SGM aggregation at a single pixel on an aggregation path. All components of the SGM path aggregation (cf. Equation (2.11)) are calculated simultaneously. A final minimum operation will yield the result which is stored in the aggregated cost volume. (Ruf et al. 2021b, Fig. 8.)



(a) Straight paths **(b)** Diagonal paths

Figure 3.10: Illustration of different traversal strategies in the aggregation of the SGM path costs within the optimized implementation using NEON intrinsics. (Ruf et al. 2021b, Fig. 9.)

In the second step of the SGM aggregation, each thread sums up all SGM path costs and finds the disparity with the minimum cost, i.e. the WTA cost, for each image stripe assigned to it. The different path costs are first copied into different vector-registers and then summed up. While summing up, each thread additionally stores the currently processed disparity as well as the minimum aggregated cost and the corresponding WTA disparity in additional vector-registers. Afterwards, the aggregated costs are compared to the minimum costs which are updated if necessary. If the minimum costs are updated, the corresponding WTA disparity is updated accordingly. The final aggregated path costs are stored into an aggregated cost volume $\bar{\mathcal{S}}$, which is needed for the subsequent consistency check, while the final WTA disparity is written into the disparity map. (Ruf et al. 2021b, Sec. 2.3.2)

3.3.2.3 Consistency Check

In the optimization of the consistency check for the CPU, the same difficulty as in the implementation of the approximated consistency check for the GPU is encountered, namely that the required data from the aggregated cost volume does not lie physically next to each other (cf. Figure 3.5). This makes it inefficient to load the data into vector-registers first and then process it with NEON intrinsics. There are two ways to solve this problem: The first possibility, which was proposed by Spangenberg et al. (2014), is to first transform the aggregated cost volume $\bar{\mathcal{S}}$ into a temporary volume in such a way that the data which is required to compute the approximated disparity map $\mathcal{U}^{\mathrm{R}}$ will physically lie next to each other in memory. Afterwards, the vector-registers can be filled with the data and the approximated disparity map $\mathcal{U}^{\mathrm{R}}$ corresponding to the matching image can be calculated efficiently with SIMD instructions. The second option, which is realized in the scope of this work, is to refrain from using SIMD instructions for the consistency check. Thus, in the implementation of the consistency check, only a thread-level parallelization, in which each thread is processing a different part of the cost volume, is used. This saves the need to rearrange the cost volume necessary to use SIMD instructions. (Ruf et al. 2021b, Sec. 2.3.3)

3.3.2.4 Median Filter Based on Sorting Networks

After the consistency check, a final 3×3 median filter is employed in order to remove any small outliers that still remain in the disparity map $\mathcal{U}$. This requires a sorting of all disparity values within the local neighborhood. In general, sorting algorithms like BubbleSort, MergeSort or QuickSort are able to sort an arbitrary number of input values and, in turn, require an indefinite number of comparative and swapping operations, making a naive implementation inappropriate for vectorized processing. In the case of the fixed-size median filter, however, the number of considered input values are fixed and are known in advance. This allows to use sorting networks (Knuth 1998), which sort an input vector of known size with a fixed number of comparators and swapping operations. A sorting network is comprised of two basic building blocks, namely:

(i) *wires*, which hold and transport one value of the input vector each, and

(ii) *comparators*, which are responsible for comparing the values of the connected wires and swap these if necessary.

The comparators always connect two wires with each other and assign the smaller value to the upper wire. Figure 3.11 illustrates the BubbleSort algorithm implemented using a sorting network. In this, the horizontal lines represent the wires, while the comparators are illustrated by the vertical connections. The values of the input vector move along the wires from left to right and get rearranged by the comparators, resulting in a sorted output vector on the right with the smallest value at the top. (Ruf et al. 2021b, Appendix F.)

In ReS^{2}tAC, the concept of sorting networks is utilized in order to allow for a parallel sorting with SIMD intrinsics and, in turn, for the vectorized processing of the 3×3 median filter. In this, the wires of the sorting network are realized using the vector-lanes of the vector-register, distributed among different vector-registers, utilizing one vector-lane from each register. Thus, for the implementation of one sorting network for the median filter with nine wires, nine different vector-lanes distributed over nine different vector-registers are utilized (Figure 3.12). (Ruf et al. 2021b, Sec. 2.3.4)

The comparators of the sorting network are implemented by using two comparison instructions of the NEON instruction set, that compare each vector-lane of two vector-registers and store the minimum or maximum in a third register. Thus, for each vector-lane, the minimum and maximum value are first extracted from the

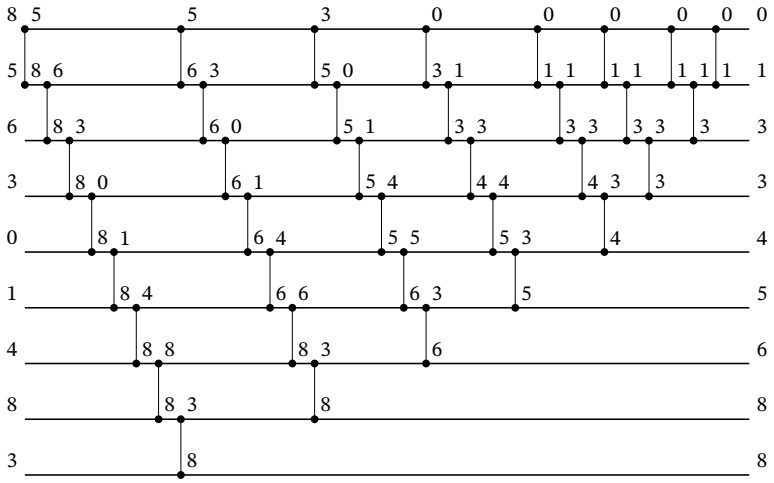


Figure 3.11: Exemplary sorting network, implementing the BubbleSort algorithm for nine input values. The horizontal lines represent the wires, while the comparators are illustrated by the vertical connections. The values of the input vector move along the wires from left to right and get rearranged by the comparators, assigning the smaller value to the upper wire. This results in a sorted output vector on the right with the smallest value at the top. (Ruf et al. 2021b, Fig. A4.)

two source-vector-registers. Then, the minimum is copied to the register which represents the upper lane of the sorting network, while the maximum is copied to the register which represents the lower lane. By this vectorized parallelization, the median filter is computed for 16 pixels simultaneously. Inherent to the nature of the BubbleSort algorithm, it is only necessary to calculate the first five iterations to get the median of a support region with a size of 3×3 pixels. (Ruf et al. 2021b, Sec. 2.3.4)

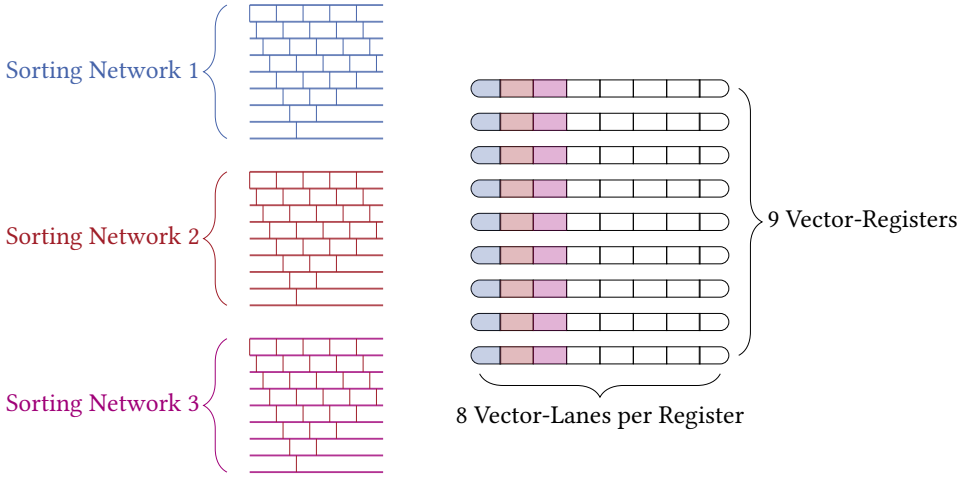


Figure 3.12: Illustration of the implementation of sorting networks using vectorized SIMD processing. The nine wires of a sorting network are mapped to vector-lanes of nine different vector-registers. (Ruf et al. 2021b, Fig. 10.)

3.4 Experiments

In the scope of this work, ReS^{2}tAC is quantitatively evaluated on two public stereo benchmarks with respect to accuracy, run-time and power consumption. In this, the results are also compared to those achieved by related approaches from the literature. For a qualitative demonstration of the usability of ReS^{2}tAC for real-time on-board processing a UAV, it is deployed on a DJI Manifold 2-G, which, in turn, is mounted on a DJI Matrice 210v2 RTK. While the quantitative evaluation is presented in Section 3.4.1, qualitative results of the execution of ReS^{2}tAC on board a UAV are presented in Section 3.4.2.

3.4.1 Quantitative Evaluation of Accuracy on Public Stereo Benchmarks

The quantitative assessment of the performance of ReS^{2}tAC and its different configurations for embedded stereo processing comprises an evaluation with respect to their accuracy in Section 3.4.1.2, as well as studies on the effects of the subpixel disparity refinement in Section 3.4.1.3 and the improvements gained by an accurate left-right consistency check in Section 3.4.1.4. Moreover, in Section 3.4.1.5, a study on the processing speed and power consumption of ReS^{2}tAC, together with the effects of reducing the aggregation paths of the SGM optimization, is conducted. In the scope of the quantitative evaluation, ReS^{2}tAC was deployed on the NVIDIA Jetson Xavier AGX with an 8-core 64-bit ARMv8.2 CPU and a 512-core Volta GPU. All measurements with respect to accuracy, timings and power consumption were done on this hardware. (Ruf et al. 2021b, Sec. 3.1.)

3.4.1.1 Evaluation Datasets and Error Measures

For the evaluation of the accuracy of ReS^{2}tAC and its different configurations, the training set of the KITTI 2015 stereo benchmark (Menze and Geiger 2015), which consists of 200 stereo image pairs and ground truth disparity

maps captured by a LiDAR sensor from on top of a car driving around urban areas, as well as the Middlebury 2014 stereo benchmark (Scharstein et al. 2014) are used. The latter one allows a more thorough evaluation on the accuracy and the effects of different optimizations in the processing pipeline, since it consists of 15 high-resolution stereo pairs of indoor scenes, together with highly accurate and dense ground truth disparity maps computed by means of structured lighting. (Ruf et al. 2021b, Sec. 3.1.)

The standard evaluation routine of the KITTI 2015 stereo benchmark (Menze and Geiger 2015) states the accuracy as the amount of erroneous pixels (D1-all), averaged over all m ground truth pixels in the evaluation set, for which the estimated disparity Δu_{est} differs by 3 or more pixels with respect to the ground truth Δu_{gt}:

$$\text{D1-all}(\Delta u_{\text{est}}, \Delta u_{\text{gt}}) = \frac{1}{m} \sum_{i=1}^{m} \left[|\Delta u_{\text{est}} - \Delta u_{\text{gt}}| \geq 3 \right], \tag{3.1}$$

with $[\cdot]$ representing the Iverson bracket. Since the ground truth was generated from a LiDAR sensor mounted at a slightly different position as the camera, for which the disparity map is estimated, the ground truth also provides disparity values in areas which are occluded in the second camera image and, in turn, usually only contains limited information in the estimated disparity map. Although the KITTI benchmark also provides ground truth maps which only contain *non-occluded (noc)* areas, the standard evaluation protocol uses the *occluded (occ)* dataset, which has also been used for evaluation scores in Table 3.2. Furthermore, the benchmark distinguishes between the results of the actual estimated (Est) disparity maps and interpolated versions of them (All). The latter ones allow a comparison between disparity maps of different density by applying a background interpolation to fill the pixels in the estimated disparity map for which no data is available. However, since ReS^{2}tAC uses a left-right consistency check and a median filter to explicitly remove outliers and inconsistent areas, the results achieved by the actual estimate are of greater interest. Nonetheless, for comparison, the results achieved by the interpolated disparity maps are also provided as well as the information on the density of the non-interpolated map, if available, which states the amount of pixels in the estimated disparity map which contain valid estimates. (Ruf et al. 2021b, Sec. 3.1.)

Similar to the evaluation routine of the KITTI 2015 stereo benchmark, the Middlebury benchmark ranks the algorithms based on four different accuracy levels, namely the amount of pixels whose error is greater than 0.5 (bad0.5), 1 (bad1), 2 (bad2) and 4 (bad4) pixels with respect to all m ground truth pixels in the evaluation set:

$$\text{bad}\theta(\Delta u_{\text{est}}, \Delta u_{\text{gt}}) = \frac{1}{m} \sum_{i=1}^{m} \left[|\Delta u_{\text{est}} - \Delta u_{\text{gt}}| > \theta \right], \tag{3.2}$$

with Δu_{est} and Δu_{gt} again denoting the estimated and ground truth disparity respectively, and $[\cdot]$ representing the Iverson bracket. The data is provided in full (F) image resolution with up to 3000×2000 pixels and a disparity range of up to 800 pixels as well as half (H) and quarter (Q) image resolution. The official evaluation is always performed on the full image resolution. Thus, if the results are generated on a dataset with a smaller resolution, the results are first being upsampled before being evaluated. (Ruf et al. 2021b, Sec. 3.1)

3.4.1.2 Accuracy

In this section, the accuracy achieved on the KITTI 2015 stereo benchmark is listed and discussed first. This is followed by the evaluation of the accuracy achieved on the Middlebury 2014 stereo benchmark.

KITTI 2015 Stereo Benchmark:

Table 3.2 lists the quantitative results of the accuracy of different configurations of ReS^{2}tAC as well as those of other approaches and implementations, which are achieved on the KITTI 2015 stereo benchmark. While the results of ReS^{2}tAC were achieved on the training set of the benchmark, the results of the approaches from literature were taken either from the corresponding publication or the official listing of the benchmark, which lists the results achieved on the actual test set. The upper part of the Table 3.2 lists algorithms and configurations, which are optimized for the deployment and execution on embedded hardware. Not all of these are variants of the Semi-Global Matching stereo algorithm. Yet, they serve as a good comparison since they were deployed on the same or similar hardware as the one presented in this work. The three algorithms at the bottom of the list serve as a baseline to ReS^{2}tAC. While the one from Hirschmüller (2008) reveals the accuracy achieved by the original SGM algorithm executed on a GPU with the census transform of unknown size as a cost function, the Semi-Global Block Matching (SGBM) variant of the OpenCV library is widely spread and easy to use, however, not optimized for embedded processing. The algorithm of Schönberger et al. (2018)

Table 3.2: Accuracy achieved by different algorithms and configurations of ReS^{2}tAC on the KITTI 2015 stereo benchmark (Menze and Geiger 2015). While the upper part lists algorithms that are optimized and deployed on embedded hardware, the three algorithms at the bottom are listed as a reference and a baseline. The results achieved by different configurations of ReS^{2}tAC are listed in the middle section. The accuracy is stated as the amount of erroneous pixels (D1-all), for which the estimated disparity differs by 3 or more pixels with respect to the ground truth. The KITTI 2015 benchmark distinguishes between the result of the actual estimated (Est) disparity map and an interpolated version of it (All), in which the pixels, for which no disparity is available, get interpolated by a simple background interpolation. As an evaluation ground truth, all available pixels were considered, not only the non-occluded ones. The density indicates, how many pixels inside the computed disparity maps have an estimate. †: The accuracy stated is computed with respect to the non-occluded pixels in the ground truth. (Ruf et al. 2021b, Tab. 2.)

Approach	Configuration	HW Device	Resolution (in pixels)	Error (in %)		Density (in %)
				D1-all (Est.)	D1-all (All)	
Zhao et al. (2020)	$CT_{5\times5}$ - SGM	FPGA	1242×375	-	11.8	-
Zhao et al. (2020)	$CT_{7\times7}$ - SGM	FPGA	1242×375	-	9.5	-
Ruf et al. (2018b)	$CT_{5\times5}$ - SGM	FPGA	640×360	4.6	31.2	46.4
Rahnama et al. (2018a)	$CT_{5\times5}$ - MGM	FPGA	1242×375	6.7	13.6	81.0
Rahnama et al. (2018a)	$CT_{13\times13}$ - MGM	FPGA	1242×375	4.8	9.9	85.0
Cui and Dahnoun (2019)†	$NCC_{9\times9}$-nativ	GPU	1242×375	-	16.6	-
Cui and Dahnoun (2019)†	$NCC_{9\times9}$-optimized	GPU	1242×375	-	13.1	-
Chang et al. (2020)	Z^2-ZNCC	GPU	1242×375	7.6	7.7	99.9
Hernandez-J. et al. (2016)	$CT_{9\times7}$ - SGM	GPU	1242×375	8.2	8.2	100
ReS^{2}tAC- CUDA	$CT_{5\times5}$ - SGM	GPU	640×480	5.4	8.4	94.5
ReS^{2}tAC- CUDA	$CT_{5\times5}$ - SGM	GPU	1242×375	4.3	8.3	88.8
ReS^{2}tAC- CUDA	$CT_{9\times7}$ - SGM	GPU	640×480	5.1	7.9	94.6
ReS^{2}tAC- CUDA	$CT_{9\times7}$ - SGM	GPU	1242×375	4.0	7.7	90.0
ReS^{2}tAC- CUDA	$NCC_{5\times5}$ - SGM	GPU	640×480	5.3	7.8	94.8
ReS^{2}tAC- CUDA	$NCC_{5\times5}$ - SGM	GPU	1242×375	4.3	8.1	90.0
ReS^{2}tAC- CUDA	$NCC_{9\times9}$ - SGM	GPU	640×480	5.9	8.2	94.7
ReS^{2}tAC- CUDA	$NCC_{9\times9}$ - SGM	GPU	1242×375	4.8	8.3	91.1
ReS^{2}tAC- NEON	$CT_{5\times5}$ - SGM	CPU	640×480	5.0	7.9	94.5
ReS^{2}tAC- NEON	$CT_{5\times5}$ - SGM	CPU	1242×375	4.6	8.5	90.0
Schönberger et al. (2018)	$NCC_{7\times7}$ - SGM-Forest	CPU	1242×375	4.3	4.4	99.9
OpenCV-SGBM	$SAD_{3\times3}$ - SGM	CPU	1242×375	5.9	10.9	90.4
Hirschmüller (2008)	CT - SGM	GPU	1242×375	6.4	6.4	100

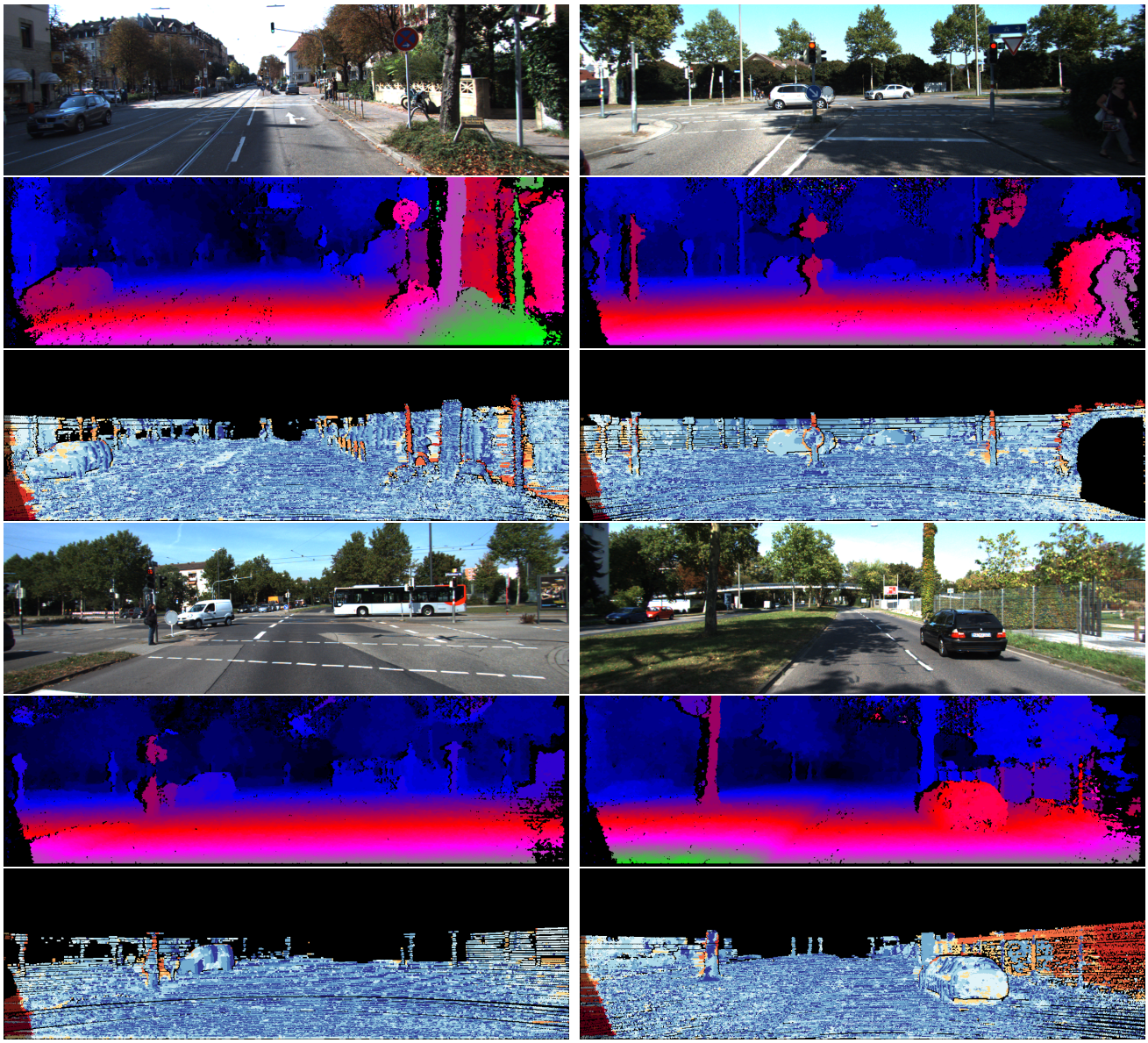


Figure 3.13: Four exemplary results from the KITTI 2015 stereo benchmark, computed with the $CT_{9\times7}$ - SGM configuration on the original image size. **Rows 1 & 4:** Reference images. **Rows 2 & 5:** Estimated disparity maps. **Rows 3 & 6:** Color-coded error images between the prediction and the ground truth. Error images use a log-color scale as described by Menze and Geiger (2015), marking correct estimates in blue color tones and wrong estimates in red color tones. (Ruf et al. 2021b, Fig. 11.)

is evaluated on both the KITTI 2015 stereo benchmark and the Middlebury 2014 stereo benchmark. They propose to use a random forest classifier to learn to efficiently fuse the different scan-line optimizations of the SGM algorithm, in order to reduce the number of optimization paths for embedded processing more efficiently. (Ruf et al. 2021b, Sec. 3.1.1.)

The accuracy of ReS^{2}tAC is evaluated using different cost functions and different support regions. For each configuration, the results achieved on the original image size provided by the KITTI benchmark, i.e. 1242×375 pixels, as well as on images with VGA resolution were evaluated. In the case of VGA resolution, the original images were first downsampled to a resolution of 640×480 pixels. Then, the stereo disparity estimation was performed and finally the resulting disparity maps were again upsampled to the original image size with a nearest-neighbor interpolation and a scaling of the disparities by the horizontal scale factor. Thus, for the

calculation of the accuracy measurements, the original image size was used. For the selection of the SGM penalties, different values were empirically evaluated. Those achieving the best results were then chosen for the subsequent experiments. These are $\varphi_1 = 27$ and $\varphi_2 = 86$ for the $CT_{9\times7}$ and $\varphi_1 = 90$ and $\varphi_2 = 880$ for the $NCC_{5\times5}$. An excerpt on the qualitative results for the best configuration of ReS^{2}tAC is presented in Figure 3.13. (Ruf et al. 2021b, Sec. 3.1.1.)

The results in Table 3.2 reveal, that the use of the census transform with a support region of 9×7 pixels and its Hamming distance as a matching cost function achieves the best results in the evaluation of both the actual estimate and the interpolated version. As can be expected, the use of downsampled versions of the input images yields less accurate results and yet, achieves a higher density in the resulting disparity maps. Furthermore, the use of the normalized cross correlation as a cost function achieves slightly less accurate results, while leading to a smaller throughput as discussed in Section 3.4.1.5. The implementation on the CPU using NEON SIMD intrinsics with a CT of size 5×5 pixels achieves similar and, in case of the smaller image resolution, slightly better results than its GPU counterpart. In summary, when considering the actual estimate, the configuration $CT_{9\times7}$ - SGM executed on the GPU, with the original image size of 1242×375 pixels, outperforms the baseline implementations as well as the other algorithms optimized for embedded hardware. When evaluating the interpolated disparity maps, the approaches achieve similar and mostly better results than the other embedded algorithms. The superiority of the baseline algorithms from Hirschmüller (2008) and Schönberger et al. (2018) with respect to the quality of the disparity estimation is to be expected, since they were optimized with respect to accuracy and not speed or throughput. (Ruf et al. 2021b, Sec. 3.1.1.)

Middlebury 2014 Stereo Benchmark:

The results achieved by ReS^{2}tAC on the training set of the Middlebury 2014 stereo benchmark are listed in Table 3.3, with a qualitative excerpt of the results achieved by the best configuration presented in Figure 3.14. None of the other approaches for stereo processing on embedded hardware, which were listed in the evaluation on the KITTI benchmark, have also been evaluated on the Middlebury 2014 stereo benchmark and, thus, these are not listed in this evaluation. However, results on the non-embedded baseline algorithms are available and are again listed in the lower part of Table 3.3. In this evaluation, the same configurations as those evaluated

Table 3.3: Accuracy achieved by different algorithms and configurations of ReS^{2}tAC on the Middlebury 2014 stereo benchmark (Scharstein et al. 2014). While the upper part lists the results of different configurations of ReS^{2}tAC, the three algorithms at the bottom are listed as a reference and a baseline. The accuracy is stated as the amount of erroneous pixels, whose error is greater than 0.5 (bad0.5), 1 (bad1), 2 (bad2) and 4 (bad4) pixels with respect to the ground truth. The density indicates, how many pixels inside the computed disparity maps have an estimate. The Middlebury 2014 stereo benchmark provides the image data in full (F), half (H) and quarter (Q) image resolution. The results of ReS^{2}tAC were computed on the quarter image resolution and evaluated according to the standard evaluation pipeline on the full resolution. (Ruf et al. 2021b, Tab. 3.)

Approach	Configuration	Resolution	Error (in %)				Density (in %)
			bad0.5	bad1	bad2	bad4	
ReS^{2}tAC- CUDA	$CT_{5\times5}$ - SGM	Orig. Q	77.0	59.5	35.4	13.5	93.0
ReS^{2}tAC- CUDA	$CT_{9\times7}$ - SGM	Orig. Q	77.3	59.9	35.7	13.6	93.3
ReS^{2}tAC- CUDA	$NCC_{5\times5}$ - SGM	Orig. Q	77.1	60.1	36.3	14.4	92.8
ReS^{2}tAC- CUDA	$NCC_{9\times9}$ - SGM	Orig. Q	77.1	61.2	38.7	17.1	91.9
ReS^{2}tAC- NEON	$CT_{5\times5}$ - SGM	Orig. Q	76.2	59.0	35.1	13.4	92.1
Schönberger et al. (2018)	$NCC_{7\times7}$ - SGM-Forest	Orig. H	43.1	14.8	7.0	3.7	100
OpenCV-SGBM	$SAD_{3\times3}$ - SGM	Orig. Q	67.3	42.1	25.5	17.3	100
Hirschmüller (2008)	CT - SGM	Orig. H	51.5	28.2	17.7	12.2	100

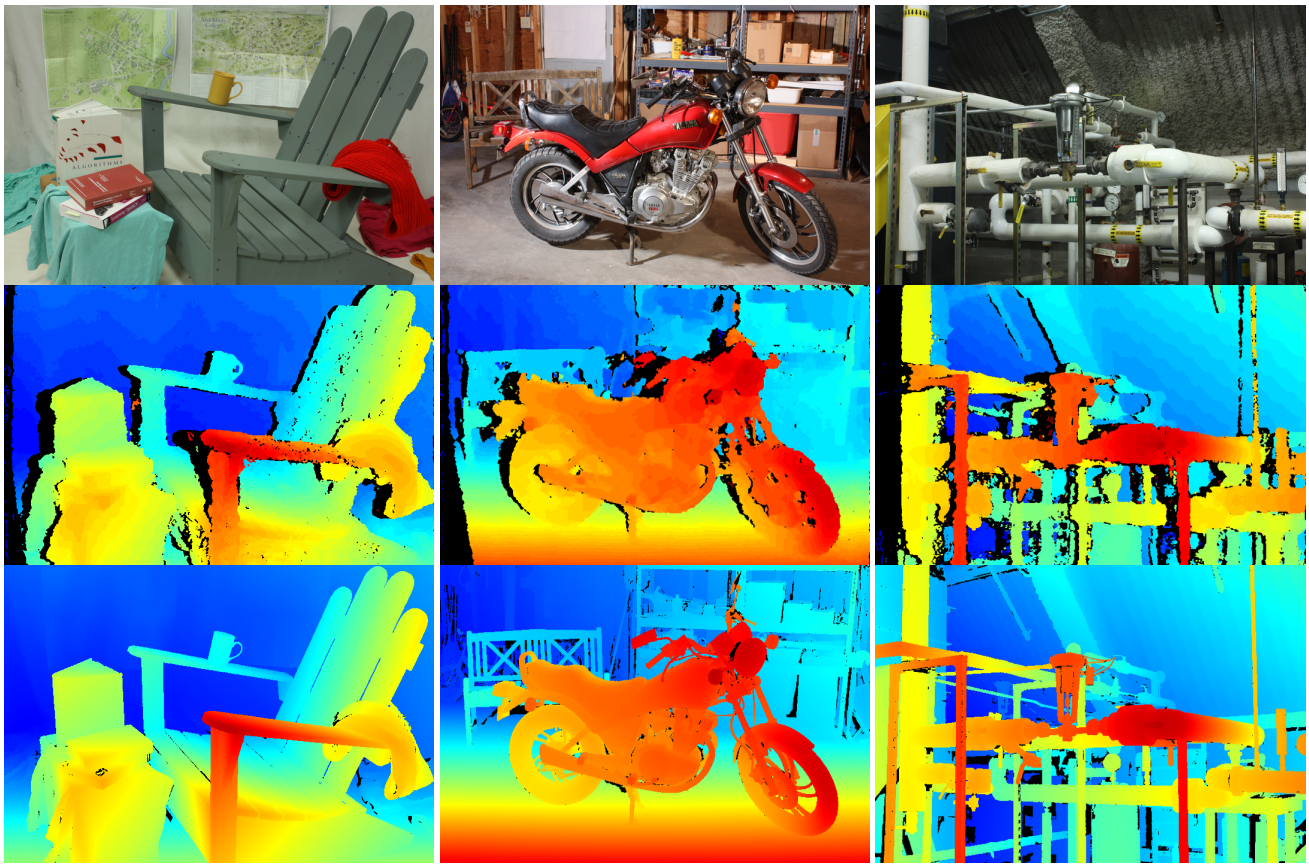


Figure 3.14: Three exemplary results from the Middlebury 2014 stereo benchmark, computed with the best-performing base configuration, i.e. $CT_{5\times5}$ - SGM on the quarter image resolution. **Row 1:** Reference images. **Row 2:** Estimated disparity maps. **Row 3:** Ground truth disparity maps. (Ruf et al. 2021b, Fig. 12.)

on the KITTI benchmark were studied. Since ReS^{2}tAC can only handle a disparity range of up to 256 pixels, the disparity maps were computed on the provided quarter image resolution (Orig. Q). Just as in the standard evaluation routine of the benchmark, the results listed were found after upsampling the disparity maps to the original image resolution with a nearest-neighbor interpolation and scaling the contained disparities with a factor of 4. Again, for the selection of the SGM penalties, different values were empirically evaluated and selected according to the results achieved, being $\varphi_1 = 11$ and $\varphi_2 = 39$ for the $CT_{5\times5}$ and $\varphi_1 = 140$ and $\varphi_2 = 730$ for the $NCC_{5\times5}$. (Ruf et al. 2021b, Sec. 3.1.1.)

Table 3.3 does not reveal any satisfying results. This, however, is not surprising, since only the quarter image resolution was used to compute the results, which were then upsampled by a factor of 4 for evaluation, introducing a lot of errors due to interpolation. As stated by Scharstein et al. (2014), the aim of this benchmark is to provide new challenges for modern stereo algorithms in terms of image resolution, accuracy and scene complexity, and not necessarily on the evaluation of optimizations with respect to computational efficiency and run-time. Nonetheless, the accuracy in the ground truth and the evaluation protocol of the Middlebury 2014 stereo benchmark allows for an evaluation of the improvement gained by a subpixel disparity refinement as done in the following section. (Ruf et al. 2021b, Sec. 3.1.1.)

3.4.1.3 The Effect of Subpixel Disparity Refinement

As described in Section 2.3.1, a subpixel disparity refinement can be computed for each pixel in the disparity map by fitting a parabola through the matching costs of the winning disparity and its two neighbors. This achieves an increase in accuracy of up to 0.8 % in case of the KITTI benchmark (Table 3.4), and up to 9 % in case of the Middlebury benchmark (Table 3.5), and yet only requires a small computational overhead (cf. Table 3.6). (Ruf et al. 2021b, Sec. 3.1.2.)

Table 3.4: Accuracy achieved by selected configurations with an additional subpixel disparity refinement on the KITTI 2015 stereo benchmark (Menze and Geiger 2015). The corresponding differences to the accuracies listed in Table 3.2 are given in parentheses. (Ruf et al. 2021b, Tab. 4.)

Approach	Configuration	Resolution (in pixels)	Error (in %)	
			D1-all (Est.)	D1-all (All)
ReS^{2}tAC- CUDA	$CT_{5\times5}$ - SGM - fine	1242×375	3.5 (−0.8)	8.0 (−0.4)
ReS^{2}tAC- CUDA	$CT_{9\times7}$ - SGM - fine	1242×375	3.3 (−0.7)	7.4 (−0.3)
ReS^{2}tAC- CUDA	$NCC_{5\times5}$ - SGM - fine	1242×375	3.7 (−0.6)	7.7 (−0.4)
ReS^{2}tAC- CUDA	$NCC_{9\times9}$ - SGM - fine	1242×375	4.3 (−0.5)	7.9 (−0.4)

Table 3.5: Accuracy achieved by selected configurations with an additional subpixel disparity refinement on the Middlebury 2014 stereo benchmark (Scharstein et al. 2014). The corresponding differences to the accuracies listed in Table 3.3 are given in parentheses. (Ruf et al. 2021b, Tab. 5.)

Approach	Configuration	Resolution	Error (in %)			
			bad0.5	bad1	bad2	bad4
ReS^{2}tAC- CUDA	$CT_{5\times5}$ - SGM - fine	Orig. Q	72.4 (−4.6)	52.1 (−7.4)	26.1 (−9.3)	10.4 (−3.1)
ReS^{2}tAC- CUDA	$CT_{9\times7}$ - SGM - fine	Orig. Q	73.0 (−4.3)	52.7 (−7.2)	26.6 (−9.1)	10.7 (−2.9)
ReS^{2}tAC- CUDA	$NCC_{5\times5}$ - SGM - fine	Orig. Q	73.8 (−3.3)	53.8 (−6.3)	27.8 (−8.5)	12.1 (−2.0)
ReS^{2}tAC- CUDA	$NCC_{9\times9}$ - SGM - fine	Orig. Q	73.8 (−3.3)	55.2 (−6.0)	30.6 (−8.1)	14.8 (−2.3)

3.4.1.4 Accurate Left-Right Consistency Check

To evaluate the effects of only approximating the disparity map corresponding to the right image of the stereo pair, which is needed for the left-right consistency check (cf. Section 2.3.3), a more exact and computationally more expensive consistency check was also implemented for the GPU. In this, the reference and matching images are switched and flipped and a second disparity map for the original matching image is calculated. This leads to a more accurate disparity map $\mathcal{U}^{\mathrm{R}}$ for the right input image, which, in turn, is used by Equation (2.13) of the consistency check. With a more accurate $\mathcal{U}^{\mathrm{R}}$, it is assumed that the consistency check is more effective in filtering outliers, but does the high computational overhead of fully calculating two disparity maps justify the increase in accuracy? The results of the evaluation do not indicate a significant improvement. The studies reveal an increase in accuracy of only 0.4-1.0 %, when calculating the disparity map of the right image from scratch compared to only approximating it from the cost volume corresponding to the left disparity map. However, the throughput is nearly halved when using a more accurate consistency check, as illustrated by configurations with the suffix “exact-cc” (exact consistency check) in Table 3.6. (Ruf et al. 2021b, Sec. 3.1.3.)

3.4.1.5 Throughput, Frame Rates and Power Consumption

A typical measure to quantify the processing speed of a stereo algorithm is the number of *frames per second (FPS)* which can be calculated. However, since the FPS greatly depends on the image size of the output and the disparity range, the efficiency of ReS^{2}tAC is assessed based on the throughput achieved, which is measured in *million disparity estimations per second (MDE/s)*:

$$\text{MDE/s} = \frac{w \cdot h \cdot |\Gamma_u|}{\text{run-time}}, \tag{3.3}$$

with w and h being the width and the height of the resulting disparity map, and $|\Gamma_u|$ being the size of the disparity range. Given the throughput achieved by a certain configuration or algorithm, it is possible to deduce the expected frame rates for a set of image size and disparity range:

$$\text{FPS} = \frac{\text{MDE/s}}{w \cdot h \cdot |\Gamma_u|}, \tag{3.4}$$

as done in Figure 3.15. Furthermore, the throughput allows for a better comparison between different algorithms with respect to their processing speed, due to its independence of a fixed image size and disparity range. (Ruf et al. 2021b, Sec. 3.1.4.)

In Table 3.6, the throughput achieved by different configurations of ReS^{2}tAC as well as the throughput of selected embedded algorithms from literature is listed. While all measurements in this work were done on the NVIDIA Jetson Xavier AGX with maximum performance, the related work optimized for the execution on GPU was deployed on the NVIDIA Jetson TX1 and TX2. In case of the related work, which has not explicitly stated the throughput of the respective algorithm, the values were calculated by using the stated frame rate and the corresponding image size according to Equation (3.4). For the run-time measurements, the whole processing

Table 3.6: Throughput achieved by ReS^{2}tAC and selected embedded algorithms from literature. The throughput is measured in million disparity estimations per second (MDE/s). In case of ReS^{2}tAC, all the measurements were done on the NVIDIA Jetson Xavier AGX board, with the power setting set to maximum performance. (Ruf et al. 2021b, Tab. 6.)

Approach	Configuration	HW Device	Throughput (in MDE/s)
Zhao et al. (2020)	$CT_{5\times5}$ - SGM	FPGA	9589.9
Zhao et al. (2020)	$CT_{7\times7}$ - SGM	FPGA	8743.7
Ruf et al. (2018b)	$CT_{5\times5}$ - SGM	FPGA	400.8
Rahnama et al. (2018a)	$CT_{5\times5}$ - MGM	FPGA	4246.9
Cui and Dahnoun (2019)	$NCC_{9\times9}$-optimized	GPU (TX2)	6497.1
Chang et al. (2020)	Z^2-ZNCC	GPU (TX2)	1669.2
Hernandez-Juarez et al. (2016)	$CT_{9\times7}$ - SGM	GPU (TX1)	747.1
ReS^{2}tAC-CUDA	$CT_{5\times5}$ - SGM	GPU (AGX)	652.7
ReS^{2}tAC-CUDA	$CT_{9\times7}$ - SGM	GPU (AGX)	644.9
ReS^{2}tAC-CUDA	$CT_{5\times5}$ - SGM - fine	GPU (AGX)	640.9
ReS^{2}tAC-CUDA	$CT_{9\times7}$ - SGM - fine	GPU (AGX)	633.1
ReS^{2}tAC-CUDA	$CT_{5\times5}$ - SGM - exact-cc	GPU (AGX)	365.7
ReS^{2}tAC-CUDA	$CT_{9\times7}$ - SGM - exact-cc	GPU (AGX)	361.8
ReS^{2}tAC-CUDA	$NCC_{5\times5}$ - SGM	GPU (AGX)	442.4
ReS^{2}tAC-CUDA	$NCC_{9\times9}$ - SGM	GPU (AGX)	344.1
ReS^{2}tAC-NEON	$CT_{5\times5}$ - SGM	CPU	166.2

pipeline was considered, including the upload of the stereo image pair and the download of the disparity image to and from the device memory of the GPU. (Ruf et al. 2021b, Sec. 3.1.4.)

Firstly, the measurements reveal the higher computational efficiency of the census transform as a matching cost function with respect to the normalized cross correlation. And secondly, they show the small computational overhead of the subpixel disparity refinement (fine), as discussed in Section 3.4.1.3. However, the measurements also unveil that the presented optimizations are less efficient than those from the literature. As expected, the implementations which are optimized and deployed on FPGA architectures are superior to those running on an embedded GPU. Furthermore, the superiority in terms of throughput of algorithms, such as those from Cui and Dahnoun (2019) and Chang et al. (2020), that do not rely on a complex regularization scheme like the SGM, is also to be expected. Nonetheless, ReS^{2}tAC has a lower throughput than a similar implementation of Hernandez-Juarez et al. (2016), while simultaneously being deployed on a more powerful system. (Ruf et al. 2021b, Sec. 3.1.4.)

A common way to further increase the throughput of the SGM algorithm is to reduce the number of aggregation paths. Most of the implementations aggregate the matching costs for each pixel along eight concentric paths. However, studies (Banz et al. 2010, Hernandez-Juarez et al. 2016) suggest that a reduction of the aggregation paths from eight to four does not have a significant negative impact on the accuracy of the resulting disparity map, while greatly increasing the processing speed. This is also supported by the experiments done in this work, in which the diagonal aggregation paths of the SGM optimization were omitted in the regularization of the cost volume, since they are the longest ones. The results of the conducted experiments are listed in Table 3.7, showing an increase in the throughput by a factor of up 1.45, while reducing the accuracy and the density by a maximum of 0.2 % and 1.6 %, respectively. In this, the approach of Hernandez-Juarez et al. (2016), which is comparable to the approach of ReS^{2}tAC, is outperformed in terms of both accuracy and throughput. (Ruf et al. 2021b, Sec. 3.1.4)

With the throughput listed in Tables 3.6 and 3.7, the expected FPS, which are to be achieved for sets of different image sizes and disparity ranges according to Equation (3.4), are calculated and plotted as curves in Figure 3.15. Here, not all configurations of ReS^{2}tAC are plotted. Instead, one configuration for each cost function and hardware as well as the corresponding versions with only four paths in the SGM optimization were selected. Additionally, one configuration that performs a subpixel disparity refinement was selected for comparison. Furthermore, the FPS curves for the related approaches from literature deployed on FPGA (†) and GPU (∗) architectures are plotted, as well as one curve achieved by ReS^{2}tAC on a high-end desktop NVIDIA RTX 2070 Super GPU. The image sizes and disparity ranges corresponding to the KITTI 2015 and Middlebury 2014 Q benchmark are printed in bold. The curves provide a good visual representation of the throughput listed in Tables 3.6 and 3.7 and show that with decreasing complexity, i.e. a reduced image size and disparity range,

Table 3.7: Throughput and accuracy achieved by ReS^{2}tAC with a reduced number of aggregation paths in the SGM optimization. Instead of eight aggregation paths, only the two horizontal and the two vertical paths were used. The accuracy in terms of error rate and density was measured on the KITTI 2015 stereo benchmark. (Ruf et al. 2021b, Tab. 7.)

Approach	Configuration	HW Device	Throughput (in MDE/s)	Error (in %) D1-all (Est.)	Density (in %)
ReS^{2}tAC-CUDA	$CT_{5\times5}$ - 4-Path-SGM	GPU (AGX)	924.1 (×1.42)	4.3 (+0.0)	88.1 (−0.7)
ReS^{2}tAC-CUDA	$CT_{9\times7}$ - 4-Path-SGM	GPU (AGX)	904.4 (×1.40)	4.2 (+0.2)	89.2 (−0.8)
ReS^{2}tAC-CUDA	$NCC_{5\times5}$ - 4-Path-SGM	GPU (AGX)	615.4 (×1.39)	4.3 (+0.0)	88.6 (−1.4)
ReS^{2}tAC-CUDA	$NCC_{9\times9}$ - 4-Path-SGM	GPU (AGX)	436.5 (×1.27)	4.6 (+0.2)	89.5 (−1.6)
ReS^{2}tAC-NEON	$CT_{5\times5}$ - 4-Path-SGM	CPU	241.8 (×1.45)	4.8 (+0.2)	89.3 (−0.7)

the frame rates increase rapidly. Thus, the curves reveal how different configurations, approaches and utilized hardware compare in terms of processing speed. (Ruf et al. 2021b, Sec. 3.1.4)

A key characteristic of embedded systems is their power consumption and, depending on which platform the system is deployed, this can be a very crucial characteristic. The simple metric of FPS per watt (FPS/W) helps to quantify the efficiency of image processing algorithms with respect to the power consumption of the system on which they are deployed. The NVIDIA Jetson Xavier AGX, on which ReS^{2}tAC is deployed, allows to set four different power settings, namely:

MAXN This is the setting enabling the maximum performance. With this, all eight cores of the ARM CPU are activated and can clock up to a maximum of 2.3 GHz. The maximum clock rate of the GPU is set to 1.4 GHz. This is the setting with which all previous experiments were conducted.

30 W In this setting, again all eight cores of the CPU are enabled. However, they are restricted to a maximum clock rate of 1.2 GHz. Furthermore, the clock rate of the GPU is restricted to 905 MHz.

15 W In this setting, four cores of the CPU are enabled which clock at a maximum rate of 1.2 GHz, while the GPU clocks up to 675 MHz.

10 W In the smallest setting, only two cores of the CPU are enabled with a maximum of 1.2 GHz and the clock rate of the GPU is restricted to only 522 MHz.

In Figure 3.16, the FPS/W which are expected to be achieved by different configurations of ReS^{2}tAC as well as by some approaches from the literature are plotted, again, in dependence of different image sizes and disparity ranges. These calculations are based on the throughput and power consumption measured or stated. In case of ReS^{2}tAC, the two configurations reaching the highest throughput on the GPU and the CPU were selected. In these experiments, the throughput and power consumption, the latter being provided by internal sensors of the AGX, were measured given different power settings. Note that the actual power consumption on the AGX does not coincide with the statement of the power setting, as the latter one only indicates an upper bound on the consumption. From literature, approaches which provide a value on the power consumption in addition to the throughput or frame rate were chosen for comparison. Unfortunately, in terms of related work that involves deploying a stereo algorithm on embedded GPUs, this was only done by Hernandez-Juarez et al. (2016). (Ruf et al. 2021b, Sec. 3.1.4)

The curves in Figures 3.15 and 3.16 clearly reveal the superiority of FPGA-based approaches over those deployed on GPUs. Not only do they achieve much higher frame rates, but also require significantly less power and, in turn, also achieve higher FPS/W. However, the emerging embedded GPUs also achieve quite reasonable frame rates with respect to their power consumption, and depending on where the systems are deployed, e.g. quadrotor-based systems, the power required by the GPU is negligible, when compared to that required by the rotors. But more on this is provided in the discussion on what the results mean for a possible use-case (cf. Section 3.5). Interestingly, the curves in Figure 3.16 show that both the CUDA and NEON implementation of ReS^{2}tAC achieve the best efficiency on the 30 W power setting. Even the CUDA implementation being run on the 15 W power setting is still more efficient than the same run on the setting with maximum performance. It is assumed that this is the result of clocking down the CPU, since it is less efficient than the GPU. Nonetheless, the 30 W and 15 W power settings reduce the throughput by approximately 29 % and 42 %, respectively, compared to that achieved on MAXN. (Ruf et al. 2021b, Sec. 3.1.4.)

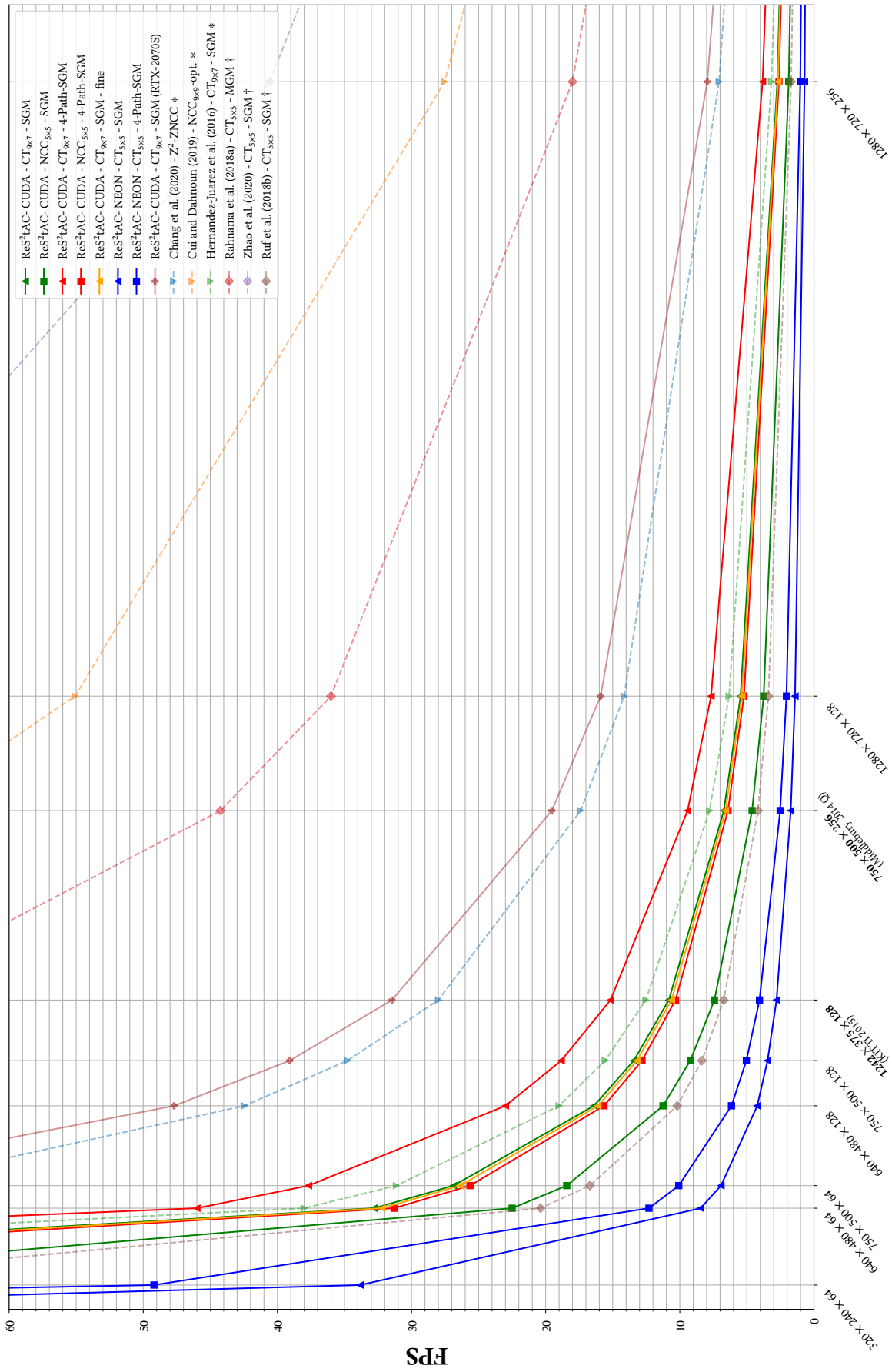


Figure 3.15: Expected FPS for sets of different image sizes and disparity ranges, based on the throughput achieved by different configurations and approaches listed in Table 3.6 and Table 3.7. Image resolution and disparity ranges corresponding to the KITTI 2015 and Middlebury 2014 Q benchmark are printed in bold. *: Approaches from literature deployed on embedded GPU hardware. †: Approaches from literature deployed on embedded FPGA hardware. (Ruf et al. 2021b, Fig. 13.)

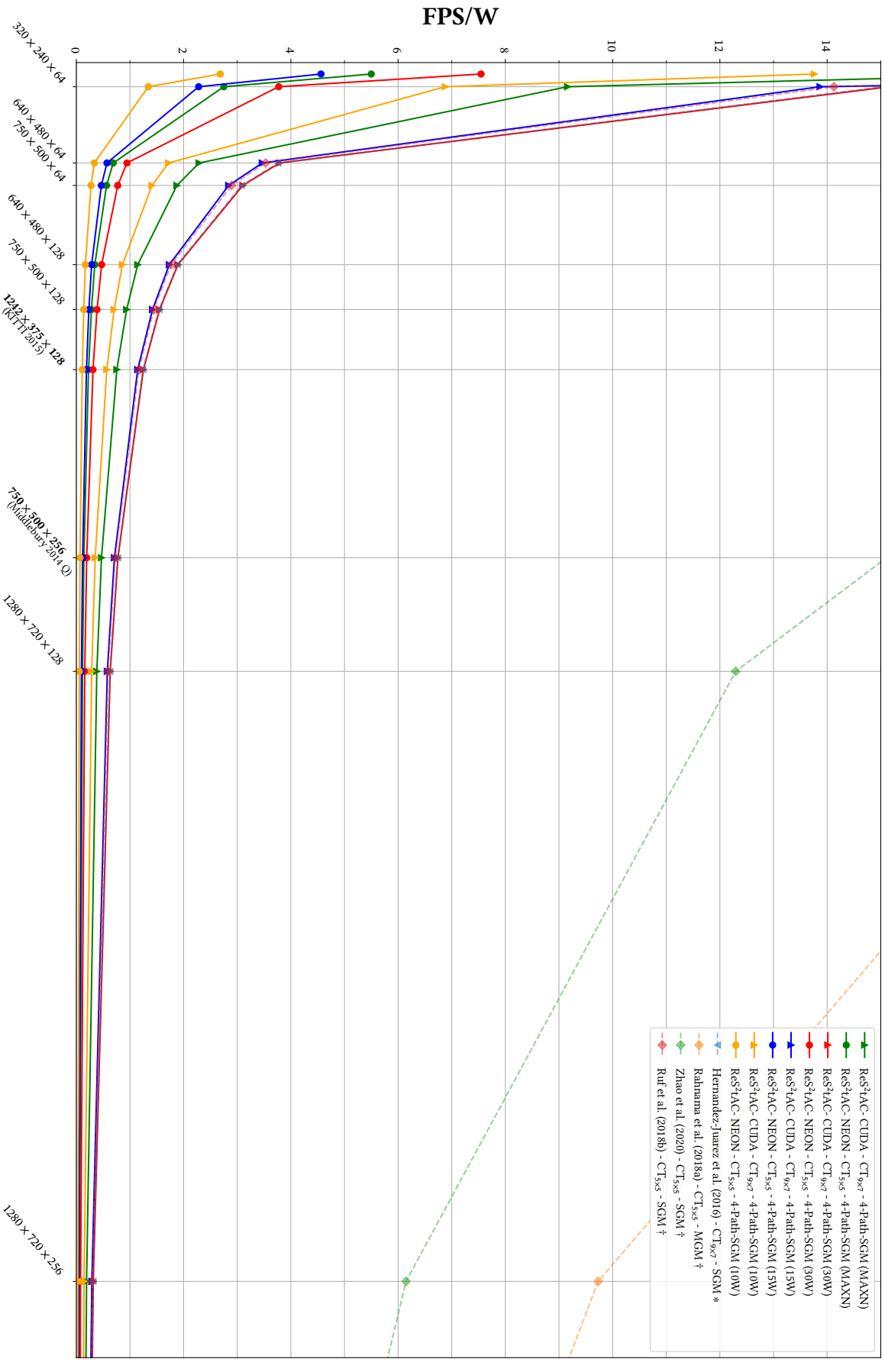


Figure 3.16: Expected FPS/W for sets of different image sizes and disparity ranges, based on the throughput and the power consumption achieved by different configurations and approaches. In case of ReS²tAC, the power settings of the AGX were varied in order to measure its efficiency. Image resolution and disparity ranges corresponding to the KITTI 2015 and Middlebury 2014 Q benchmark are printed in bold. *: Approaches from literature deployed on embedded GPU hardware. †: Approaches from literature deployed on embedded FPGA hardware. (Ruf et al. 2021b, Fig. 14.)

3.4.2 Qualitative and Quantitative Evaluation of Real-Time Stereo Processing On-Board Commodity UAVs

As part of use-case-specific experiments, in which the aim is to bring real-time stereo processing with an embedded CUDA device on board a UAV, a DJI Matrice 210v2 RTK was equipped with a DJI Manifold 2-G processing unit (cf. Figure 1.1), which is based on the NVIDIA Jetson TX2 architecture containing a 4-core 64-bit ARMv8 CPU and a 256-core Pascal GPU. As a stereo camera, the integrated stereo vision sensor was used, which can be accessed by the Manifold through the DJI onboard SDK. (Ruf et al. 2021b, Sec. 3.2.)

In the maximum power setting (MAXN), the ARM Cortex A57 CPU of the TX2 inside the Manifold has four cores with a maximum clock rate of 2 GHz, while the built-in Tegra GPU clocks up to a maximum of 1.3 GHz. The integrated vision sensor provides non-rectified, grayscale stereo image pairs with an image resolution of up to 640×480 pixels at a frame rate of 20 FPS. To calibrate the stereo sensor and, in turn, precompute the rectification maps needed to transform the input images into a rectified stereo pair prior to the actual stereo processing, the standard calibration routine of OpenCV was used. The integrated stereo vision sensor also provides precomputed disparity maps at a frame rate of 10 FPS and with an image resolution of 320×240 pixels (DJI 2021). (Ruf et al. 2021b, Sec. 3.2.)

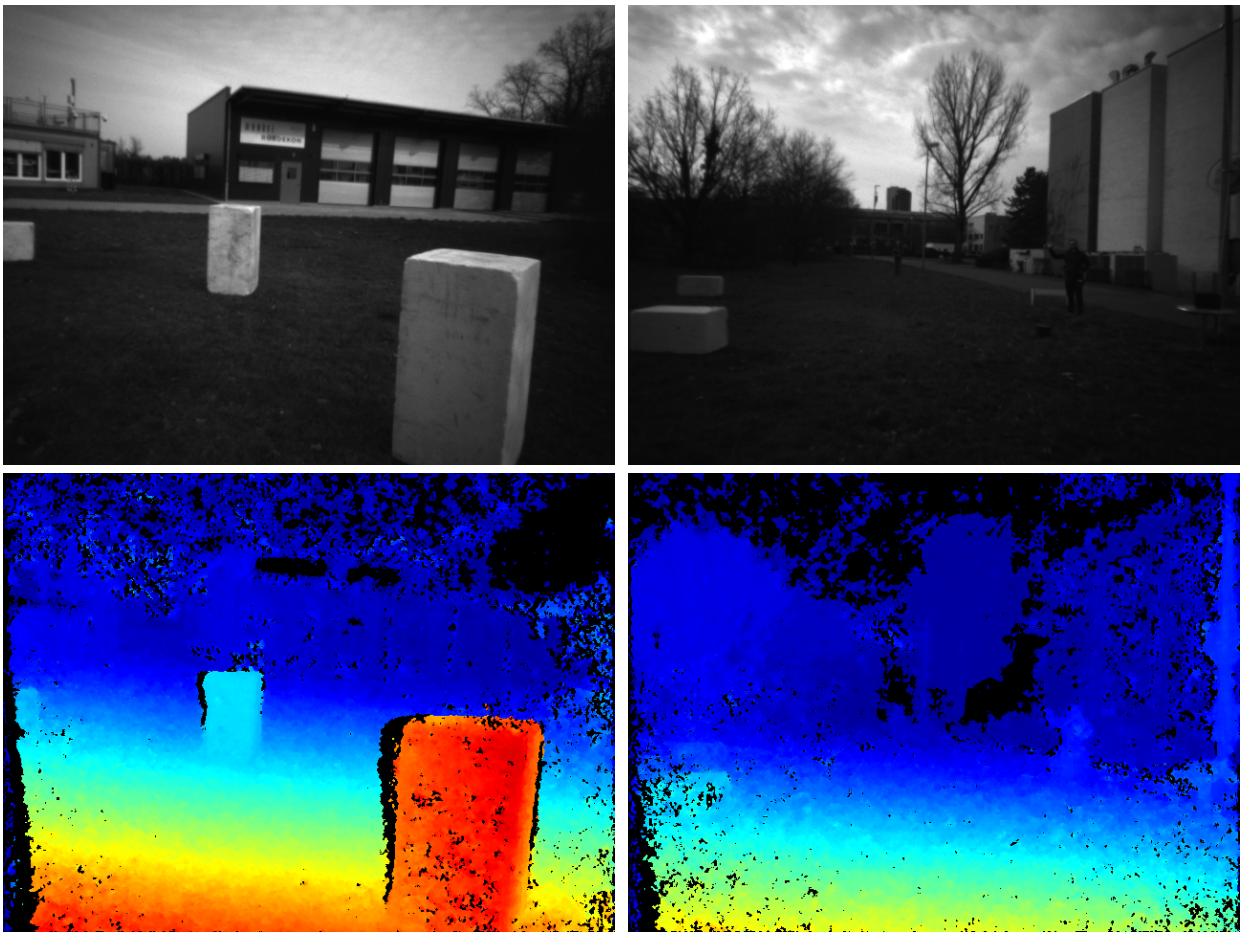


Figure 3.17: Qualitative results of ReS^{2}tAC run on the DJI Manifold 2-G mounted on the DJI Matrice 210v2 RTK, using the data of the stereo vision sensor as input. In this, the UAV was flying 1-2 m above the ground in order to demonstrate the ability of ReS^{2}tAC to appropriately estimate the scene depth. Top row: Rectified reference images. Bottom row: Corresponding disparity maps, color-coded with the jet color map, going from red (0.9 m), over yellow and green to blue (57 m). (Ruf et al. 2021b, Fig. 16.)

Table 3.8: Throughput and frame rates achieved by two configurations of ReS^{2}tAC on the DJI Manifold equipped with a Jetson TX2. (Ruf et al. 2021b, Tab. 8.)

Approach	Configuration	HW Device	Throughput (in MDE/s)	Frame Rate at $640 \times 480 \times 64$ pixels (in FPS)	Frame Rate at $320 \times 240 \times 64$ pixels (in FPS)
ReS^{2}tAC-CUDA	$CT_{9\times7}$ - 4-Path-SGM	GPU (TX2)	304.7	15.5	62.0
ReS^{2}tAC-NEON	$CT_{5\times5}$ - 4-Path-SGM	CPU	102.2	5.2	20.8

In the scope of this work, two configurations of ReS^{2}tAC were deployed and tested, one running on the GPU and the other on the CPU, namely: ReS^{2}tAC-CUDA with $CT_{9\times7}$ - 4-Path-SGM and ReS^{2}tAC-NEON with $CT_{5\times5}$ - 4-Path-SGM. Both configurations rely on the Hamming distance of the census transform as a cost function and use only four paths in the SGM optimization in order to reach a higher throughput. The throughput and frame rates as well as qualitative results achieved by the two configurations are listed in Table 3.8 and shown in Figure 3.17, respectively. (Ruf et al. 2021b, Sec. 3.2.)

3.5 Discussion

When assessing the usability of embedded stereo algorithms for the deployment and usage in actual applications, the accuracy, processing speed as well as power consumption are crucial characteristics. Thus, in the following, the experimental results are discussed with respect to each of these aspects, namely accuracy (Section 3.5.1), processing speed (Section 3.5.2) and power consumption (Section 3.5.3).

3.5.1 Accuracy

The quantitative evaluation on the KITTI 2015 stereo benchmark (Table 3.2) reveals a high and state-of-the-art accuracy of ReS^{2}tAC, both in the actual estimated disparity map (Est), in which inconsistent regions are removed, as well as in the interpolated versions (All). The latter are being used as part of the standard evaluation routine of the benchmark. Understandably, the accuracy of the interpolated disparity maps is used as a primary ranking in the benchmark, since it allows to compare disparity maps with different densities, however, in the assessment of the performance of ReS^{2}tAC, the accuracies of the filtered disparity maps are of greater importance. In particular, when looking at the qualitative results in Figure 3.13 and the stated densities of the estimated disparity maps, it becomes clear that the few areas, that are being removed by the post-filtering, mostly arise in occluded areas and are thus legitimately removed, since it is not possible for the algorithm to reason about the depth in areas which are only seen by one camera. (Ruf et al. 2021b, Sec. 4.1.)

Unfortunately, the results of the Middlebury 2014 stereo benchmark render the accuracy of ReS^{2}tAC far from the state-of-the-art. With an error rate of over 35 % for three out of four accuracy levels, the results are not really satisfying. Apart from the high accuracy levels in the evaluation and the high-resolution ground truth, it is assumed that these poor results can also be attributed to the fact that the resolution, with which the disparity maps are being evaluated, is 4× bigger than the input resolution, introducing a lot of error in the process of upscaling and interpolation. Thus, the results on ground truth data with only a quarter of the original image resolution were also evaluated. They reveal that the error rate is on average reduced by way over 50 %. For the configuration with a 5×5 census transform, being the best-performing configuration on this dataset, the accuracies calculated are 35.8 % (bad0.5), 14.2 % (bad1), 7.4 % (bad2) and 4.9 % (bad4), which is in the comparable range to that obtained on the KITTI benchmark. (Ruf et al. 2021b, Sec. 4.1.)

As the name suggests, ReS^{2}tAC is intended for real-time, rather than high accuracy, stereo processing. The main use-case of this work is the deployment on COTS UAVs equipped with embedded ARM or CUDA hardware, with the purpose of obstacle detection and collision avoidance. In the light of this use-case, the KITTI benchmark is more appropriate than the Middlebury benchmark, since it comprises image data that depict real-world scenes in a quality that can also be expected from cameras mounted on UAVs. Moreover, for collision avoidance, it is not necessary to have disparity maps with high subpixel accuracy as evaluated by the Middlebury benchmark. It is more important to reliably detect the location of objects in the perceived scene and extensively reconstruct their appearance. Not only do the quantitative results prove the high accuracy of the disparity maps, the qualitative presentation of some of the disparity maps also shows that ReS^{2}tAC is able to reveal objects, which are only visible by a second glance, such as the person on the right of Figure 3.13 (Row 2, Column 2) or Figure 3.17 (Column 2). (Ruf et al. 2021b, Sec. 4.1.)

With respect to the KITTI 2015 benchmark, most of the configurations of ReS^{2}tAC outperform the other approaches that perform real-time stereo estimation on embedded hardware. Even compared to the baseline algorithms, the results are compatible, especially when considering the frame rates achieved. Furthermore, the conducted experiments on the effects of subpixel disparity refinement (cf. Tables 3.4 and 3.5) show, that its use can increase the accuracy by up to 35 % without significantly decreasing the throughput (cf. Table 3.6). (Ruf et al. 2021b, Sec. 4.1.)

3.5.2 Processing Speed

Compared to the related work, the throughput and processing speed of ReS^{2}tAC are not very impressive. It is to be expected that a significantly lower throughput is reached by ReS^{2}tAC, compared to that achieved by approaches running on FPGAs (Zhao et al. 2020, Rahnama et al. 2018a), as well as a slightly lower throughput compared to approaches that do not rely on a computationally expensive optimization scheme but are run on an embedded GPU (Cui and Dahnoun 2019, Chang et al. 2020). However, the $CT_{9\times7}$ - SGM configuration of ReS^{2}tAC has a lower throughput than that of a comparable configuration by Hernandez-Juarez et al. (2016), while at the same time running on a hardware generation that is two times newer and that has twice the number of CUDA cores. This is not very satisfactory and something that will need to be further investigated in the scope of future work. One difference between the work of Hernandez-Juarez et al. (2016) and this one is that in the time measurements conducted in the scope of this work the data transfer to and from the memory of the GPU is included. Hernandez-Juarez et al. (2016) argue that the processing can be overlapped with the computation and thus is not relevant for the computation of the throughput. However, in the case of this work, the data transfer only takes up 4-6 % of the processing time, which is not enough to reach the throughput of Hernandez-Juarez et al. (2016), if omitted. Other optimization steps are the utilization of SIMD instructions to vectorize the cost aggregation with the CUDA kernels, or to streamline the aggregation of the last path and the disparity computation to reduce memory access, which should lead to a 1.35× performance speed-up (Hernandez-Juarez et al. 2016). A third and significant difference between ReS^{2}tAC and the approach of Hernandez-Juarez et al. (2016) with respect to performance is that Hernandez-Juarez et al. (2016) refrain from any post-processing like left-right consistency check or median filter. This allows to reach a higher throughput but, in turn, reduces the accuracy of the results, leading to an error-rate which is twice as high as that of ReS^{2}tAC in terms of the actual estimates (cf. Table 3.2). (Ruf et al. 2021b, Sec. 4.2.)

Furthermore, in the light of the use-case, addressing on-board stereo processing on COTS UAVs, the throughput reached to this end by ReS^{2}tAC is sufficient, as another limiting factor is the camera sensor and the data throughput between the sensor and processing board that is provided by commodity systems. Since a FPGA is

typically located closer to the sensor with a direct and high-bandwidth connection, which allows to stream the image data directly into the memory of the FPGA, a high data throughput is of greater importance. However, embedded systems equipped with a GPU, like the NVIDIA Jetson series, that are mounted on COTS UAVs, are usually connected to the sensor via USB or similar, with the CPU capturing the data and storing it in global memory, from where it gets transferred to the device memory of the GPU before it can be processed. This process does not allow for a high input frame rate (usually between 20 FPS and 30 FPS), as can be seen in the example of the DJI Matrice (cf. Section 3.4.2), and therefore does not require an extremely high throughput. Nonetheless, the more time spent on the estimation of the disparity map, the less time is available for the successive interpretation, e.g. obstacle detection and avoidance. This raises the interest to further looking into the optimization of the processing speed in the future. (Ruf et al. 2021b, Sec. 4.2.)

3.5.3 Power Consumption

The final aspect, which is discussed, is the power consumption of ReS^{2}tAC running on the NVIDIA Jetson Xavier AGX in the light of deployment on a rotor-based UAV and whether it is feasible to use embedded CPUs for stereo processing. The average power consumption of the complete NVIDIA Jetson Xavier AGX (i.e. including the GPU, the CPU and the EMC) under the maximum power setting MAXN during the execution of the CUDA- and NEON-based implementations is approximately 20.1 W and 17.9 W, respectively. The DJI Matrice 210v2 RTK is powered by two batteries with a total energy of 349.2 Wh, which allows a maximum flight time of 33 minutes when no payload is attached (DJI 2020b). Thus, during flight, the bare DJI Matrice 210v2 RTK consumes around 634.9 W per minute. This power consumption obviously increases with each gram of payload that is being attached. Given these measurements, it can be calculated that the power consumption of the AGX under the highest power setting only makes up 3.2 % and 2.8 % with respect to the total power consumption of the DJI Matrice, thus reducing the flight time by a maximum of 1 minute, when the CUDA- and NEON-based implementations are executed, respectively. This is an upper bound on the relative power consumption of the AGX, as the power consumption during flight of the DJI increases when payload is attached. The use of other power settings, that will increase the FPS/W ratio (cf. Figure 3.16), but also decrease the absolute frame rate, is dependent on the use-case and on whether the gain of few seconds in flight time is more valuable than a major reduction in frame rate. (Ruf et al. 2021b, Sec. 4.3.)

The power consumption during the execution of the presented CUDA-based implementation is higher than during the execution of the NEON-based implementation. This is expected as the GPU, which consumes more power than the CPU, is clocked down when not in use. However, the reduced power consumption does not stand in relation to the loss in processing speed of the NEON-based implementation compared to the ones based on CUDA. This is clearly revealed by the curves in Figure 3.16, depicting that the NEON-based implementations running on the CPU have the worst FPS/W ratio. Thus, it can be concluded that the use of embedded GPUs is preferred over embedded CPUs. However, some drones, e.g. VOXL[1], are only equipped with an embedded ARM CPU for which a vectorized stereo processing with NEON intrinsics is an option. (Ruf et al. 2021b, Sec. 4.3.)

[1] https://www.modalai.com/pages/voxl

3.6 Conclusion

In conclusion, with ReS^{2}tAC, this work presents an approach for real-time stereo processing on embedded ARM and CUDA devices, such as those attached to modern COTS UAVs. In this, a disparity estimation algorithm, which is based on the SGM approach (Hirschmüller 2005, Hirschmüller 2008), is optimized for embedded CUDA GPUs, such as the NVIDIA Tegra, by general-purpose computation on a GPU, as well as for embedded ARM CPUs by utilizing the NEON intrinsics for vectorized SIMD processing. It is demonstrated, that ReS^{2}tAC reaches state-of-the-art accuracies when evaluated on public stereo benchmark datasets. The CUDA-based implementation for stereo processing on embedded GPUs reaches real-time performance, even though it does not outperform related work in terms of processing speed. The frame rates of the NEON-based implementation, however, outperform all related work on stereo processing on embedded CPUs. In a use-case-specific scenario, the suitability of ReS^{2}tAC is demonstrated for real-time stereo estimation on a COTS UAV, namely the DJI Matrice 210v2 RTK equipped with a DJI Manifold 2-G. (Ruf et al. 2021b, Sec. 5.)

Furthermore, in the scope of this work, it is demonstrated that in case of rotor-based UAVs a modern embedded GPU is a suitable alternative to an embedded FPGA, especially due to its shorter and thus less expensive development cycles. Even though the GPU has a much greater power consumption than a FPGA and a significantly worse FPS/W ratio, its power consumption is negligible compared to the energy needed by rotor-based UAVs during flight and will reduce the flight time of the DJI Matrice 210v2 RTK by a maximum of 1 minute. However, for embedded systems with stricter power constraints, a FPGA-based approach should be considered. The experiments have also shown that, although the CPU requires less energy than the GPU, it has the worst FPS/W ratio. Thus, the optimization based on NEON intrinsics for vectorized SIMD processing should only be used if neither GPU nor FPGA are available. (Ruf et al. 2021b, Sec. 5.)

Finally, the experiments have revealed that a further investigation is needed in order to identify which part of the CUDA-based optimization needs to be further optimized, since ReS^{2}tAC does not reach the processing speeds of comparable approaches from the literature. Furthermore, in future, other approaches, e.g. deep-learning-based algorithms that are capable of reaching higher accuracies than ReS^{2}tAC, should also be considered. However, especially in case of approaches that are based on deep learning, experiments with respect to the generalization to unknown scenes and deployment on commodity hardware need to be conducted. Studies with respect to the reliability of the predictions that are produced by approaches that are based on deep learning are particularly important when considering safety-critical applications. (Ruf et al. 2021b, Sec. 5.)

As discussed, ReS^{2}tAC is primarily aimed to facilitate the perception of the environment of a COTS UAV or other vehicles and, in turn, allow a safe and autonomous use by using the estimated disparity or depth maps to perform obstacle detection and collision avoidance. A combination of ReS^{2}tAC with a simple and yet effective algorithm for the task of obstacle detection and collision avoidance is presented and discussed in Section 6.1. Even though, in the scope of this work, the use of ReS^{2}tAC is portrayed only with respect to a single use-case, there are also a lot of other possible applications, such as RGB-D odometry or rapid 3D mapping, that can be facilitated with a real-time disparity estimation algorithm.

4 Fast Multi-View Stereo Depth Estimation from Monocular Video Data

This chapter includes material from the following publication:

Ruf, B.; Weinmann, M., and Hinz, S. (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821*. Reprinted with permission. It is cited as (Ruf et al. 2021a) and marked with a green sidebar.

In the second part of the proposed framework (cf. Section 1.2), the visual payload camera is used to facilitate two high-level applications, namely rapid 3D mapping and 3D change detection. These applications, in turn, aim at supporting first responders during disaster relief or search-and-rescue (SAR) missions, by allowing a quick and large-scale assessment of the situation or enabling the monitoring of areas which are inaccessible for ground forces (Restas 2015, Furutani and Minami 2021). In this, image-based techniques and photogrammetry based on aerial reconnaissance are a key element in supporting the rescue workers, provided that the environmental conditions, e.g. whether and daytime, allow for a visual inspection (Furutani and Minami 2021). There exists a large collection of software toolboxes, such as COLMAP (Schönberger and Frahm 2016, Schönberger et al. 2016), for performing offline photogrammetric 3D reconstruction allowing to accurately reconstruct the disaster site from aerial imagery. Their focus, however, is primarily on offline and accurate processing. This hinders the use for a rapid 3D mapping during the image acquisition, due to the required run-time for high-accurate 3D modeling and the consideration of all input images for processing. However, in order to efficiently aid first responders in their tasks, the run-time of the algorithms matters which, in turn, raises the need for efficient, fast and incremental 3D mapping in order to support a rapid assessment. Here, the availability of 3D data, for example, allows to reason on damage caused by an incidence, or structural integrity of a partly collapsed building, as well as route planning through areas which are difficult to access or to account for 3D geometry when creating an orthographic map. (Ruf et al. 2021a, Sec. 1.)

4.1 Contributions and Outline

To accommodate the above-mentioned applications, this chapter presents a novel approach for fast multi-view stereo with surface-aware Semi-Global Matching, denoted as FaSS-MVS. The approach

- uses plane-sweep sampling to perform hierarchical dense multi-image matching,
- utilizes and extends the widely used SGM algorithm (Hirschmüller 2005, Hirschmüller 2008) to favor not only fronto-parallel surfaces in the computation of dense depth maps, by incorporating a surface-aware regularization based on local surface normals,
- efficiently computes dense depth, normal and confidence maps from image sequences, allowing to facilitate the task of incremental UAV-borne 3D mapping, and
- is quantitatively evaluated on two public datasets for dense multi-view stereo (MVS) with accurate ground truth and, additionally, results are presented for two use-case-specific datasets.

And even though FaSS-MVS is proposed with the discussed use-case in mind, it is not restricted to airborne data and can also be used to perform an incremental and online 3D mapping of an environment captured by a ground-based robot or sensor system. (Ruf et al. 2021a, Sec. 1.)

The contributions presented in this chapters have partly been published in:

- Ruf, B.; Weinmann, M., and Hinz, S. (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821*. Published as preprint. Cited as (Ruf et al. 2021a).
- Ruf, B.; Pollok, T., and Weinmann, M. (2019): "Efficient surface-aware semi-global matching with multi-view plane-sweep sampling". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W7, pp. 137–144. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2019).
- Ruf, B.; Thiel, L., and Weinmann, M. (2018a): "Deep cross-domain building extraction for selective depth estimation from oblique aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-1, pp. 125–132. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2018a).
- Ruf, B.; Erdnüß, B., and Weinmann, M. (2017): "Determining plane-sweep sampling points in image space using the cross-ratio for image-based depth estimation". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W6, pp. 325–332. Cited as (Ruf et al. 2017).

This chapter is structured as follows: In Section 4.2, the related work on incremental image-based 3D mapping for online processing as well as modern approaches for learning-based dense image matching (DIM) and MVS are briefly summarized. In this, it is also delineated, how the presented approach differs from those found in the related work. In Section 4.3, the overall processing pipeline of the presented approach is illustrated and outlined with a quick overview. This is followed by a detailed description on the implementation and methodology of the individual steps of the processing pipeline. The approach is quantitatively and qualitatively evaluated on two public and two private datasets. The datasets, the error metrics as well as the results of the conducted experiments are presented in Section 4.4. Subsequently, the findings are discussed and put into context of the considered use-case in Section 4.5, before providing a summary and concluding remarks as well as a short outlook on future work in Section 4.6. (Ruf et al. 2021a, Sec. 1.1.)

4.2 Related Work

Due to the ever-increasing demand for detailed 3D models, the research in the fields of photogrammetry, remote sensing and computer vision has brought up a number of software suites and applications, that focus on the estimation of accurate and dense depth and geometry information from a large set of input images, by means of DIM and MVS. Prominent and widely used representatives of such applications are MVE (Goesele et al. 2007), PMVS (Furukawa and Ponce 2010), SURE (Rothermel et al. 2012, Wenzel et al. 2013b), COLMAP (Schönberger et al. 2016) and OpenMVS (OpenMVS 2022), to name a few. These approaches, however, are designed for offline processing, aiming at the accuracy and completeness of the resulting 3D model, while assuming that all input data is available at the time of reconstruction and that no critical constraints on the computation time or hardware resources are set. In contrast, the aim of FaSS-MVS is to extract dense depth and geometry information from image sequences, while they are acquired. Or at least while the image data stream is received.

For example, in case a direct processing is not possible due to the acquisition by a small UAV and its limited hardware resources. Thus, the focus lies in the incremental and online processing of the input data by DIM and MVS. (Ruf et al. 2021a, Sec. 1.2.)

4.2.1 Incremental Camera-Based Mapping for Online Processing

Early work on incremental and online camera-based mapping of the local environment was mainly driven by robotic and augmented reality (AR) applications (Klein and Murray 2007, Davison et al. 2007, Eade and Drummond 2006). Here, the main goal was to robustly localize the camera pose, and in turn the sensor carrier, with respect to its surrounding, in order to navigate through the environment or enhance the camera images with additional information. Since the focus of these so-called simultaneous localization and mapping (SLAM) algorithms is the estimation of the camera pose and trajectory, the detailed and dense mapping of the environment was rather of secondary interest. Thus, these approaches mainly relied on point features for the tracking and mapping rather than direct pixel matching. However, in order to provide a convincing AR experience, a dense and detailed model of the environment is essential. Subsequent works (Newcombe and Davison 2010, Newcombe et al. 2011) have proposed a dense mapping simultaneous to the acquisition of the image data and the localization of the camera, resulting in a detailed reconstruction of a small AR workspace. Since these approaches, however, aim to reconstruct rather small-scale environments, they make use of short baseline video clips for the image matching, which in turn allows to rely on dense optical flow methods to find dense pixel correspondences (Newcombe and Davison 2010). In contrast, as input to FaSS-MVS, it is assumed to have image data captured by a UAV, which is typically flying several tens of meters away from the object of interest. Thus, FaSS-MVS is rather aimed to densely map a large-scale environment, which in turn hinders to track pixel-wise correspondences between consecutive frames, but requires a wide-baseline image matching instead. However, FaSS-MVS is not solely restricted to large-scale environments and a wide-baseline image matching, as experiments with respect to the employed multi-image matching, done in other work (Ruf et al. 2017), show. (Ruf et al. 2021a, Sec. 1.2.1.)

Early work on camera-based mapping and reconstruction of urban surroundings was done by Gallup et al. (2007) and Pollefeys et al. (2008), who employed the plane-sweep algorithm for true multi-image matching (cf. Section 2.1.3) to map and reconstruct building facades from images captured by a vehicle-mounted camera in real-time. In this, they rely on vanishing points, which are detected in the input images, and on data from an additional IMU to recover the orientations of the building facades and the ground plane relative to the camera. To find the optimal plane configuration for each pixel and, in turn, extract a depth map from the results of the DIM, Pollefeys et al. (2008) employ a Bayesian formulation with a subsequent selection of the WTA solution, while Gallup et al. (2007) minimize a formulated energy functional. In (Pollefeys et al. 2008), the estimation of the camera poses is done by using a Kanade-Lucas-Tomasi (KLT) feature tracker. A big advantage in the mapping and reconstruction of urban areas is that most objects in such scenery can be approximated well by planar structures, which is why plane-sweep DIM is well-suited for this task. Other approaches for urban reconstruction from ground-based imagery, like those from Furukawa et al. (2009), Sinha et al. (2009) and Gallup et al. (2010), perform a piece-wise planar reconstruction by fitting multiple, differently oriented planes into the scene and optimizing photometric consistency. In this, they minimize an energy functional by a graph-cut algorithm, which takes a couple of minutes on a commodity CPU. Algorithms having a couple of minutes runtime to estimate a single depth map do not seem to be suitable for fast and online processing at first sight. However, depending on their ability to be parallelized and optimized for the execution on a GPU, they might be useful after all. (Ruf et al. 2021a, Sec. 1.2.1.)

Around the same time as the previously mentioned work was released, Hirschmüller (2005) and Hirschmüller (2008) proposed the so-called SGM algorithm (cf. Section 2.2.1), which evolved into one of the most widely used approaches for both online and offline DIM, due its efficiency and convincing results. It has been deployed on both desktop (Spangenberg et al. 2014, Banz et al. 2011) and embedded (Hernandez-Juarez et al. 2016, Zhao et al. 2020, Ruf et al. 2021b) hardware as already discussed in Section 3.2. Moreover, the SGM algorithm is used in a wide range of applications, such as advanced driver assistance system (ADAS) (Spangenberg et al. 2014), real-time obstacle detection and collision avoidance on-board UAVs (Barry et al. 2015, Ruf et al. 2018b) and urban mapping and reconstruction from aerial imagery (Rothermel et al. 2012, Wenzel et al. 2013b, Haala et al. 2015). In their work, Sinha et al. (2014) combine the plane-sweep multi-image matching with the SGM algorithm to estimate dense and highly accurate disparity maps. In contrast to the presented approach, Sinha et al. (2014) use local slanted planes, which are extracted from feature correspondences, to create disparity hypotheses and employ the SGM algorithm to recover a disparity map. They, evaluate their approach on a high-resolution stereo benchmark and achieve significant improvement over the standard SGM algorithm in both run-time and accuracy. The improvement in terms of run-time is attributed to the fact that the local plane-sweep allows to test a locally confined part of the complete disparity range for each pixel, thus reducing the computational complexity of the optimization within the SGM algorithm. Similar improvements to overcome the problem of high computational complexity due to a large disparity range, which is inherent to oblique aerial imagery, were done by Haala et al. (2015), by embedding the SGM into a hierarchical coarse-to-fine processing. (Ruf et al. 2021a, Sec. 1.2.1.)

Even though a large number of urban environments can be well abstracted by piecewise planar reconstructions, not all structures are fronto-parallel, meaning that their surface orientations are not parallel to the image plane. In order to account for slanted surfaces, Kuschk and Cremers (2013), for example, have incorporated a second-order smoothness assumption into their energy function. The initial formulation of the SGM algorithm, however, only models a first-order smoothness term and thus favors fronto-parallel surfaces, leading to stair-casing artifacts when reconstructing slanted surfaces. Especially when aiming for a visually appealing reconstruction of the environment, this is to be avoided. While Hermann et al. (2009) and Ni et al. (2018) propose to incorporate a second-order smoothness assumption into the formulation of the SGM energy function, Scharstein et al. (2017) propose a more simplistic and yet effective improvement to address this issue. More specifically, plane priors are used, which, for example, can be recovered from normal maps or point correspondences, to adjust the zero-cost transition within the path aggregation of the SGM, thus penalizing deviations from the surface orientation represented by the prior. The major advantage over the other approaches is that the pixel-wise offset for the zero-cost transition can be calculated in advance and is in its magnitude the same for opposite aggregation paths, making its use very efficient. (Ruf et al. 2021a, Sec. 1.2.1.)

4.2.2 Learning of Dense Image Matching and Multi-View Stereo Reconstruction

With the advancements and success of deep-learning-based methods in other topics of computer vision, remote sensing and photogrammetry, such as object detection, classification or image segmentation, it was just a matter of time when the first learning-based approaches for the task of DIM and MVS, that would outperform state-of-the-art model-based approaches, would be presented. Early works (Han et al. 2015, Zbontar and LeCun 2016, Hartmann et al. 2017) use deep CNNs to learn similarity measures between image patches and, in turn, build up a 3D cost volume from which a disparity or depth map is extracted by conventional methods, e.g. SGM (Hirschmüller 2005, Hirschmüller 2008). (Ruf et al. 2021a, Sec. 1.2.2.)

Early approaches to perform actual MVS with deep learning are the so-called MVSNet (Yao et al. 2018) and DeepMVS (Huang et al. 2018). Both use the plane-sweep algorithm to match the pixels of multiple input images based on learned features and similarity measures and to build up a cost volume, just like the conventional approaches. To regularize the computed cost volume and to extract the depth map, both use a 3D U-Net (Ronneberger et al. 2015). 3D CNNs, such as the 3D U-Net, use a great amount of memory and are computationally not very efficient, which is why other approaches, such as the ones presented in (Yao et al. 2019, Yan et al. 2020), exchange the 3D U-Net by a cascade of 2D CNNs. Further approaches (Cheng et al. 2020, Gu et al. 2020) remedy the high memory consumption by the use of hierarchical coarse-to-fine processing, as also done by FaSS-MVS. In the construction of the cost volume, there also exist other strategies, such as using gated convolution (Yi et al. 2020) or reprojecting the image data into a 3D voxel grid (Ji et al. 2017). (Ruf et al. 2021a, Sec. 1.2.2.)

What all these approaches have in common, however, is that they are trained in a supervised manner, requiring datasets with appropriate ground truth. Most of them are using, among others, the DTU MVS benchmark (Jensen et al. 2014), which also serves as evaluation dataset for FaSS-MVS. The availability and versatility of appropriate datasets, however, is not very high, especially with respect to real-world scenarios, which still greatly hinders the practical use of deep-learning-based MVS approaches. To overcome this problem, recent approaches, such as the ones presented in (Khot et al. 2019, Huang et al. 2021) try to train models in an unsupervised, or sometimes also denoted as self-supervised, manner. But again, their practical use and ability for generalization still needs more studies (Khot et al. 2019). These limitations are the reasons why learning-based approaches for the task of MVS are not yet practical for the considered use-case, namely to reliably assist emergency forces in the incremental and online mapping of the operational area. (Ruf et al. 2021a, Sec. 1.2.2.)

In summary, FaSS-MVS uses a plane-sweep algorithm similar to the one presented by Pollefeys et al. (2008) to perform efficient dense multi-image matching and employs an improved implementation of the SGM algorithm to extract the depth map from the results of the DIM. The use of a plane-sweep algorithm for the task of DIM is mainly motivated by its ability to create depth hypotheses by matching an arbitrary number of input images as well as the fact that it can efficiently be optimized for the massively parallel execution on GPUs, making particularly suitable for online processing. In the improved implementation of the SGM algorithm, this work, among others, adopts the approach presented by Scharstein et al. (2017) to account for non-fronto-parallel surfaces by adjusting the zero-cost transition based on surface information stored inside a normal map. Very similar to FaSS-MVS seems to be the approach from Roth and Mayer (2019). They also rely on the improvements proposed by Scharstein et al. (2017) and combine the SGM with a plane-sweep DIM. However, their work focuses on the estimation of disparity images from ground-based stereo image pairs and was only evaluated on synthetic scenes so far. Moreover, this work also proposes to reduce the fronto-parallel bias of the SGM algorithm, by adjusting the zero-cost transition in the path aggregation based on the gradient of the minimum cost path. The complete depth estimation pipeline is embedded in a hierarchical processing scheme, just as proposed by Haala et al. (2015), in order to reduce the computational complexity induced by a large scene depth inherent to oblique aerial imagery. (Ruf et al. 2021a, Sec. 1.2.2.)

4.3 FaSS-MVS – Fast Multi-View Stereo with Surface-Aware Semi-Global Matching

In Section 4.3.1, first an overview on the processing pipeline for fast multi-view stereo using plane-sweep multi-image matching and surface-aware SGM is given. A detailed description of the dense multi-image matching by means of plane-sweep sampling is given in Section 4.3.2, followed by an in-depth description of the surface-aware extension of the SGM algorithm in Section 4.3.3. In addition to the depth map, FaSS-MVS also computes a normal map and a confidence map which is described in Section 4.3.4 and Section 4.3.5, respectively. Lastly, in Section 4.3.6, the employed post-processing steps to filter further outliers are described.

4.3.1 Overview on the full Processing Pipeline

An overview of the processing pipeline of the presented approach for fast MVS, based on plane-sweep sampling and a surface-aware SGM optimization (FaSS-MVS) is illustrated by Figure 4.1. Given an input bundle $(\mathcal{I}, \mathbf{P})_k \in \Omega$, consisting of k input images $\mathcal{I}$, extracted from an image sequence in sequential order, as well as corresponding camera poses $\mathbf{P}$, the approach computes depth, normal and confidence maps $(\mathcal{D}, \mathcal{N}, \mathcal{C})$ for a defined reference image $\mathcal{I}_{\text{ref}}$, which is typically the middle one of the input bundle Ω. In this, it is assumed that the input has been calibrated, i.e. that the images are free of lens distortion and that the full projection

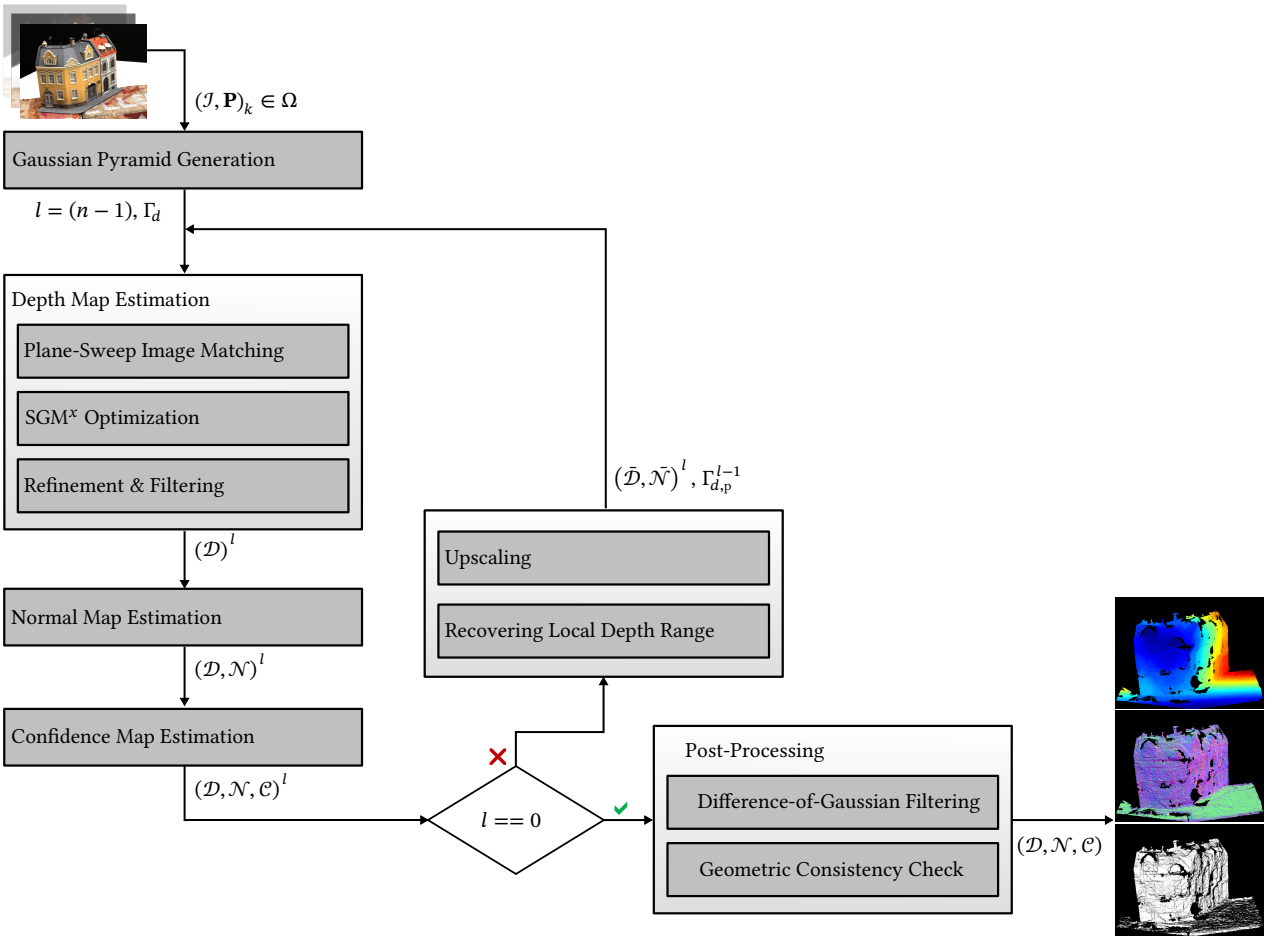


Figure 4.1: Overview of the processing pipeline for Fast Multi-View Stereo with Surface-Aware Semi-Global Matching (FaSS-MVS). Given a bundle of images and corresponding camera poses $(\mathcal{I}, \mathbf{P})_k$ of an input sequence, a hierarchical MVS estimation is performed to recover a depth, normal and confidence map $(\mathcal{D}, \mathcal{N}, \mathcal{C})$. (Ruf et al. 2021a, Fig. 1.)

matrix $\mathbf{P}_k = \mathbf{K}\left[\mathbf{R}_k^\intercal \quad -\mathbf{R}_k^\intercal \mathrm{C}_k\right]$ of each image is known. Here, $\mathrm{C}_k \in \mathbb{R}^3$ denotes the locations of the camera centers with respect to a reference coordinate system $\mathrm{O}_{\mathrm{ref}}$, while the column vectors of $\mathbf{R}_k^\intercal \in SO(3)$ hold the normalized coordinate axes of the camera coordinate system $\mathrm{O}_{\mathrm{cam}}$, as seen from $\mathrm{O}_{\mathrm{ref}}$. The intrinsic calibration matrix $\mathbf{K}$ is equal for all cameras, since it is assumed that the images are captured from a single camera. Typical for an MVS approach, it is assumed that the input images depict the scene which is to be reconstructed from slightly different viewpoints. (Ruf et al. 2021a, Sec. 2.1.)

Before any processing, a Gaussian image pyramid with n pyramid levels is computed for each image of the input bundle, allowing a subsequent hierarchical processing. The lowest pyramid levels ($l = 0$) hold the input images with their original image size. While moving up the pyramid, the images are first blurred with a Gaussian filter of size of 3×3 pixels and with $\sigma = 1$, before being scaled down by a factor of 0.5 in both image directions. Furthermore, the intrinsic calibration matrices are also adjusted by halving the focal length and the coordinates of the principal point in order to account for the reduced image size. This results in an augmentation of the input bundle Ω by $n - 1$ additional sets. In the following, a superscript is used to mark the results and processes at a specific pyramid level. The pipeline is initialized at the coarsest pyramid level $l = (n-1)$ with the smallest image size, executing three subsequent computational parts at each pyramid level, thus resulting in a coarse-to-fine processing. (Ruf et al. 2021a, Sec. 2.1.)

The first part of the actual processing, namely the depth estimation, computes a depth map $\mathcal{D}^l$ and is, in turn, subdivided into a plane-sweep multi-image matching creating depth hypotheses and the SGM^x optimization extracting the optimal depth from the set of hypotheses. The latter one adopts the SGM approach, which was first presented by Hirschmüller (2005) and Hirschmüller (2008), to the plane-sweep matching and extends it to account for non-fronto-parallel surface structures. Section 4.3.2 gives a detailed description on the employed plane-sweep algorithm, while the extension of the SGM algorithm is discussed in Section 4.3.3. A concluding depth refinement and median filter with a kernel size of 5×5 pixels is used to filter small outliers in the resulting depth map. (Ruf et al. 2021a, Sec. 2.1.)

The second part of the hierarchical processing estimates a normal map $\mathcal{N}^l$ from the previously computed depth map $\mathcal{D}^l$. Apart from being an additional output of FaSS-MVS, the normal map is also used as part of the SGM^x optimization to account for the surface orientation in the next hierarchical iteration. The normal map is regularized by an appearance-based weighted Gaussian smoothing in order to smooth out small variations while preserving discontinuities. Section 4.3.4 describes the details on how the normal map is extracted from a single depth map. (Ruf et al. 2021a, Sec. 2.1.)

In the third part, a confidence map $\mathcal{C}^l$ is computed, holding pixel-wise confidence scores in the interval of $[0, 1]$ with respect to the depth estimates in $\mathcal{D}^l$. The final confidence scores are computed based on the surface orientation at the considered pixel. Details on the computation of $\mathcal{C}^l$ are given in Section 4.3.5. (Ruf et al. 2021a, Sec. 2.1.)

Inherent to a hierarchical coarse-to-fine processing, the depth map $\mathcal{D}^l$ and normal map $\mathcal{N}^l$ computed at level l are used to initialize the depth map estimation at the next pyramid level $l - 1$, as long as the lowest levels of the image pyramids are not yet reached, i.e. while $l > 0$. In this, $\mathcal{D}^l$ and $\mathcal{N}^l$ are upscaled with nearest neighbor interpolation to the image size of the next pyramid level, yielding $\bar{\mathcal{D}}^l$ and $\bar{\mathcal{N}}^l$. Then, $\bar{\mathcal{D}}^l$ is first used to compute the pixel-wise sampling range $\Gamma_{d,\mathrm{p}}^{l-1}$ of the multi-image plane-sweep algorithm at the next pyramid level. Here, the $\Gamma_{d,\mathrm{p}}^{l-1}$ is computed for each pixel p separately, based on the previous depth estimate $\bar{d}_{\mathrm{p}}^{l} = \bar{\mathcal{D}}^l(\mathrm{p})$

and a predefined window with a radius of Δd around $\bar{d}_{\mathrm{p}}^{\,l}$:

$$\begin{aligned} \Gamma_{d,\mathrm{p}}^{l-1} &= \left[d_{\mathrm{p,min}}^{l-1}, d_{\mathrm{p,max}}^{l-1}\right], \text{with} \\ d_{\mathrm{p,min}}^{l-1} &= \bar{d}_{\mathrm{p}}^{\,l} - \Delta d, \\ d_{\mathrm{p,max}}^{l-1} &= \bar{d}_{\mathrm{p}}^{\,l} + \Delta d. \end{aligned} \tag{4.1}$$

In the first iteration, the sampling range is set equally for all pixels and is parameterized by minimum and maximum scene depth: $\Gamma_d = [d_{\mathrm{min}}, d_{\mathrm{max}}]$. The upscaled normal map $\bar{\mathcal{N}}^l$ is used by one of the proposed SGM extensions to account for the surface orientation within the scene. The final depth, normal and confidence maps are the outcome of the processing at the lowest pyramid level. They are denoted as $\mathcal{D}$, $\mathcal{N}$ and $\mathcal{C}$ respectively, and have the same image size as the input images. (Ruf et al. 2021a, Sec. 2.1.)

In a final post-processing step, a DOG filter (cf. Section 2.3.2.2) is used to unmask image regions, which do not have distinctive texture information that allows to perform a reliable matching, as well as a geometric consistency check (cf. Section 2.3.3). (Ruf et al. 2021a, Sec. 2.1.)

4.3.2 Real-Time Dense Multi-Image Matching with Plane-Sweep Sampling

Dense image matching refers to the process of finding a dense correspondence field between pixels of two or more images from which depth hypotheses are extracted (cf. Section 2.1). If the relative camera poses of input images are known, these correspondences can further be transformed into a depth map, which encodes the pixel-wise scene depth from the vantage point of the reference camera for which the correspondence field and depth map are computed. As described in Section 2.1.3, Collins (1996) proposed the so-called plane-sweep algorithm for true multi-image matching to compute depth hypotheses. FaSS-MVS adopts the plane-sweep algorithm for real-time multi-image matching and embeds it into a hierarchical coarse-to-fine processing scheme in order to efficiently handle large scene depths (Section 4.3.2.1). Moreover, FaSS-MVS utilizes the cross-ratio, which is invariant under perspective projection, to efficiently compute the distance of the sampling planes with respect to the reference camera for an effective sampling of the scene-space (Section 4.3.2.3). (Ruf et al. 2021a, Sec. 2.2.)

4.3.2.1 The Hierarchical Plane-Sweep Algorithm for Real-Time Multi-Image Matching

The presented approach for hierarchical multi-image matching is based on the plane-sweep algorithm presented by Pollefeys et al. (2008) and is described in Algorithm 4.1. As part of the actual image matching, the Hamming distance of the census transform (CT) (Zabih and Woodfill 1994) (cf. Section 2.1.1) as well as a negated, truncated and scaled form of the normalized cross correlation (NCC) (Scharstein et al. 2017, Sinha et al. 2014) (cf. Section 2.1.2) are employed and evaluated as cost function $C(\cdot)$. Since FaSS-MVS considers a bundle of input images with an equal number of matching images to either side of the reference image, the approach presented by Kang et al. (2001) is adopted in order to account for occlusions, using the minimum aggregated matching cost of the left and right subset of the matching images. The resulting three-dimensional cost volume $\mathcal{S}^l$ is of size $w^l \times h^l \times |\delta^l|$, with w^l and h^l being the width and height of the reference image and $|\delta^l|$ being the number plane positions at which the matching is performed, all with respect to the current pyramid level l. In this, the cost volume $\mathcal{S}^l$ is implemented as a dynamic cost volume (Haala et al. 2015) for all but the topmost pyramid level, since the sampling range $\Gamma_{d,\mathrm{p}}^l$ is determined independently for each pixel p. Nonetheless, the complete set of plane distances $\delta \in [\delta_{\mathrm{min}}, \delta_{\mathrm{max}}]$, deduced from Γ_d, are precomputed for each

Algorithm 4.1: Plane-sweep multi-image matching executed at a specific pyramid level l of the proposed hierarchical processing scheme. (Ruf et al. 2021a, Alg. 1.)

Input Data: a calibrated image bundle Ω^l at the pyramid level l, a set of planes Π with a normal vector N_Π and varying distances δ as well as a local depth sampling range $\Gamma^l_{d,\mathrm{p}} = [d^{\,l}_{\mathrm{p,min}}, d^{\,l}_{\mathrm{p,max}}]$.

Result: three-dimensional cost volume $\mathcal{S}$, holding the pixel-wise matching score for each pixel $\mathrm{p}^{\mathrm{ref}} \in \mathcal{I}^l_{\mathrm{ref}}$ and plane Π.

· determine bounding planes Π_{min} and Π_{max} located at δ_{min} and δ_{max}, so that the local depth range $\Gamma^l_{d,\mathrm{p}}$ is completely sampled (cf. Section 4.3.2.2).

foreach *pixel* $\mathrm{p}^{\mathrm{ref}} \in \mathcal{I}^l_{\mathrm{ref}}$ **and** *distance* $\delta \in [\delta_{\mathrm{min}}, \delta_{\mathrm{max}}]$ **do**

· configure scene plane $\Pi = (\mathrm{N}_\Pi, \delta)$.

· determine pixels p^k in all matching images $\mathcal{I}^l_k \in \Omega^l \setminus \mathcal{I}^l_{\mathrm{ref}}$ according to the homography $\mathbf{H}$ (cf. Equation (2.7)):

$$\mathrm{p}^k = \mathbf{H}\left(\Pi, \mathbf{P}^l_{\mathrm{ref}}, \mathbf{P}^l_k\right) \cdot \mathrm{p}^{\mathrm{ref}}.$$

· warp local image patches $\mathcal{P}^l_k \in \mathcal{I}^l_k$ around p^k, with the same size as the support region of the matching cost function $C(\cdot)$, into $\mathcal{I}^l_{\mathrm{ref}}$:

$$\tilde{\mathcal{P}}^l_k = \mathbf{H}\left(\Pi, \mathbf{P}^l_{\mathrm{ref}}, \mathbf{P}^l_k\right)^{-1} \cdot \mathcal{P}^l_k.$$

· compute the matching cost $s\,(\mathrm{p}, \Pi)$ between the reference patch $\mathcal{P}^l_{\mathrm{ref}} \in \mathcal{I}^l_{\mathrm{ref}}$ and $\tilde{\mathcal{P}}^l_k$ for left and right subset of cameras separately :

$$s^{\mathrm{L}}(\mathrm{p}, \Pi) = \sum_{k<\mathrm{ref}} C\left(\mathcal{P}^l_{\mathrm{ref}}, \tilde{\mathcal{P}}^l_k\right),$$

$$s^{\mathrm{R}}(\mathrm{p}, \Pi) = \sum_{k>\mathrm{ref}} C\left(\mathcal{P}^l_{\mathrm{ref}}, \tilde{\mathcal{P}}^l_k\right).$$

· store the minimum of left and right matching cost (accounting for occlusions as described by Kang et al. (2001)) into the three-dimensional cost volume $\mathcal{S}$:

$$\mathcal{S}^l(\mathrm{p}, \Pi) = \min\{s^{\mathrm{L}}(\mathrm{p}, \Pi), s^{\mathrm{R}}(\mathrm{p}, \Pi)\}.$$

end

pyramid level l and are the same for all pixels. This, in turn, allows to precompute the homographic mappings for all planes Π. The following sections describe how the bounding planes Π_{min} and Π_{max} and the corresponding distances δ_{min} and δ_{max} are found (Section 4.3.2.2), as well as how the cross-ratio is used to find appropriate sampling steps within Γ_d (Section 4.3.2.3). (Ruf et al. 2021a, Sec. 2.2.1.)

4.3.2.2 Determining the Bounding Planes Corresponding to the Given Depth Range

As previously described, it is assumed that two bounding planes, namely Π_{min} and Π_{max} with corresponding distances δ_{min} and δ_{max}, between which the scene is to be sampled, are known. In case of a fronto-parallel sampling strategy, i.e. $\mathrm{N}_\Pi = (0\ 0\ {-1})^{\mathrm{T}}$ with respect to the local camera coordinate system, the distances δ_{min} and δ_{max} are equal to the minimum and maximum depth, namely d_{min} and d_{max}. This, however, does not hold for non-fronto-parallel plane orientations. To find the bounding planes for slanted plane orientations, a view-frustum, corresponding to the reference camera, is first constructed for which the depth is to be estimated. This view-frustum is represented by a pyramid that resembles the field-of-view of the camera and that is truncated by two fronto-parallel near and far planes that are located at d_{min} and d_{max}. Given the four corner points of the

view-frustum on the near plane M_i^{near} and the four on the far plane M_i^{far}, the minimum and maximum distance δ_{min} and δ_{max} are found according to:

$$\begin{aligned} \delta_{min} &= \min_i(|N_\Pi^\intercal \cdot M_i^{near}|)\,, \text{ and} \\ \delta_{max} &= \max_i(|N_\Pi^\intercal \cdot M_i^{far}|). \end{aligned} \tag{4.2}$$

In order to avoid an orientation flip of the image data, all camera centers C_i need to lie in front of Π_{min} with respect to the sweeping direction, thus for all camera centers $N_\Pi^\intercal \cdot C_i + \delta_{min} > 0$ must hold. (Ruf et al. 2021a, Sec. 2.2.2.)

4.3.2.3 Finding the Sampling Steps by Utilizing the Cross-Ratio

As stated by Equation (2.7), the sampling planes Π of the plane-sweep algorithm are parameterized by two parameters, namely the normal vector N_Π, denoting the orientation and sweeping direction of the plane, and the orthogonal distance δ from the optical center of the reference camera C_{ref}. While N_Π allows to adjust the warping of the images and, thus, the image matching to the surface orientation within the scene, the second parameter δ determines the step-size with which the scene is sampled. In this, a straight-forward approach would be to select the step-size in such a way that the scene is sampled with a desired resolution, i.e. sweeping the planes at equidistant unit intervals through the scene-space. However, it is not guaranteed that a thorough sampling of the scene with a small step-size results in a higher accuracy. If the step-size is not chosen in accordance with the camera positions of the input images and the baseline between the cameras, the matching results of two or more consecutive plane positions might not reveal enough difference and, thus, introduce ambiguities between multiple plane hypotheses. Furthermore, in terms of efficiency, it is important to vary the sampling rate in scene-space with respect to the distance of the plane relative to the reference camera, since the perspective projection requires an increasingly smaller step-size as the plane moves closer to the camera. (Ruf et al. 2021a, Sec. 2.2.3.)

Thus, a common approach is to select the sampling positions of the planes according to the disparity change induced by two consecutive planes. In this, the pixel-wise motion between the warped images of two consecutive planes should not exceed an absolute value of 1 (Szeliski and Scharstein 2004, Pollefeys et al. 2008). To ensure that the maximum disparity change between the warped images of two consecutive planes is less or equal to 1, Pollefeys et al. (2008) evaluate the displacements occurring at the boundaries of the most distant images for a set of planes with predefined parameters and only select those that fulfill the stated criterion for the actual image matching. Thus, for each plane within the predefined set, an additional test is performed, involving the warping of image points at the boundaries, in order to determine whether the plane is suitable or not. (Ruf et al. 2021a, Sec. 2.2.3.)

In contrast, FaSS-MVS aims to directly derive the distances of the sampling planes from correspondences in image space, implicitly leading to an inverse depth sampling in scene-space. Given an image point $m^{ref} \in \mathcal{I}_{ref}$ in the reference image, an intuitive approach would be to select multiple sampling points $m^k \in \mathcal{I}_k$ on the corresponding epipolar line l_m^k in one of the other cameras, and find the corresponding plane distances by triangulation between m^{ref} and m^k. The effort of triangulation, however, can be avoided by relying on the cross-ratio which is invariant under perspective projection. (Ruf et al. 2021a, Sec. 2.2.3.)

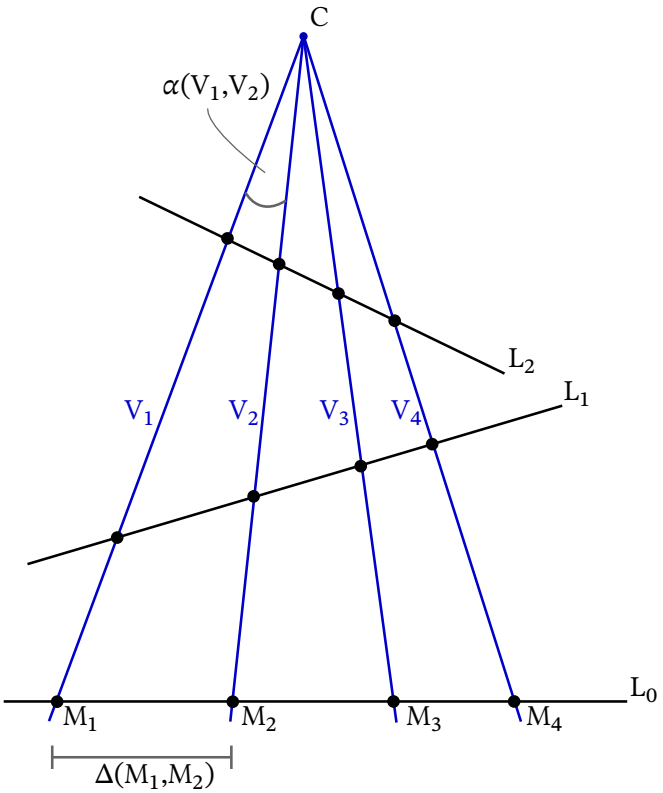


Figure 4.2: Illustration of the cross-ratio. The cross-ratio between four collinear points is the same on all three lines and thus invariant under the perspective projection. (Ruf et al. 2021a, Fig. A12.)

The Cross-Ratio as Invariant under Perspective Projection:

Given four collinear points, the cross-ratio describes the relative distance between them. While under perspective projection the relative distances between the points change, their cross-ratio does not change and is thus invariant. As illustrated by Figure 4.2, four points M_1, M_2, M_3 and M_4 that lie on a straight line L_0 are perspectively projected onto further non-parallel lines, e.g. L_1 and L_2. If the distance $\Delta(M_i,M_j)$ between two points M_i and M_j is known, the cross-ratio $Q(\cdot)$ is calculated according to:

$$Q(M_1, M_2, M_3, M_4) = \frac{\Delta(M_1,M_3) \cdot \Delta(M_2,M_4)}{\Delta(M_1,M_4) \cdot \Delta(M_2,M_3)}. \tag{4.3}$$

It is the same on all three lines. Thus, if the positions of the points on one line are known, their positions on the other lines can be deduced. Moreover, given the four rays V_1, V_2, V_3 and V_4, which are going through the center of projection C and each of the four points, as well as their pairwise enclosed angles $\alpha(V_i,V_j)$, the cross-ratio can be extended to:

$$\begin{aligned} Q(M_1, M_2, M_3, M_4) &= Q(V_1, V_2, V_3, V_4) \\ &= \frac{\sin\alpha(V_1,V_3) \cdot \sin\alpha(V_2,V_4)}{\sin\alpha(V_1,V_4) \cdot \sin\alpha(V_2,V_3)}. \end{aligned} \tag{4.4}$$

(Ruf et al. 2021a, Appendix A.)

Finding Plane Distances by Utilizing the Cross-Ratio:

Given the projection matrices $\mathbf{P}_{\text{ref}}$ and $\mathbf{P}_k$ of two cameras, as illustrated in Figure 4.3, the coordinate system is first centered in the optical center of the reference camera C_{ref}, so that $\mathbf{P}_{\text{ref}} = \mathbf{K}\,[\mathbf{I}\ 0]$. Thus, the plane distances determined by this approach are relative to C_{ref}. In case of MVS with multiple cameras, the camera which will induce the largest image offset and thus gives an upper bound on the disparity range is selected, for C_k. As already noted by Pollefeys et al. (2008), this is typically the camera which is most distant from the reference camera. (Ruf et al. 2021a, Sec. 2.2.3.)

Furthermore, it is again assumed that two bounding planes Π_{min} and Π_{max}, which limit the sweep space of the planes, are given. Just as in the work of Pollefeys et al. (2008), it is ensured that the sweep space does not intersect the convex hull of the cameras, in order to avoid inversions in the process of image matching (cf. Section 4.3.2.2). The image point m^{ref} is chosen as the pixel which induces the largest disparity when warped from $\mathcal{I}_{\text{ref}}$ to $\mathcal{I}_k$ via $\mathbf{H}\,(\Pi_{\text{min}}, \mathbf{P}_{\text{ref}}, \mathbf{P}_k)$, which is typically one of the four corners. This guarantees a maximum disparity change between successive planes and allows to find a set of sampling planes, with distances δ relative to C_{ref} according to Algorithm 4.2 and illustrated by Figure 4.3. (Ruf et al. 2021a, Sec. 2.2.3.)

This approach is computationally efficient and not restricted to a fronto-parallel orientation of the sampling planes as long as the optical axis of the reference camera intersects with the planes and the sweeping vector has a component that is parallel to the optical axis. Furthermore, in order to accommodate for all possible setups of C_{ref} and C_k, it is important to use $Q(\mathrm{V}^k_{\mathrm{e}_{\text{ref}}}, \mathrm{V}^k_{\mathrm{m}_{\text{min}}}, \mathrm{V}^k_{\mathrm{m}_i}, \mathrm{V}^k_{\mathrm{m}_{\text{max}}})$ in Algorithm 4.2, since $\mathrm{e}^k_{\text{ref}}$ would flip to the side of $\mathrm{m}^k_{\text{max}}$ if the focal plane of the reference camera is behind C_k. (Ruf et al. 2021a, Sec. 2.2.3.)

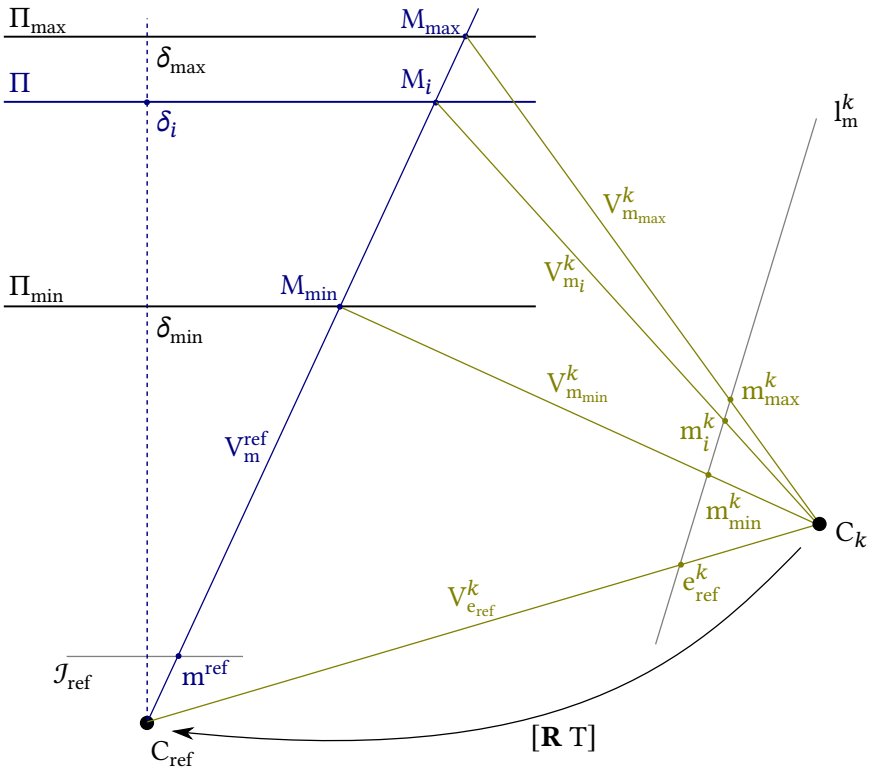


Figure 4.3: Illustration of determining the orthogonal distance parameter of the sampling planes of the plane-sweep multi-image matching by using the cross-ratio and epipolar geometry. Here, C_{ref} and C_k represent the positions of the optical centers of the two cameras. (Ruf et al. 2021a, Fig. 3.)

Algorithm 4.2: Finding plane distances δ by utilizing the cross-ratio. (Ruf et al. 2021a, Alg. 2.)

Input Data: two cameras with full projection matrices $\mathbf{P}_{\text{ref}}$ and $\mathbf{P}_k$, an image point m^{ref} inducing largest disparity when warped from $\mathcal{I}_{\text{ref}}$ to $\mathcal{I}_k$, as well as two bounding planes Π_{min} and Π_{max}.

Result: list of orthogonal plane distances δ relative to C_{ref}, such that the maximum pixel-displacement between the warped images of two consecutive planes is less than or equal to 1.

· calculate the viewing ray $\text{V}_{\text{m}}^{\text{ref}}$, going through C_{ref} and m^{ref}, and intersect it with Π_{min} and Π_{max}, yielding the scene points M_{min} and M_{max}.

· project the optical center C_{ref} as well as M_{min} and M_{max} onto the image plane of the second camera, yielding the epipole e_{ref}^k and the two image points m_{min}^k and m_{max}^k, all lying on the epipolar line l_{m}^k.

· determine the unit vector $\text{k} = \frac{\text{m}_{\text{min}}^k - \text{m}_{\text{max}}^k}{||\text{m}_{\text{min}}^k - \text{m}_{\text{max}}^k||}$, being the normalized direction of l_{m}^k, pointing from m_{max}^k to m_{min}^k.

for $\text{m}_i^k \leftarrow \text{m}_{\text{max}}^k$ **to** m_{min}^k **by** $\text{m}_{i+1}^k = \text{m}_i^k + \text{k}$ **do**

· given the viewing rays $\text{V}_{\text{e}_{\text{ref}}}^k$, $\text{V}_{\text{m}_{\text{min}}}^k$, $\text{V}_{\text{m}_i}^k$ and $\text{V}_{\text{m}_{\text{max}}}^k$ going through the optical center C_k and e_{ref}^k, m_{min}^k, m_i^k and m_{max}^k respectively, apply Equation (4.3) and Equation (4.4) to compute $\text{M}_i \in \text{V}_{\text{m}}^{\text{ref}}$ according to:

$$\begin{aligned} Q(\text{V}_{\text{e}_{\text{ref}}}^k, \text{V}_{\text{m}_{\text{min}}}^k, \text{V}_{\text{m}_i}^k, \text{V}_{\text{m}_{\text{max}}}^k) &= \frac{\sin(\alpha(\text{V}_{\text{e}_{\text{ref}}}^k, \text{V}_{\text{m}_i}^k)) \cdot \sin(\alpha(\text{V}_{\text{m}_{\text{min}}}^k, \text{V}_{\text{m}_{\text{max}}}^k))}{\sin(\alpha(\text{V}_{\text{e}_{\text{ref}}}^k, \text{V}_{\text{m}_{\text{max}}}^k)) \cdot \sin(\alpha(\text{V}_{\text{m}_{\text{min}}}^k, \text{V}_{\text{m}_i}^k))} \\ &= \frac{\Delta(\text{C}_{\text{ref}}, \text{M}_i) \cdot \Delta(\text{M}_{\text{min}}, \text{M}_{\text{max}})}{\Delta(\text{C}_{\text{ref}}, \text{M}_{\text{max}}) \cdot \Delta(\text{M}_{\text{min}}, \text{M}_i)}. \end{aligned}$$

· since $Q(\text{C}_{\text{ref}}, \text{M}_{\text{min}}, \text{M}_i, \text{M}_{\text{max}}) = Q(\text{C}_{\text{ref}}, \delta_{\text{min}}, \delta, \delta_{\text{max}})$, derive δ relative to C_{ref} according to:

$$\frac{\delta \cdot (\delta_{\text{max}} - \delta_{\text{min}})}{\delta_{\text{max}} \cdot (\delta - \delta_{\text{min}})} = \frac{\sin(\alpha(\text{V}_{\text{e}_{\text{ref}}}^k, \text{V}_{\text{m}_i}^k)) \cdot \sin(\alpha(\text{V}_{\text{m}_{\text{min}}}^k, \text{V}_{\text{m}_{\text{max}}}^k))}{\sin(\alpha(\text{V}_{\text{e}_{\text{ref}}}^k, \text{V}_{\text{m}_{\text{max}}}^k)) \cdot \sin(\alpha(\text{V}_{\text{m}_{\text{min}}}^k, \text{V}_{\text{m}_i}^k))}.$$

end

4.3.3 Depth Map Computation with Surface-Aware Semi-Global Matching

The hierarchical plane-sweep algorithm for multi-image matching, as described in Section 4.3.2, produces at each pyramid level a three-dimensional cost volume $\mathcal{S}^l(\text{p}, \Pi)$, which holds matching costs for each pixel $\text{p} \in \mathcal{I}_{\text{ref}}$ induced by a given plane Π located at distance δ orthogonal to the optical center C_{ref} of the reference camera. In the second stage of the depth estimation within FaSS-MVS, the cost volume is regularized by a semi-global optimization scheme, yielding a dense depth map $\mathcal{D}^l$. Building upon the original SGM approach (Hirschmüller 2005, Hirschmüller 2008) (cf. Section 2.2.1), three different optimization schemes (SGM^x) are proposed in the scope of this work. Apart from a straight-forward adaptation of the SGM approach to the plane-sweep sampling, FaSS-MVS adopts the approach of Scharstein et al. (2017) to also favor slanted surfaces by considering surface information available in the form of surface normals. Furthermore, a third extension penalizes deviations from the gradient of the minimum cost path within the SGM optimization scheme. (Ruf et al. 2021a, Sec. 2.3.)

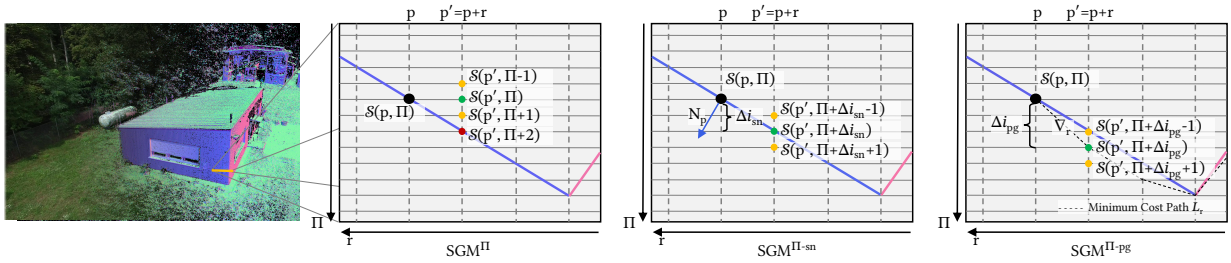


Figure 4.4: Illustration of the different path aggregation strategies along one path direction r within the three presented SGM^x optimization schemes. **Column 1:** Reference image and normal map of a building. Illustrated area marked with yellow line. **Column 2:** SGM$^\Pi$ path aggregation. The blue and pink lines represent the blue and pink surface orientations on the building facade. Aggregating the path costs for pixel p at plane Π, SGM$^\Pi$ will incorporate the previous costs at the same plane position (green) without additional penalty. The previous path costs at $I(\Pi) \pm 1$ (yellow) will be penalized with φ_1. The previous path costs located at $I(\Pi)+2$ (red), which is actually located on the corresponding surface, will be penalized with the highest penalty φ_2. **Column 3:** SGM$^{\Pi\text{-sn}}$ uses the normal vector $\mathrm{N_p}$, encoding the surface orientation at pixel p, and computes a discrete index jump Δi_{sn}, which ideally adjusts the zero-cost transition, causing the previous path costs at $I(\Pi)+2$ to not be penalized. **Column 4:** Similar to SGM$^{\Pi\text{-sn}}$, SGM$^{\Pi\text{-pg}}$ adjusts the zero-cost transition. However, the discrete index jump Δi_{pg} is derived from the running gradient ∇r of the minimum cost path. (Ruf et al. 2021a, Fig. 4.)

4.3.3.1 Extension of Semi-Global Matching to Plane-Sweep Matching and Different Surface Orientations

The following section describes in detail how the proposed SGM algorithm is adopted with respect to the creation of depth hypotheses with the help of plane-sweep multi-image matching, as well as the extensions that are employed to account for non-fronto-parallel surfaces. The proposed extensions SGM^x only affect the aggregation of the matching costs along the concentric paths. Thus, the extraction of the depth map $\mathcal{D}$ is done analogously to Equation (2.12) and Equation (2.9) of the SGM algorithm, with the disparity being substituted by depth. If a fronto-parallel plane orientation is considered during the plane-sweep, the depth can be directly extracted from the plane parameterization. However, for non-fronto-parallel orientations, the depth map $\mathcal{D}$ is computed by a pixel-wise intersection of the viewing rays with the corresponding WTA solutions (cf. Equation (2.8)). (Ruf et al. 2021a, Sec. 2.3.2.)

Resolving Plane Hypotheses with Semi-Global Matching:

Since the plane-sweep algorithm does not compute hypotheses on disparities, but rather pixel-wise plane distances relative to the reference camera and thus depth, the first SGM extension proposed is a straight-forward adaptation of the standard SGM algorithm to a multi-view plane-sweep sampling. In this, the formulation of the SGM path aggregation (cf. Equation (2.11)) is modified to

$$L_{\mathrm{r}}(\mathrm{p},\Pi) = \mathcal{S}(\mathrm{p},\Pi) + \min_{\Pi'}\left(L_{\mathrm{r}}(\mathrm{p}-\mathrm{r},\Pi') + V_\Pi(\Pi,\Pi')\right), \tag{4.5}$$

where Π denotes the sampling plane at distance δ. The smoothness term V_Π now penalizes the selection of different planes between adjacent pixels along the path L_{r}, instead of disparities. It is formulated as:

$$V_\Pi(\Pi,\Pi') = \begin{cases} 0 & , \text{if } I(\Pi) = I(\Pi') \\ \varphi_1 & , \text{if } |I(\Pi) - I(\Pi')| = 1 \\ \varphi_2 & , \text{if } |I(\Pi) - I(\Pi')| > 1, \end{cases} \tag{4.6}$$

with $I(\cdot)$ being a function that returns the index of Π within the set of sampling planes (cf. Figure 4.4, Column 2). This extension is denoted as *plane-wise* SGM (SGM^{Π}). Given a pixel-wise WTA plane parameterization, the corresponding depth is extracted by intersecting the viewing ray through pixel p with the corresponding plane according to Equation (2.8). (Ruf et al. 2021a, Sec. 2.3.2.)

Incorporating Surface Normals to Adjust the Zero-Cost Transition:

The smoothness term of the initial SGM algorithm is formulated with discrete disparity differences (cf. Equation (2.11)), penalizing discrete disparity jumps between neighboring pixels. In its optimization scheme, it does not consider any subpixel disparity and thus favors fronto-parallel surface structures, leading to stair-casing artifacts if no post-processing is employed (Scharstein et al. 2017). The same holds for the first extension, SGM^{Π}. Even though the plane-sweep sampling also supports non-fronto-parallel plane orientations, the smoothness term of SGM^{Π} (cf. Equation (4.6)) does not, and strongly penalizes index jumps in the sampling planes of more than 1. While this is desired if the plane-orientation coincides with the surface orientation, it will still lead to stair-casing artifacts if the surface and plane orientations do not align. In order to overcome the favoring of fronto-parallel structures and adjust the smoothness term of SGM to surfaces that are slanted with respect to the sampling direction, Scharstein et al. (2017) suggest to add an offset to the smoothness term. This offset can be extracted from additional information on the surface orientation, e.g. surface normals, that will make the zero-cost transition coincide with the surface orientation. In the scope of this work, this approach is adopted as part of the second extension. It is thus called *surface normal* SGM ($\text{SGM}^{\Pi\text{-sn}}$). (Ruf et al. 2021a, Sec. 2.3.2.)

In the presented hierarchical approach, the normal vectors from the normal map $\mathcal{N}^{l+1}$, which was estimated in the previous level of the pyramid (cf. Figure 4.1), are extracted. The pixel-wise normal vectors $\mathrm{N_p} = \mathcal{N}^{l+1}(\mathrm{p})$ indicate the surface orientation at the scene point M, which is computed by intersecting the viewing ray through p with the plane Π. From this, the discrete index jump Δi_{sn} through the set of sampling planes can be calculated which is caused by the tangent plane to $\mathrm{N_p}$. Since the plane-sweep sampling is not restricted to fronto-parallel plane orientations, the index jump Δi_{sn} needs to be computed based on the difference between the tangent plane at M and the orientation of the sampling planes in the direction r of the currently considered aggregation path. With Δi_{sn}, the smoothness term used by the extension $\text{SGM}^{\Pi\text{-sn}}$ is adjusted according to

$$V_{\Pi\text{-sn}}(\Pi,\Pi') = \begin{cases} 0 & \text{, if } I(\Pi) + \Delta i_{\text{sn}} = I(\Pi') \\ \varphi_1 & \text{, if } |I(\Pi) + \Delta i_{\text{sn}} - I(\Pi')| = 1 \\ \varphi_2 & \text{, if } |I(\Pi) + \Delta i_{\text{sn}} - I(\Pi')| > 1, \end{cases} \tag{4.7}$$

This allows to align the zero-cost transition of the SGM path aggregation to the surface orientation of the scene (cf. Figure 4.4, Column 3). The pixel-wise discrete index jumps can be computed once for each pixel p and each path direction r, as also noted by Scharstein et al. (2017), providing little computational overhead. (Ruf et al. 2021a, Sec. 2.3.2.)

Penalizing Deviations from the Gradient of the Minimum Cost Path:

Instead of relying on additional information, e.g. normal vectors, the third extension computes the running gradient $\nabla \mathrm{r}$ in scene-space, which corresponds to the minimal path costs, in order to adjust the zero-cost transition in the aggregation of the path costs for non-fronto-parallel surface orientations. Hence, it is denoted as *path gradient* SGM ($\text{SGM}^{\Pi\text{-pg}}$). (Ruf et al. 2021a, Sec. 2.3.2.)

The gradient vector $\nabla \mathrm{r} = \mathrm{M} - \mathrm{M}'$ in scene-space is dynamically computed while traversing along the path r. In this, M again denotes the scene point that is found by intersecting the viewing ray through p with Π, while M' denotes the scene point, which is parameterized by p' and the plane $\hat{\Pi}'$. Here, $\mathrm{p}' = \mathrm{p} + \mathrm{r}$ represents the predecessor of p along the path r and $\hat{\Pi}'$ denotes the plane at distance $\hat{\delta} = \arg\min_{\delta} L_{\mathrm{r}}(\mathrm{p}', \Pi)$ associated with the previous minimal path costs. (Ruf et al. 2021a, Sec. 2.3.2.)

From this, a discrete index jump Δi_{pg} is computed, which is again used to account for possibly slanted surfaces in scene-space by adjusting the zero-cost transition of the smoothness term according to

$$V_{\Pi\text{-pg}}(\Pi,\Pi') = \begin{cases} 0 & , \text{if } I(\Pi) + \Delta i_{\mathrm{pg}} = I(\Pi') \\ \varphi_1 & , \text{if } \left|I(\Pi) + \Delta i_{\mathrm{pg}} - I(\Pi')\right| = 1 \\ \varphi_2 & , \text{if } \left|I(\Pi) + \Delta i_{\mathrm{pg}} - I(\Pi')\right| > 1, \end{cases} \tag{4.8}$$

This implicitly penalizes deviations from the running gradient between two scene points corresponding to two consecutive pixels on the aggregation path r (cf. Figure 4.4, Column 4). (Ruf et al. 2021a, Sec. 2.3.2.)

4.3.3.2 Adaptive Smoothness Penalties Within Semi-Global Matching

In the original publication (Hirschmüller 2005, Hirschmüller 2008) of the SGM algorithm, the author suggests to use an adaptive adjustment of the second penalty φ_2 according to the image gradient along path r. This should enforce a preservation of depth discontinuities at object boundaries. In this work, the adaptive adjustment of φ_2 is based on the absolute intensity difference ($\Delta \mathcal{I}_{\mathrm{pq}}$) between two neighboring pixels p and q. It is formulated by

$$\varphi_2 = \varphi_1 \cdot \left(1 + \alpha \cdot \exp\left(-\frac{\Delta \mathcal{I}_{\mathrm{pq}}}{\beta}\right)\right) \tag{4.9}$$

with $\Delta \mathcal{I}_{\mathrm{pq}} = |\mathcal{I}(\mathrm{p}) - \mathcal{I}(\mathrm{q})|$. Moreover, $\alpha = 8$ and $\beta = 10$ are set according to Scharstein et al. (2017). Since FaSS-MVS uses multiple matching images, the SGM penalties are multiplied with the number of input images inside the left and right subsets with respect to $\mathcal{I}_{\mathrm{ref}}$, since the matching costs are summed up within these image sets. (Ruf et al. 2021a, Sec. 2.3.3.)

4.3.3.3 Depth Refinement

In a final optimization step in the process of depth map estimation, the pixel-wise estimates are refined to lie in between the actual sampling steps of the plane-sweep sampling. Here, the simple and yet effective approach to fit a parabola through the WTA solution and its two neighbors is employed. This is done analogously to implementing a disparity refinement based on curve fitting for the stereo normal case (cf. Section 2.3.1). But since the sampling points are not equidistantly spaced between each other, the depth differences between the WTA solution and its two neighbors needs to be considered in the optimization and the finding of the curves' minimum. (Ruf et al. 2021a, Sec. 2.3.4.)

4.3.4 Extraction of Surface Normals from Depth Maps

From the estimated depth map $\mathcal{D}$, a normal map $\mathcal{N}$ is computed, holding the local surface orientations in the form of three-dimensional normal vectors. The surface normal vectors N_{p} are calculated for each pixel p by reprojecting $\mathcal{D}$ into a three-dimensional point cloud, based on the intrinsic camera parameters and the

corresponding depth estimate. Then, the cross-product $N_p = H_p \times V_p$ is computed, with H_p being the difference vector between the scene points of two neighboring pixels to p in horizontal direction, and V_p being the difference vector in vertical direction. (Ruf et al. 2021a, Sec. 2.4.)

Solely using the cross-product to compute the surface orientation does not incorporate any local smoothness assumption, which is why an a-posteriori smoothing is applied to the normal map. In this, an appearance-based weighted Gaussian smoothing in a local two-dimensional window $\mathcal{W}_p$ around p is employed, which adjusts the smoothing strength depending on the intensity difference between $q \in \mathcal{W}_p$ and p:

$$\mathcal{N}(p) = \frac{\bar{N}_p}{|\bar{N}_p|}, \tag{4.10}$$

with

$$\bar{N}_p = N_p + \sum_{q \in \mathcal{W}_p} N_q \cdot \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left(-\frac{(q-p)^2}{2\sigma^2} - \frac{\Delta\mathcal{I}_{pq}}{\beta}\right). \tag{4.11}$$

Similar as in Equation (4.9), β is set to 10, while σ is fixed to the radius of $\mathcal{W}_p$. (Ruf et al. 2021a, Sec. 2.4.)

4.3.5 Estimation of Confidence Measures Based on Surface Orientation

Apart from the depth map $\mathcal{D}$ and the normal map $\mathcal{N}$, FaSS-MVS also computes confidence measures with respect to the depth estimates in the range of $[0, 1]$ and stores them inside a confidence map $\mathcal{C}$. Such confidence measures allow a subsequent reasoning on the certainty of the corresponding estimates and, in turn, improve further processing. Thus, confidence maps are helpful byproducts for subsequent steps, such as depth map fusion or scene interpretation. Furthermore, they allow to gain more insight on the effects of different configurations of the presented approach, based on receiver operating characteristic (ROC) curve analysis (cf. Figure 4.8). The computation of the pixel-wise confidence measures in the scope of this work relies on the geometric characteristic of the estimated depth map and is deduced from the normal vectors stored inside the normal map $\mathcal{N}$ and the plane orientations of the plane-sweep sampling. (Ruf et al. 2021a, Sec. 2.5.)

In particular, the geometric confidence measure is based on the enclosed angles between the local surface orientation stored inside the normal map $N_p = \mathcal{N}(p)$, the orientation of the sampling plane N_Π and the reverted viewing direction V. This is adopted from the geometric weighting factor proposed by Kolev et al. (2014). They argue that a depth estimate is more accurate if the surface orientation of the observed geometry is fronto-parallel to the image plane of the camera, and less accurate if the camera is observing slanted surfaces. This correlation is modeled by the scalar product between the surface orientation and the reverted viewing direction. Furthermore, since the image warping, as part of the image matching, can be aligned to the surface orientation by adjusting the normal vector of the plane-sweep algorithm, the plane orientation is also considered. Thus, the geometry-based weighting factor is computed according to:

$$\mathcal{C}(p) = \begin{cases} \frac{\langle N_p, N_\Pi \rangle \langle N_\Pi, V \rangle - \cos\rho}{1 - \cos\rho} & , \text{if } \{\sphericalangle(N_p, N_\Pi) \wedge \sphericalangle(N_\Pi, V)\} \leq \rho \\ 0 & , \text{otherwise .} \end{cases} \tag{4.12}$$

All of the above vectors are assumed to be normalized and given with respect to the local coordinate system of the camera, thus $V = (0\ 0\ {-1})^\intercal$. Just as in the work of Kolev et al. (2014), a critical angle $\rho = 60°$ is used to mark the measurements, for which the enclosed angles exceed this threshold, as unreliable. The additional

consideration of N_Π in Equation (4.12) implicitly models the indirect matching of the input images via the plane-induced homography. (Ruf et al. 2021a, Sec. 2.5.)

4.3.6 Post-Processing and Depth Map Filtering

In a final post-processing step, remaining outliers are removed from the depth, normal and confidence maps by applying a difference-of-Gaussian (DOG) filtering and a geometric consistency check, as described in Section 2.3.2.2 and Section 2.3.3.2, respectively. Here, for all pixels that are invalidated in the depth map, the corresponding pixels inside the normal and confidence maps are also invalidated. (Ruf et al. 2021a, Sec. 2.6.)

In the scope of the DOG filter (cf. Algorithm 2.2), the kernel size of the Gaussian filter kernel and the dilation is set to 7×7 pixels and 3×3 pixels, respectively. Furthermore, the respective speckle filter sizes are set to $\alpha = 7$ and $\beta = 21$. For the multi-view geometric consistency check (cf. Algorithm 2.3), the size of the sliding window is set to $|\Psi| = 5$, the threshold for the reprojection error to $\eta_\mathrm{r} = 10$ and the consistency threshold to $\eta_\mathrm{h} = 3$.

4.4 Experiments

The following sections present the results of experiments conducted in the scope of this work. In these, the effects of different configurations of the presented approach are evaluated and analyzed with respect to individual aspects, such as accuracy, efficiency and application-specific usability. First, the datasets and evaluation metrics used to investigate the potential of FaSS-MVS are introduced in Section 4.4.1 and Section 4.4.2, respectively. Then, the need for a hierarchical processing is evaluated and the optimal number of input images, i.e. the optimal size of the input bundle, is empirically found in Section 4.4.3. This is followed by a short comparison in Section 4.4.4, where the focus is set on the two similarity-metrics and cost functions that are implemented in the scope of this work. In Section 4.4.5, the ability of the three SGM extensions to reconstruct non-fronto-parallel surface structures is evaluated and compared to the effects of using a non-fronto-parallel plane orientation within plane-sweep sampling. An evaluation of the improvements gained by post-filtering is presented in Section 4.4.6, before comparing the results of the best configurations to those achieved by approaches for offline MVS, e.g. COLMAP (Schönberger et al. 2016), in Section 4.4.7. Finally, before discussing the presented results, the outcomes of use-case-specific experiments are shown and qualitatively illustrated in Section 4.4.8. (Ruf et al. 2021a, Sec. 3.)

The complete processing pipeline of the presented approach, except the generation of the Gaussian image pyramids and the parameterization of the plane-sweep algorithm, is implemented in CUDA and is thus optimized for massively parallel computing by GPGPU, which, in turn, is embedded in a C++ application. All experiments, and thus all timing measurements, were conducted using a NVIDIA Titan X GPU and an Intel XEON CPU E5-2650 running with 2.20 GHz. Even though the CPU is designed for a server architecture, only a small part of FaSS-MVS is run on the CPU, and thus its superiority over commodity desktop hardware is insignificant. (Ruf et al. 2021a, Sec. 3.)

4.4.1 Evaluation Datasets

FaSS-MVS is quantitatively evaluated on two public datasets, which also provide an appropriate ground truth, namely the DTU Robot MVS dataset (Jensen et al. 2014, Aanæs et al. 2016) and the dataset from the 3DOMcity Benchmark (Oezdemir et al. 2019). In order to provide appropriate data for a quantitative evaluation, these

Figure 4.5: Overview of the four datasets used for performance evaluation of FaSS-MVS. **Column 1:** Two building models from the DTU Robot MVS dataset. **Column 2:** Example images in oblique and nadir view from the 3DOMcity Benchmark dataset. **Column 3:** Excerpt of the privately acquired TMB dataset. **Column 4:** Use-case-specific dataset acquired during an exercise of the local fire brigade. (Ruf et al. 2021a, Fig. 5.)

two datasets rely on images of scale-modeled buildings and an urban scenery from which an accurate ground truth is acquired. For a qualitative evaluation and a discussion regarding the usability of FaSS-MVS for online dense image matching and 3D reconstruction, two privately captured datasets of real-world scenes are used, henceforth referred to as the TMB and the FB dataset. In the following, the characteristics of these datasets are briefly introduced, and information on which portions of the data sets are used and what kind of ground truth is available for the evaluation are discussed. (Ruf et al. 2021a, Sec. 3.1.)

DTU Robot MVS Dataset:

The *DTU Robot MVS dataset* (Figure 4.5, Column 1) is comprised of 124 different tabletop scenes. For each scene, input images are provided, which were captured under eight different lighting conditions from 49 locations. These locations are the same for all scenes, since the camera is mounted on an industrial robot arm, which could repeatedly be moved to the set camera poses. Furthermore, for each scene, a ground truth is provided in the form of a detailed point cloud captured by a structured light scanner. (Ruf et al. 2021a, Sec. 3.1.)

For the quantitative evaluation of the performance of the presented approach, 21 scans of different building models were selected, since these scenes are closest to the targeted use-case, namely the online reconstruction of urban scenes and man-made structures. In this, the already undistorted images with an image resolution of 1600×1200 pixels, together with the corresponding intrinsic and extrinsic camera data, were used as input data to the approach. The benchmark provides an evaluation routine and a corresponding score table for the assessment of a reconstructed point cloud with respect to the ground truth captured by the structured light scanner. However, the focus of this work lies in the estimation of depth and normal maps only, without the subsequent fusion into a three-dimensional point cloud. Thus, in order to evaluate the accuracy of the estimated depth and normal maps, corresponding ground truth data is extracted from the structured light scans by rendering them from the viewpoints of the corresponding input images, given the intrinsic and extrinsic camera data. (Ruf et al. 2021a, Sec. 3.1.)

3DOMcity Benchmark Dataset:

Within the DTU benchmark, the camera is moved along a circular trajectory, resembling an orbital flight of a UAV, with the camera focusing on the object of interest. This kind of camera movement or flight trajectory is typical for the case when an object is to be fully reconstructed with a maximum precision (Wenzel et al. 2013a). However, depending on the aircraft and the surrounding constraints, such a flight is not always feasible or desired. Typical for the mapping of a larger area, a grid flight is performed, where the aircraft is flying linearly over the area of interest, with the camera orientation being fixed with respect to the sensor carrier. (Ruf et al. 2021a, Sec. 3.1.)

Such a configuration is simulated by the data provided as part of the 3*DOMcity Benchmark* (Oezdemir et al. 2019) (Figure 4.5, Column 2). In this, images of a scale-modeled urban scene, comprised of differently sized and shaped buildings, as well as streets and vegetation, are captured with a DSLR camera which is moved along a rigid bar in parallel lines over the model. The distorted and undistorted images are provided with a maximum resolution of 6016×4016 pixels from oblique and nadir vantage points, with a forward and sideways overlap of 85/75 % in case of the oblique views and 80/65 % between the nadir views. While the benchmark internally uses a point cloud of two exemplary building models, which is captured by a multi-stripe triangulation-based laser scanner in order to assess the accuracy of submitted DIM point clouds, it publicly only provides a reference point cloud for the task of classification, which is computed by a semi-global DIM algorithm. (Ruf et al. 2021a, Sec. 3.1.)

To use the data of the 3DOMcity Benchmark for the quantitative evaluation of the performance of the presented approach, the already undistorted images are first downscaled to a size of 1798×1200 pixels, preserving the initial aspect ratio, before estimating the intrinsic camera parameters with the help of COLMAP (Schönberger and Frahm 2016). The extrinsic camera data is extracted from the reference that is provided as part of the benchmark. For the accuracy assessment of the depth maps, the reference data is computed by rendering the reference DIM point cloud of the whole model, provided for the task of point cloud classification, from the viewpoints of the input images, just as in the case of the dataset of the DTU benchmark. (Ruf et al. 2021a, Sec. 3.1.)

Real-World Use-Case-Specific TMB and FB Dataset:

The strength and aim of the datasets from the DTU and 3DOMcity benchmarks are their small scale and the associated ability to record or compute accurate reference data, which in turn facilitates a quantitative evaluation of the accuracy of the assessed algorithms. These datasets, however, are recorded under controlled environments and do not fully accommodate for the use-case aimed at with FaSS-MVS, namely the online MVS from monocular video data captured by COTS UAVs, flying at altitudes below 100 m. In order to account for the named use-case and perform a qualitative evaluation on real-world data, appropriate test data was captured and collected into a private dataset. (Ruf et al. 2021a, Sec. 3.1.)

This dataset is two-fold. The first part, namely the *TMB dataset* (Figure 4.5, Column 3), consists of four sequences captured by a DJI Phantom 3 professional, flying around a free-standing house and containers at altitudes between 8 m to 15 m. For the second part, which is denoted as the *fire brigade (FB) dataset* (Figure 4.5, Column 4), images were acquired during a fire brigade exercise around a big industrial building. The data was captured using a DJI Matrice 200 with a Zenmuse XT2 sensor flying linearly over the area on which the exercise was performed. For all sequences, the images were captured at a frame rate of about 1 FPS and downsampled to an image size of 1920×1080 pixels. However, due to the varying velocities, the distances

between the frames are not always the same and thus images that are not appropriate as input to the presented approach, e.g. by providing too little offset, are discarded. As reference data, detailed point clouds of each sequence were computed by means of structure-from-motion (SFM) and MVS using COLMAP (Schönberger and Frahm 2016, Schönberger et al. 2016). Given the GNSS metadata provided by the UAVs for each input image, the reference data was transformed into a local east-north-up (ENU) frame in order to have a metric reference. The undistorted images as well as the intrinsic and extrinsic camera data produced by COLMAP serve as input to FaSS-MVS for the evaluation with respect to the TMB and FB dataset. (Ruf et al. 2021a, Sec. 3.1.)

4.4.2 Error Measures

For a direct quantification of the error between the estimated depth map $\mathcal{D}_{\text{est}}$ and the corresponding ground truth $\mathcal{D}_{\text{gt}}$, absolute and relative L1 measures are used:

$$\text{L1-abs}(\mathcal{D}_{\text{est}}, \mathcal{D}_{\text{gt}}) = \frac{1}{|\mathcal{V}|} \sum_{\text{p} \in \mathcal{V}} |\mathcal{D}_{\text{est}}(\text{p}) - \mathcal{D}_{\text{gt}}(\text{p})| \ , \text{ and} \tag{4.13}$$

$$\text{L1-rel}(\mathcal{D}_{\text{est}}, \mathcal{D}_{\text{gt}}) = \frac{1}{|\mathcal{V}|} \sum_{\text{p} \in \mathcal{V}} \frac{|\mathcal{D}_{\text{est}}(\text{p}) - \mathcal{D}_{\text{gt}}(\text{p})|}{\mathcal{D}_{\text{gt}}(\text{p})} \ . \tag{4.14}$$

In this, $\mathcal{V}$ denotes the set of pixels for which both $\mathcal{D}_{\text{est}}$ and $\mathcal{D}_{\text{gt}}$ have valid depth measurements. While L1-abs provides an absolute and, in turn, interpretable insight on the mean error of the estimated depth map, it is rather unsuitable for comparing the results across multiple datasets with different depth ranges. This is because the error of depth measurements typically increases with increasing depth, which leads to a higher absolute error for datasets with a larger scene depth. In order to compensate for this effect, the relative L1-rel measure normalizes the absolute difference by the depth stored at the corresponding ground truth pixel. This reduces the effect that erroneous pixels in distant areas of the scene have on the error score, while at the same time increasing the weight of the pixels that are close to the camera. (Ruf et al. 2021a, Sec. 3.2.)

The two error measures introduced above provide a simple strategy to evaluate the error of the estimates. However, they do not allow to reason about the completeness and density of the estimated depth map. Thus, since the focus of this work is on dense MVS, it is also of great interest to know how many pixels of $\mathcal{D}_{\text{est}}$ are actually filled with correct estimates. To do so, two tightly coupled error scores, namely the accuracy (Acc_θ) and completeness (Cpl_θ), are also used. These scores are typically used to evaluate classification tasks, but have also been used to evaluate range measurements in recent years (Schöps et al. 2017, Knapitsch et al. 2017). On the one hand, the accuracy Acc_θ indicates the amount of pixels within the estimated depth map $\mathcal{D}_{\text{est}}$, for which the corresponding depth value is within a given threshold θ to the ground truth:

$$\text{Acc}_\theta(\mathcal{D}_{\text{est}}, \mathcal{D}_{\text{gt}}) = \frac{1}{|\mathcal{E}|} \sum_{\text{p} \in \mathcal{V}} \left[\max \left\{ \frac{\mathcal{D}_{\text{est}}(\text{p})}{\mathcal{D}_{\text{gt}}(\text{p})}, \frac{\mathcal{D}_{\text{gt}}(\text{p})}{\mathcal{D}_{\text{est}}(\text{p})} \right\} < \theta \right] . \tag{4.15}$$

The completeness Cpl_θ, on the other hand, indicates the fraction of the ground truth pixels, for which estimates exist, which are within the given distance threshold to the reference:

$$\text{Cpl}_\theta(\mathcal{D}_{\text{est}}, \mathcal{D}_{\text{gt}}) = \frac{1}{|\mathcal{G}|} \sum_{\text{p} \in \mathcal{V}} \left[\max \left\{ \frac{\mathcal{D}_{\text{est}}(\text{p})}{\mathcal{D}_{\text{gt}}(\text{p})}, \frac{\mathcal{D}_{\text{gt}}(\text{p})}{\mathcal{D}_{\text{est}}(\text{p})} \right\} < \theta \right] . \tag{4.16}$$

Again, $\mathcal{V}$ holds the set of pixels for which both $\mathcal{D}_{\text{est}}$ and $\mathcal{D}_{\text{gt}}$ have valid depth measurements. Similarly, $\mathcal{E}$ denotes the pixel set with valid estimates, while $\mathcal{G}$ holds the pixels with valid ground truth values. In both Equation (4.15) and Equation (4.16), the operator $[\cdot]$ refers to the Iverson bracket. The threshold θ is given as the percentage of the corresponding ground truth value. For example, $\text{Acc}_{1.25}$ and $\text{Cpl}_{1.25}$ hold the fraction of pixels with respect to the $\mathcal{D}_{\text{est}}$ and $\mathcal{D}_{\text{gt}}$, for which the difference between the estimate and ground truth is smaller than 25 % of the corresponding ground truth depth. These two measures can further be summarized into a combined score, namely the F_θ-score, which is the harmonic mean between the Acc_θ and Cpl_θ:

$$\text{F}_\theta(\mathcal{D}_{\text{est}}, \mathcal{D}_{\text{gt}}) = 2 \cdot \frac{\text{Acc}_\theta \cdot \text{Cpl}_\theta}{\text{Acc}_\theta + \text{Cpl}_\theta} \,. \tag{4.17}$$

Thus, a high F_θ-score indicates a good trade-off between the achieved accuracy of the depth map and its completeness with respect to the ground truth. (Ruf et al. 2021a, Sec. 3.2.)

4.4.3 Need for Hierarchical Processing and Finding the Optimal Number of Input Images

In the first experiment, the need for and importance of the hierarchical processing scheme within FaSS-MVS as well as the best configuration on the size of the input bundle are evaluated. Here, a couple of aspects are considered in order to find the best configuration for the succeeding experiments. The objective is to find the appropriate number n of Gaussian pyramid levels and the size $|\Omega|$ of the input bundle, providing a good trade-off between

- the error of the resulting depth maps, measured by L1-abs and L1-rel,
- the sampling density of the scene and the entailed resources needed for the computation,
- as well as the resulting processing run-time.

For this experiment, a fronto-parallel plane orientation is used as part of the plane-sweep image matching and the NCC with a support region of 5×5 pixels is set as similarity measure. The optimization of the cost volume and the extraction of the optimal depth map is done by employing the SGM^{Π} scheme, which is the adoption of the standard SGM optimization to the use of plane-sweep image matching (cf. Section 4.3.3.1). The smoothness penalty within the SGM optimization is set to $\varphi_1 = 100$, while the adaptive φ_2 penalty as described in Section 4.3.3.2 is used. This, together with the 5×5 sized NCC as matching cost, was chosen in accordance with the work of Scharstein et al. (2017). To find the appropriate height of the Gaussian pyramids, the size of the input bundle, i.e. the number of input images, is first set to $|\Omega| = 3$. (Ruf et al. 2021a, Sec. 3.3.)

Table 4.1: Mean errors achieved on the DTU and 3DOMcity dataset for different number of Gaussian pyramid levels (n) as part of the hierarchical processing scheme. The error metrics used are the absolute L1-abs, measured in mm, as well as the relative L1-rel measure. Both are averaged over all evaluated depth maps within each dataset. (Ruf et al. 2021a, Tab. 1.)

		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
DTU	L1-abs	26.394	26.221	<u>23.473</u>	25.045	29.676
	(in mm)	±24.262	±23.835	±19.656	±19.298	±19.436
	L1-rel	0.036	0.036	<u>0.032</u>	0.034	0.041
		±0.032	±0.032	±0.026	±0.026	±0.026
3DOMcity	L1-abs	<u>12.789</u>	14.936	21.801	32.458	47.422
	(in mm)	±6.916	±6.754	±8.010	±9.408	±22.292
	L1-rel	<u>0.010</u>	0.012	0.017	0.026	0.037
		±0.006	±0.006	±0.007	±0.009	±0.014

Table 4.2: Processing run-time measured for different configurations of the pyramid height on the DTU and 3DOMcity dataset. In addition, the maximum number of sampling planes with which the scene was sampled at the highest pyramid level is stated. (Ruf et al. 2021a, Tab. 2.)

		$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
DTU	Run-time	2365	1315	386	220	187
	(in ms)	±15	±10	±2	±2	±1
	max. # planes	256	256	128	64	32
3DOMcity	Run-time	613	431	225	196	192
	(in ms)	±3	±3	±1	±1	±1
	max. # planes	128	64	32	16	8

Table 4.1 lists the mean L1 errors of the estimated depth maps when evaluated with different number of pyramid levels on the datasets of both the DTU and 3DOMcity benchmark. In this, the absolute and relative L1 measures are used, averaged over all depth maps within each dataset. It is to be expected, that the omission of any hierarchical processing, i.e. the use of only one pyramid level and thus no coarse-to-fine processing, would lead to the smallest error between the estimate and ground truth. However, the results reveal that in case of the DTU dataset the smallest mean error, even if it is only slightly smaller, is achieved when setting $n = 3$, while the best result in case of the 3DOMcity dataset is achieved at $n = 1$. (Ruf et al. 2021a, Sec. 3.3.)

As described in Section 4.3.2.3, the plane distances within the plane-sweep sampling, and thus the sampling points, are selected in such a way that two consecutive planes induce a maximum disparity difference of 1 pixel. Depending on the capturing setup, i.e. the relative poses between the images and their obliqueness and, in turn, the range of the scene depth, this can lead to a very high number of sampling points and with it to a large memory consumption, as the dimensions of the three-dimensional cost volume need to be set accordingly. Thus, in order to not exceed the memory limit, the maximum number of sampling points for the highest pyramid level is restricted to 256 in the implementation of the approach. In case of the camera setup of the DTU dataset and the configuration of this experiment, i.e. having a bundle size of $|\Omega| = 3$, a pyramid height of 3 is the smallest height at which the number of sampling points at the highest level does not reach or exceed the set limit, as Table 4.2 shows. Comparing Table 4.1 and Table 4.2 further reveals that on both datasets the best results are achieved, when the computation is initialized at the highest pyramid level with a maximum of 128 sampling planes. (Ruf et al. 2021a, Sec. 3.3.)

Another criterion which is used to deduce the best configuration on the height of the Gaussian pyramid is the run-time needed to estimate a single depth map. Table 4.2 additionally lists the corresponding measurements taken, i.e. the amount of milliseconds it takes to estimate a single depth map given a certain number of pyramid levels, as well as the number of planes used for sampling the scene-space at the highest pyramid level. The measurements again show, that up to $n = 3$ in case of the DTU dataset, the number of sampling planes at the highest pyramid level is equal to the limit of 256 and that with a smaller amount of sampling points the run-time decreased drastically. Furthermore, the significant drop of one second in run-time between using a pyramid height of 2 and 3 suggests that the decreasing use of processing resources on the GPU increases the processing speed and that going from $n = 2$ to $n = 3$ makes a significant improvement in its efficiency. Since the use of a higher number of pyramid levels does not only reduce the amount of sampling points, but also the image size at the highest pyramid level and with it the amount of pixels that need to be matched. Thus, depending on the camera setup, a hierarchical processing is very important in order to ensure a high sampling density of the scene-space, while at the same time efficiently utilizing the processing hardware and, in turn, alleviating high processing speeds. In case of the DTU dataset, this experiment shows that the best number of pyramid levels to be used is $n = 3$, which will thus be set for the successive experiments. In case of the

Table 4.3: Mean errors achieved on the DTU and 3DOMcity dataset for different input bundle sizes ($|\Omega|$), i.e. number of images. In addition, the differences in run-time, with respect to the measurements of the first part (i.e. $|\Omega| = 3$), are stated. (Ruf et al. 2021a, Tab. 3.)

		$\|\Omega\| = 3$	$\|\Omega\| = 5$	$\|\Omega\| = 7$
DTU	L1-abs (in mm)	23.473 ±19.656	19.832 ±16.225	21.843 ±21.605
	L1-rel	0.032 ±0.026	0.027 ±0.021	0.031 ±0.031
	Δ Run-time (in ms)		+271	+302
3DOMcity	L1-abs (in mm)	14.936 ±6.754	14.615 ±6.254	16.514 ±7.569
	L1-rel	0.012 ±0.006	0.012 ±0.007	0.014 ±0.009
	Δ Run-time (in ms)		+360	+410

3DOMcity dataset, Table 4.1 suggests that the best configuration is to use the original image size. A hierarchical processing scheme is needed, however, in order to use $\text{SGM}^{\Pi\text{-sn}}$, the extension of the SGM algorithm to consider local surface orientations in order to account for slanted surfaces. Thus, in case of the 3DOMcity dataset, the successive experiments will be executed with $n = 2$, which induces only a slightly higher mean error compared to the best configuration. (Ruf et al. 2021a, Sec. 3.3.)

In the second part of this experiment, the effects of a different number of input images and, in turn, the optimal size $|\Omega|$ of the input bundle is evaluated. Here, the settings for the plane-sweep image matching and the subsequent SGM optimization are kept the same as before. The height of the Gaussian pyramids is fixed to $n = 3$ in case of the DTU dataset and $n = 2$ in case of the data from 3DOMcity dataset. Table 4.3 lists the mean errors achieved on both datasets with different number of input images, as well as the difference in run-time with respect to the best configuration of the first part of the experiment. The results reveal that the best accuracies are achieved, when five input images are used for image matching, even though, in case of the 3DOMcity dataset, it is only a marginal improvement. As expected, the utilization of more input images in the process of image matching also leads to an increase in run-time, since more pixels are matched. At the same time, however, there is more time available to keep up with the image acquisition as discussed in Section 4.5.3. In conclusion, in the subsequent experiments, the size of the input bundle is set to $|\Omega| = 5$, while the height of the Gaussian image pyramids is set to $n = 3$ and $n = 2$ in case of the DTU and 3DOMcity dataset, respectively. (Ruf et al. 2021a, Sec. 3.3.)

4.4.4 Effects of Different Similarity Measures in the Process of Dense Multi-Image Matching

As part of the plane-sweep multi-image matching, this approach comprises two different similarity measures and cost functions: The Hamming distance of the census transform (CT) (cf. Section 2.1.1) as well as a truncated and scaled form of the normalized cross correlation (NCC) (cf. Section 2.1.2). While the CT is computationally less expensive than the NCC and is thus more suitable for real-time or online processing, it is less discriminative, which might result in a more ambiguous set of matched pixel correspondences. When working with a stereo normal case, in which the input images suffer only from little perspective distortion induced by homographic transformations, the CT outperforms the NCC in both run-time and accuracy (Ruf et al. 2021b). However, as

Table 4.4: Mean errors achieved on the DTU and 3DOMcity dataset when using different similarity measures and cost functions with different support regions. (Ruf et al. 2021a, Tab. 4.)

		$CT_{5\times5}$	$CT_{9\times7}$	$NCC_{5\times5}$	$NCC_{9\times9}$
DTU	L1-abs	42.136	42.305	19.832	19.667
	(in mm)	±37.958	±36.394	±16.225	±16.453
	L1-rel	0.056	0.057	0.027	0.027
		±0.048	±0.046	±0.021	±0.021
3DOMcity	L1-abs	22.128	26.005	14.615	13.789
	(in mm)	±14.218	±14.106	±6.254	±5.962
	L1-rel	0.019	0.022	0.012	0.011
		±0.014	±0.014	±0.007	±0.006

the results in Table 4.4 show, the perspective distortion, resulting from the warping of images from converging cameras by means of the plane-induced homography within the plane-sweep algorithm, leads to a significant increase in error when using the CT as similarity measure. (Ruf et al. 2021a, Sec. 3.4.)

Apart from the two different similarity measures, the effects of different support regions are also evaluated in the scope of this experiment. In this, for each similarity measure, the two most commonly used configurations were tested, with a support region of a size of 5×5 pixels being a good trade-off between uniqueness and computational complexity, while, in case of the CT, a support region of a size of 9×7 pixels is the biggest size for which the bit-string still fits into a single 64-bit integer. The configuration of the plane-sweep algorithm and the SGM optimization is set in accordance with the values from the first experiment (cf. Section 4.4.3). In terms of the SGM penalties, φ_1 is set to 100 for both $NCC_{5\times5}$ and $NCC_{9\times9}$, since the maximum matching cost of the NCC is normalized to 255, independent of the support region. For $CT_{5\times5}$ and $CT_{9\times7}$, however, φ_1 is set to 9 and 24, respectively, which is equivalent to the configuration for NCC, when considering the ratio between φ_1 and the maximum matching cost. (Ruf et al. 2021a, Sec. 3.4.)

Even though the NCC with a support region of 9×9 pixels achieves the best results, $NCC_{5\times5}$ is selected for further experiments, since the rise in error is only little but its computational complexity is significantly less and, in turn, its throughput higher than that of $NCC_{9\times9}$ as illustrated in Section 3.4.1.5. (Ruf et al. 2021a, Sec. 3.4.)

4.4.5 Ability to Reconstruct Non-Fronto-Parallel Surface Structures

As described in Section 4.3.3.1, apart from the straight-forward combination of SGM with the plane-sweep sampling (SGM^{Π}), FaSS-MVS comprises two further extensions of the SGM algorithm that allow to account for non-fronto-parallel surface structures: namely the incorporation of surface normals to adjust the zero-cost transition in the SGM path aggregation ($SGM^{\Pi\text{-sn}}$) and the penalization of deviations from the gradient of the minimum cost path ($SGM^{\Pi\text{-pg}}$). In addition, by selecting an appropriate normal vector, and with it a corresponding sweeping direction, it is also possible to adjust the sampling plane orientations of the plane-sweep sampling to the scene structure. In the following, the results achieved by $SGM^{\Pi\text{-sn}}$ and $SGM^{\Pi\text{-pg}}$, in combination with a fronto-parallel sampling plane orientation, are first evaluated and compared to those achieved by SGM^{Π}, before the effects of different non-fronto-parallel plane orientations are examined. The configuration of the other hyper-parameters is as described and evaluated above: the size of the input bundle is set to $|\Omega| = 5$, the NCC with a support region of 5×5 pixels is used as similarity measure and cost function. The pyramid height is set to $n = 3$ and $n = 2$ for the DTU and 3DOMcity dataset, respectively. (Ruf et al. 2021a, Sec. 3.5.)

The quantitative results displayed in Table 4.5 only reveal minor differences in the L1 error between the different implementations of the SGM optimization. While, in case of the DTU dataset, the best results are achieved by the $SGM^{\Pi\text{-pg}}$ implementation, on the 3DOMcity dataset, the standard adaptation of the SGM optimization to the plane-sweep sampling, i.e. SGM^{Π}, reaches the lowest error. The relative L1 error does not reveal any difference. This is due to the fact that the individual L1-rel scores only start to differ from the forth decimal place on-wards. Nonetheless, the ranking with respect to L1-rel score corresponds to that of the L1-abs score. In a qualitative comparison, Figure 4.6 reveals that $SGM^{\Pi\text{-sn}}$ leads to a seemingly smoother depth and normal map (e.g. on the ground plane), while at the same time, however, loosing small details and amplifying unwanted depth discontinuities in some areas, such as the building facade. When closely comparing the normal maps between SGM^{Π} and $SGM^{\Pi\text{-pg}}$, slightly smaller stair-casing artifacts can be noticed in case of $SGM^{\Pi\text{-pg}}$, which also supports the slightly lower error in Table 4.5. A qualitative comparison between the results on the 3DOMcity dataset in Figure 4.7, however, does not reveal any noticeable differences between the different implementations. The reason for the small L1-abs error achieved by SGM^{Π} on the 3DOMcity dataset is assumed to be due to the fact, that the 3DOMcity dataset also contains a subset of nadir images, in which only a small number of slanted surfaces exist and the fronto-parallel orientation of the sampling planes coincides with most of the scene structure. (Ruf et al. 2021a, Sec. 3.5.)

Table 4.5: Quantitative comparison of the results achieved by FaSS-MVS with different implementation and adaptations of the SGM algorithm in combination with a fronto-parallel sweeping direction in order to also account for non-fronto-parallel surfaces. (Ruf et al. 2021a, Tab. 5.)

		SGM^{Π}	**$SGM^{\Pi\text{-sn}}$**	**$SGM^{\Pi\text{-pg}}$**
DTU	L1-abs	19.832	19.768	<u>19.684</u>
	(in mm)	±16.225	±16.192	±16.154
	L1-rel	0.027	0.027	<u>0.027</u>
		±0.021	±0.021	±0.021
3DOMcity	L1-abs	<u>14.615</u>	14.673	15.074
	(in mm)	±6.254	±6.229	±6.133
	L1-rel	<u>0.012</u>	0.012	0.012
		±0.007	±0.007	±0.006

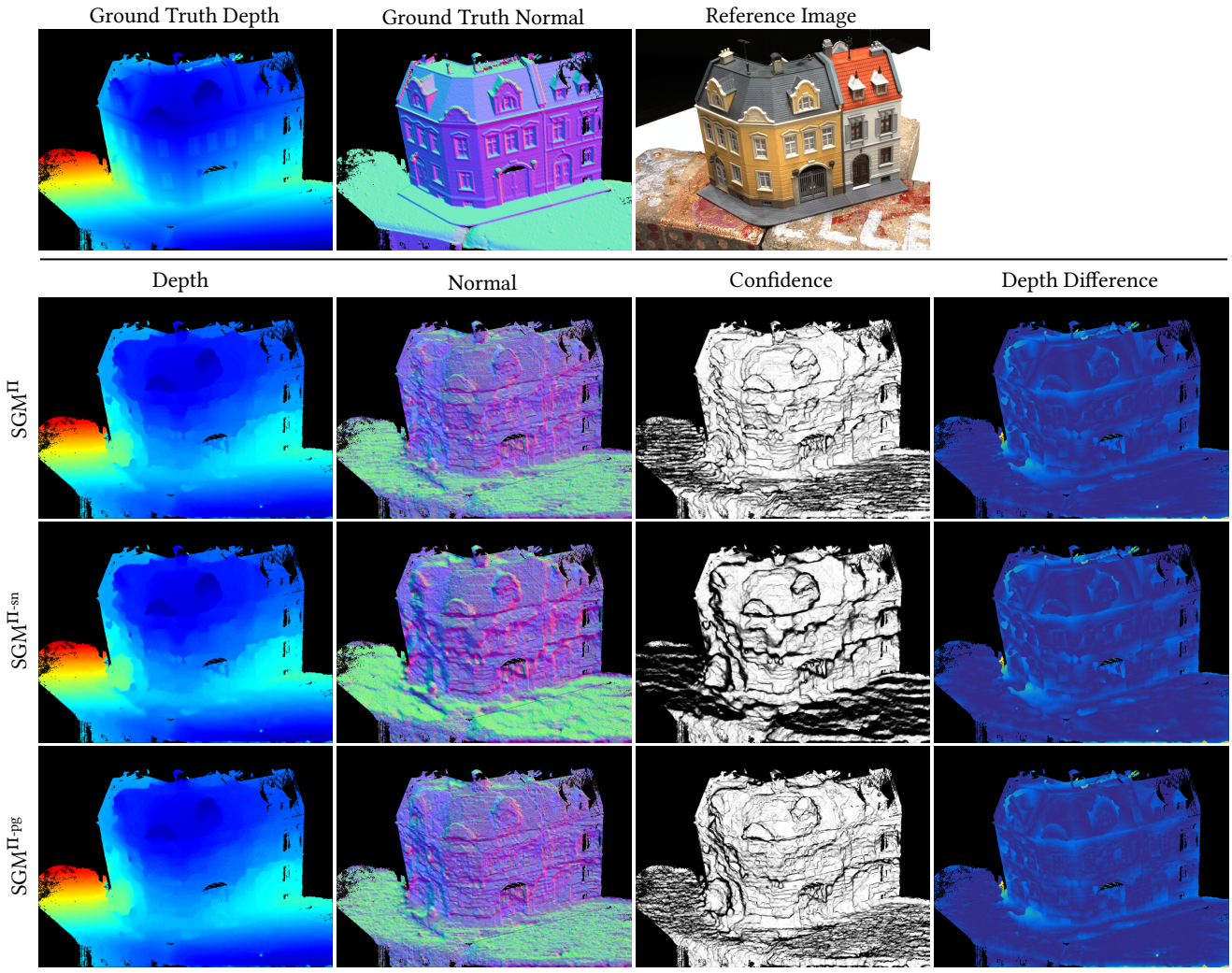


Figure 4.6: Qualitative comparison of the results achieved by the three different SGM implementations of FaSS-MVS on the DTU dataset. **Row 1:** Reference data from the dataset, i.e. the ground truth depth and normal map, as well as the reference image for which the data is computed. **Row 2 - 4:** Data, i.e. depth, normal and confidence maps, computed by SGM^{Π}, $\text{SGM}^{\Pi\text{-sn}}$ and $\text{SGM}^{\Pi\text{-pg}}$, respectively. As well as difference maps holding the pixel-wise absolute difference between the estimated depth map and the ground truth. The color encoding reaches from dark blue (low error) via green to yellow (high error). The depth range within the depth maps reaches from 580 mm (blue) to 830 mm (red). The estimated maps are masked according to the ground truth. (Ruf et al. 2021a, Fig. 6.)

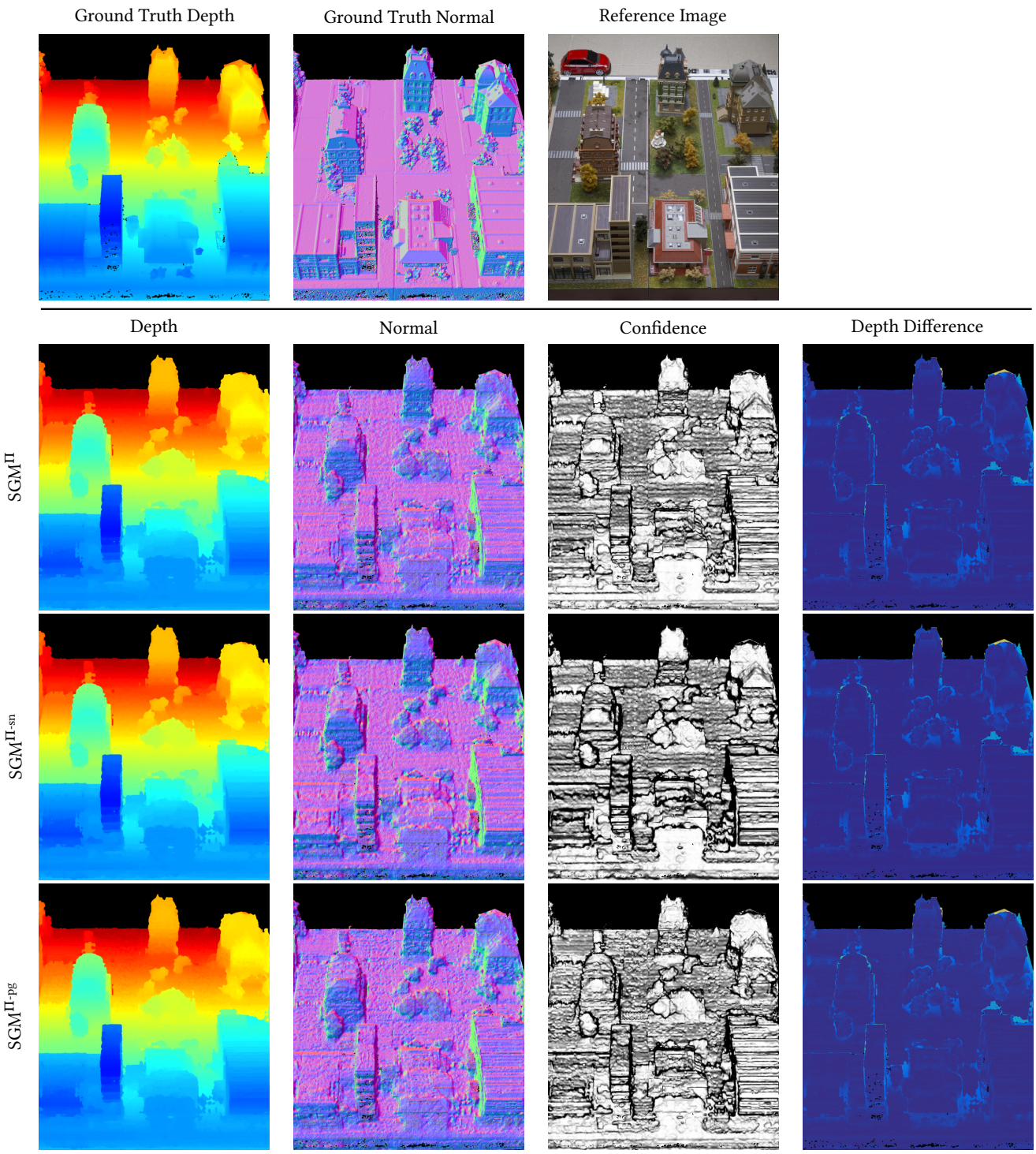


Figure 4.7: Qualitative comparison of the results achieved by the three different SGM implementations of FaSS-MVS on the 3DOMCity dataset. **Row 1:** Reference data from the dataset, i.e. the ground truth depth and normal map, as well as the reference image for which the data is computed. **Row 2 - 4:** Data, i.e. depth, normal and confidence maps, computed by SGM^{Π}, $SGM^{\Pi\text{-sn}}$ and $SGM^{\Pi\text{-pg}}$, respectively. As well as difference maps holding the pixel-wise absolute difference between the estimated depth map and the ground truth. The depth range within the depth maps reaches from 1 m (blue) to 1.8 m (red). The color encoding of the difference maps reaches from dark blue (low error) via green to yellow (high error). The estimated maps are masked according to the ground truth. For visualization in this figure, the resulting images have been rotated counterclockwise by 90°. Thus, the color encoding of the normal maps differs from that used in the other figures. Here, red represents an upwards orientation, while green represents an orientation to the left. (Ruf et al. 2021a, Fig. 7.)

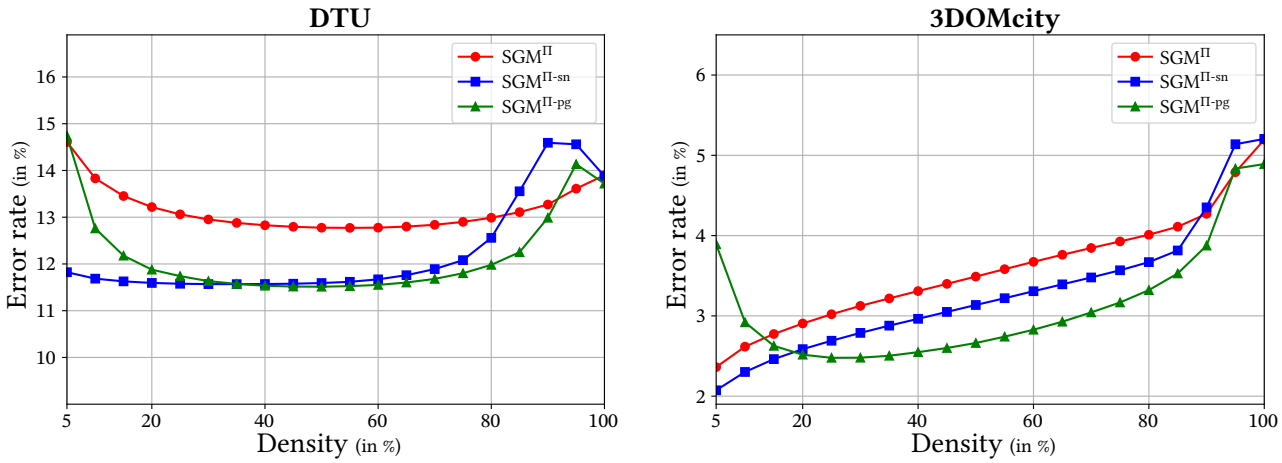


Figure 4.8: ROC curves illustrating the error rate achieved by the three different SGM implementations of FaSS-MVS as a function of increasing density of the estimated depth map. (Ruf et al. 2021a, Fig. 8.)

To further quantify the strength and weaknesses of the three different SGM aggregation strategies, three ROC curves, one for each extension, are plotted for each dataset in Figure 4.8. In this, the curves illustrate the error rate achieved by the corresponding SGM extension as a function of increasing density of the estimated depth map. The density of the depth map is varied by sampling the number of pixels in steps of 5 % based on their ordered confidence stored in $\mathcal{C}$, going from a high to a low confident estimate. The mean error rate is quantified by 1-$\text{Acc}_{1.05}$ (cf. Equation (4.15)) and states the number of sampled pixels in $\mathcal{D}$ whose absolute difference to the ground truth exceeds 5 % of the ground truth value. Thus, at a low density of $\mathcal{D}$, i.e. a high confidence threshold, the error rate should ideally be at its minimum and then increase with increasing density, reaching the overall error of $\mathcal{D}$ at a density of 100 %. The plots start off at a density of 5 %, since the error rate at a density of 0 % is undefined. The curves in Figure 4.8 further support the superiority of the surface-aware SGM extensions over the standard SGM adaptation to plane-sweep sampling, as for both datasets the curves of $\text{SGM}^{\Pi\text{-sn}}$ and $\text{SGM}^{\Pi\text{-pg}}$ are below that of SGM^{Π}, illustrating smaller error rates. The fact that most of the ROC curves start of on a high error rate at a density of 5 % and then drop down before increasing again, suggests that the estimated confidence values do not represent the certainty of the depth estimates appropriately. The reasons for this are manifold and this is further discussed in Section 4.5.4. (Ruf et al. 2021a, Sec. 3.5.)

Table 4.6: Run-time and error measurements of the three SGM extensions of FaSS-MVS with 8 and 4 aggregation paths each, conducted on the DTU dataset with an image size of 1600 × 1200 pixels. (Ruf et al. 2021a, Tab. 6.)

		SGM^{Π}	**$\text{SGM}^{\Pi\text{-sn}}$**	**$\text{SGM}^{\Pi\text{-pg}}$**
8-Path SGM	L1-abs (in mm)	19.832 ±16.225	19.768 ±16.192	19.684 ±16.154
	L1-rel	0.027 ±0.021	0.027 ±0.021	0.027 ±0.021
	Run-time (in ms)	640 ±3	895 ±2	2079 ±3
4-Path SGM	L1-abs (in mm)	21.072 ±16.634	21.091 ±16.632	20.908 ±16.599
	L1-rel	0.029 ±0.022	0.029 ±0.022	0.029 ±0.022
	Run-time (in ms)	413 ±2	546 ±2	1132 ±3

Since FaSS-MVS aims for incremental and online processing, meaning that the computation should ideally keep up with the input stream, the run-time of each SGM extension is also of interest. In Table 4.6, the run-time and error of the complete approach, with above-mentioned parameterization and with respect to each of the three SGM implementations is listed. The measurements were conducted on the dataset of the DTU benchmark. In addition to the standard use of 8 aggregation paths, which achieves the lowest error, the run-time and error when using only 4 aggregation paths is also listed. In the latter case, the diagonal aggregation paths are omitted within the SGM aggregation. This is motivated by the fact that a number of studies (Banz et al. 2010, Hernandez-Juarez et al. 2016) show that a reduction in the number of aggregation paths from 8 to 4 greatly decreases the computation time of the SGM aggregation (cf. Section 3.4.1.5), while only marginally increasing its error, which is also supported by the numbers in Table 4.6. Furthermore, the measurements reveal that especially the $\text{SGM}^{\Pi\text{-pg}}$ extension introduced a great computational complexity compared to SGM^{Π} and $\text{SGM}^{\Pi\text{-sn}}$. However, the reduction of the aggregation paths has a great impact on the run-time, reducing it by up to 45 %, but only marginally affecting the error. Whether the listed run-time is sufficient for online processing is further discussed in Section 4.5.3. (Ruf et al. 2021a, Sec. 3.5.)

In the second part of this experiment, the use of non-fronto-parallel plane orientations within the plane-sweep sampling is investigated. Here, an additional horizontal and vertical orientation, both with respect to the reference coordinate system of the scene, were selected and compared to the fronto-parallel sampling direction. For this, the DTU dataset is split into two subsets, one for the horizontal sampling, in which the camera is looking in a more downwards direction, as well as one for the vertical sampling, where the camera pitch is smaller. As reference, for both subsets, results with a fronto-parallel sampling were computed separately. The quantitative results reveal a major increase in error when non-fronto-parallel plane orientations are used for sampling, as also illustrated by the excerpt in Figure 4.9. However, the Figure 4.9 also reveals that in areas where the surface structure coincides with the sampling direction, e.g. the ground plane, the depth map is very smooth and consistent. (Ruf et al. 2021a, Sec. 3.5.)

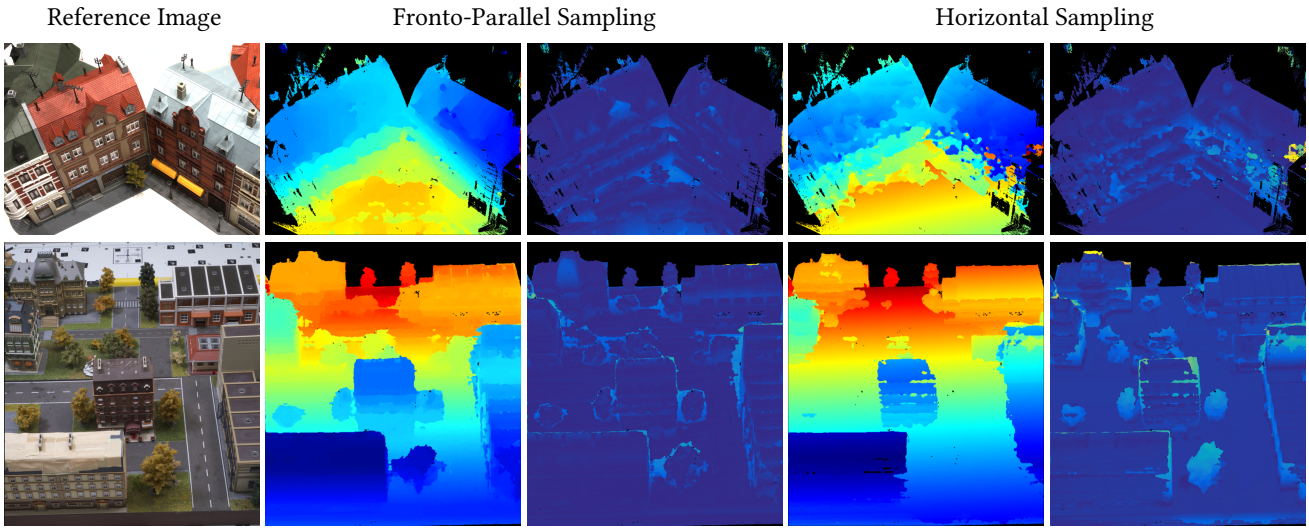


Figure 4.9: Qualitative comparison between the use of a fronto-parallel and non-fronto-parallel sampling direction in combination with SGM^{Π} of FaSS-MVS. **Columns 2 & 4:** Corresponding estimated depth map. **Columns 3 & 5:** Difference map holding the pixel-wise absolute difference between the estimated depth map and the ground truth. The color encoding reaches from dark blue (low error) via green to yellow (high error). The estimated depth maps and the difference maps are masked according to the ground truth. (Ruf et al. 2021a, Fig. 9.)

4.4.6 Improvements Gained by Post-Filtering and Geometric Consistency

In a final ablative experiment, the effects of the implemented post-filtering methods to remove remaining outliers and supposedly wrong estimates by means of difference-of-Gaussian (DOG) filtering (cf. Section 2.3.2.2) and enforcing geometric consistency (cf. Section 2.3.3.2) are studied. While the latter one relies on the actual estimates, the DOG filter is based on the assumption that image regions with low texture might lead to ambiguities in the image matching and, in turn, wrong estimates, masking out corresponding regions. This, however, might result in the false removal of good or even correct estimates. (Ruf et al. 2021a, Sec. 3.6.)

Instead of using the absolute and relative L1 metric to quantitatively assess the results achieved when employing post-filtering, the effects are evaluated using the accuracy Acc_θ (cf. Equation (4.15)) and completeness Cpl_θ measure (cf. Equation (4.16)). This is because they indirectly include information on the density of the resulting depth maps, which should ideally be as high as possible. Since the individual sequences of the 3DOMcity dataset are made up of too little images in order to perform a geometric consistency check with the parameterization mentioned in Section 2.3.3.2, this experiment is only conducted on the dataset of the DTU benchmark. Figure 4.10 depicts the results of different post-filtering strategies, i.e. DOG filtering, geometric consistency filtering, as well as a combination of both, executed in combination with the three different SGM extensions and a fronto-parallel sampling. In addition, the accuracy-completeness curve resulting from the corresponding configurations without post-filtering is also displayed as reference. In the construction of the curves, the threshold θ (i.e. the threshold used to calculate the accuracy and completeness rates) is varied within the list of $1.25, 1.20, 1.15, 1.10, 1.05$ and 1.01. Note that with a decreasing threshold, the accuracy and completeness rates drop. Thus, the highest values are achieved with $\theta = 1.25$. (Ruf et al. 2021a, Sec. 3.6.)

Most evidently, Figure 4.10 again reveals that there is not much difference in the overall error between the three SGM implementations. However, the accuracy-completeness curves clearly show the differences that the individual post-filtering strategies make. Unsurprisingly, the reference configuration where no filtering is employed reaches the highest completeness, as no estimates are removed from the predicted depth map, which, in turn, also leads to the lowest accuracy. The use of a DOG filtering clearly improves this, as it is likely to remove quite a number of wrong estimates originating from low-textured areas. However, as expected, the DOG filter presumably also removes a number of correct estimates, as the use of the filtering based on geometric consistency achieves similar completeness, while at the same time reaching a higher accuracy. Especially when considering the scores for $\theta = 1.01$, i.e. the lower left end of each curve, the use of a geometric consistency filter achieves an increase in completeness of approximately 5 %, while exceeding the accuracy of the use of a DOG filter by more than 10 %. A distinct recommendation on which filter to use, however, cannot be concluded,

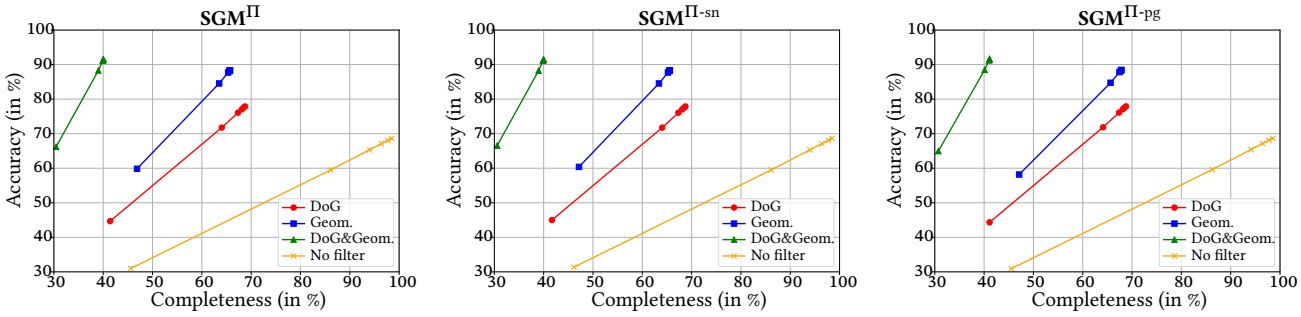


Figure 4.10: Accuracy-completeness curves of different post-filtering strategies within FaSS-MVS, i.e. DOG filtering, geometric consistency filtering, as well as a combination of both, executed in combination with the three different SGM extensions and a fronto-parallel sampling. In the construction, the threshold θ is varied within the list of 1.25, 1.20, 1.15, 1.10, 1.05 and 1.01. With a decreasing threshold, the accuracy and completeness rates drop. (Ruf et al. 2021a, Fig. 10.)

Table 4.7: F-scores achieved by the three SGM extensions of FaSS-MVS in combination with the post-filtering based on geometric consistency. (Ruf et al. 2021a, Tab. 7.)

	$\mathbf{F}_{1.25}$ (in %)	$\mathbf{F}_{1.20}$ (in %)	$\mathbf{F}_{1.15}$ (in %)	$\mathbf{F}_{1.10}$ (in %)	$\mathbf{F}_{1.05}$ (in %)	$\mathbf{F}_{1.01}$ (in %)
SGM^{Π}	74.2	74.1	74.0	73.7	71.5	51.9
$\text{SGM}^{\Pi\text{-sn}}$	74.1	74.1	74.0	73.6	71.5	52.3
$\text{SGM}^{\Pi\text{-pg}}$	75.6	75.5	75.4	75.1	72.9	51.4

since both filtering strategies have their strengths and weaknesses, especially with respect to online processing as discussed in Section 4.5.3. A combination of both filters is not motivated. Even though the accuracy slightly increases, the completeness drops partly by over 20 %. Moreover, this effect can also be achieved by decreasing the threshold of the reprojection error η_r in the geometric consistency check, which probably will increase the accuracy even more. (Ruf et al. 2021a, Sec. 3.6.)

Lastly, to directly compare the different SGM extensions in combination with the geometry-based filtering, which achieves the best results, the corresponding F-scores for each evaluated θ (cf. Equation (4.17)) are listed in Table 4.7. Just as the results displayed in Table 4.5, the F-scores reveal the superiority of $\text{SGM}^{\Pi\text{-pg}}$ over the other two implementations, since for all θ, except one, $\text{SGM}^{\Pi\text{-pg}}$ reaches the highest F-score. (Ruf et al. 2021a, Sec. 3.6.)

4.4.7 Summarized Results and Comparison to Offline Multi-View Stereo Approaches

Finally, before presenting and qualitatively assessing the results of the best configuration on real-world data, a short quantitative summary and comparison to the results produced by offline MVS is done in the following. As offline MVS approach, the widely used and open source COLMAP toolbox (Schönberger et al. 2016) is used. While COLMAP provides the full reconstruction pipeline, i.e. including a subsequent fusion of depth maps into a 3D model, only the geometric depth maps were used in order to make a fair comparison, since the fusion into a 3D model leads to a further filtering of outliers. The significance of a comparison between an online MVS approach, like FaSS-MVS, with an offline approach can be questioned, nonetheless, since the two types of approaches make different assumptions and focus on different aspects within the processing, as further discussed in Section 4.5.1. (Ruf et al. 2021a, Sec. 3.7.)

Based on the previous experiments, the following setup is selected for the final experiments on the DTU dataset. The size of the input bundle is set to $|\Omega| = 5$ and the height of the image pyramids for the hierarchical processing is set to $n = 3$. As sweeping direction and plane normal for the multi-image plane-sweep sampling, a fronto-parallel plane orientation, i.e. $\mathrm{n} = (0\ 0\ {-1})^\intercal$, is used. For the image matching, the normalized cross correlation with a support region of 5×5 pixels is selected and the penalty φ_1 for the subsequent SGM optimization is

Table 4.8: Final quantitative results of the three SGM extensions of FaSS-MVS on the DTU dataset, in combination with fronto-parallel plane-sweep sampling and post-filtering based on geometric consistency check. As reference, the quantitative results achieved by the geometric depth maps of the offline MVS toolbox COLMAP (Schönberger et al. 2016) are given. (Ruf et al. 2021a, Tab. 8.)

	L1-abs (in mm)	**L1-rel**	$\mathbf{F}_{1.25}$ (in %)	$\mathbf{F}_{1.20}$ (in %)	$\mathbf{F}_{1.15}$ (in %)	$\mathbf{F}_{1.10}$ (in %)	$\mathbf{F}_{1.05}$ (in %)	$\mathbf{F}_{1.01}$ (in %)
SGM^{Π}	8.549 ±7.509	0.012 ±0.011	74.2	74.1	74.0	73.7	71.5	51.9
$\text{SGM}^{\Pi\text{-sn}}$	8.479 ±7.559	0.012 ±0.011	74.1	74.1	74.0	73.6	71.5	52.3
$\text{SGM}^{\Pi\text{-pg}}$	8.722 ±7.255	0.013 ±0.010	75.6	75.5	75.4	75.1	72.9	51.4
COLMAP	3.745 ±5.498	0.006 ±0.004	80.2	80.2	80.1	80.0	79.6	74.4

Table 4.9: Run-time comparison between the three SGM extensions with 8 aggregation paths and MVS approach of COLMAP (Schönberger et al. 2016). Measurements were conducted on the dataset of the DTU benchmark and represent the mean run-time required by the different approaches to estimate a single depth map. While COLMAP allows to only estimate photometric depth maps, the quantitative evaluation with respect to the accuracy of COLMAP was done on the geometric depth maps (last column). (Ruf et al. 2021a, Tab. 9.)

	SGM$^{\Pi}$	**SGM$^{\Pi\text{-sn}}$**	**SGM$^{\Pi\text{-pg}}$**	**COLMAP**	**COLMAP**
	8-Path	8-Path	8-Path	Photometric	Photometric+Geometric
Run-time	640	895	2079	14687	34840
(in ms)	±3	±2	±3	±1020	±1243

set to $\varphi_1 = 100$, with φ_2 being adaptively adjusted according to the intensity difference between neighboring pixels as described in Section 4.3.3.2. For post-filtering of the resulting maps, the geometric consistency check based on the reprojection error achieves the best results and is thus selected. Since, the geometric consistency check is not applicable for the dataset of the 3DOMcity benchmark, as the individual image sequences of the dataset are too short, the final comparison is only performed on the dataset provided by the DTU benchmark. (Ruf et al. 2021a, Sec. 3.7.)

Table 4.8 lists the quantitative results of the summarized experiments, together with the results achieved by the geometric depth maps of COLMAP. As to be expected, the results of the offline MVS toolbox COLMAP have the lowest error with respect to all accuracy measures, since it uses a more global optimization scheme without any run-time constraints and, in turn, taking significantly longer to estimate a single depth map as the numbers in Table 4.9 show. Moreover, in contrast to an online approach, offline processing allows to take all input images of the sequence into account and to choose the most appropriate ones for the tasks of DIM and MVS. Nonetheless, the results of FaSS-MVS for online MVS are not too far off, with the error not even being one order of magnitude higher than that of COLMAP. Interestingly, while SGM$^{\Pi\text{-pg}}$ outperforms the other two SGM extensions with respect to the F-score, SGM$^{\Pi\text{-sn}}$ has the lowest L1 error. This can be explained by the density of the resulting depth maps. When using SGM$^{\Pi\text{-pg}}$, more estimates pass the geometric consistency check, resulting in depth maps that are slightly more dense than those produced by SGM$^{\Pi}$ and SGM$^{\Pi\text{-sn}}$ and increasing the F-score, while at the same time also increasing the L1 error. Quantitatively speaking, the difference, however, is only marginal and a conclusion whether one certain SGM extension is to be preferred over the others depends on the use-case and is to be drawn based on qualitative comparisons. (Ruf et al. 2021a, Sec. 3.7.)

In a final comparison, Table 4.10 lists the quantitative results of some representative approaches from literature, including COLMAP, on the evaluation set of the DTU dataset, calculated with the actual evaluation protocol of the benchmark. Here, in contrast to the accuracy (Acc) and completeness (Cpl) used in this work, the Acc and Cpl are calculated based on the absolute difference between the estimates and reference. And instead of using the individual depth maps, the evaluation protocol calculates the error based on the fused point clouds.

Table 4.10: Quantitative results of related work from literature on the DTU benchmark, using the actual evaluation protocol of the benchmark. This protocol calculates the errors not on the individual depth maps, but on the actual point cloud and specifies the accuracy (Acc) and completeness (Cpl) in absolute differences between the estimated points and the reference (lower is better). It is thus somehow comparable to the L1-abs error. (Ruf et al. 2021a, Tab. 10.)

	Acc	**Cpl**	**Overall**
	(in mm)	(in mm)	(in mm)
COLMAP (Schönberger et al. 2016)	0.400	0.664	0.532
Furu (Furukawa and Ponce 2010)	0.613	0.941	0.777
Gipuma (Galliani et al. 2015)	0.283	0.873	0.578
MVSNet (Yao et al. 2018)	0.396	0.527	0.462

Thus, the values are not directly comparable to the ones in Table 4.8. However, in both tables, the results of COLMAP are listed, allowing for tentative comparison. (Ruf et al. 2021a, Sec. 3.7.)

4.4.8 Use-Case-Specific Experiments Conducted on Real-World Datasets

Finally, to demonstrate the performance of FaSS-MVS on use-case-specific and real-world datasets, experiments using SGM$^{\Pi\text{-pg}}$ with 4 aggregation paths and the configuration mentioned in Section 4.4.7 are conducted on the TMB and FB dataset (cf. Section 4.4.1). An excerpt of the computed depth, normal and confidence maps is depicted in Figure 4.11, together with corresponding depth maps estimated by COLMAP as reference. Again, the experiments and timing measurements were conducted on a NVIDIA Titan X. The mean processing time in case of the TMB dataset is 690 ms, partly varying from between 320 ms and 1218 ms depending on the arrangement of the input data and, in turn, the number of plane distances δ at which the scene is sampled. For the FB dataset, the mean processing time is 800 ms, varying between 514 ms and 1419 ms, again depending on the arrangement of the input images. (Ruf et al. 2021a, Sec. 3.8.)

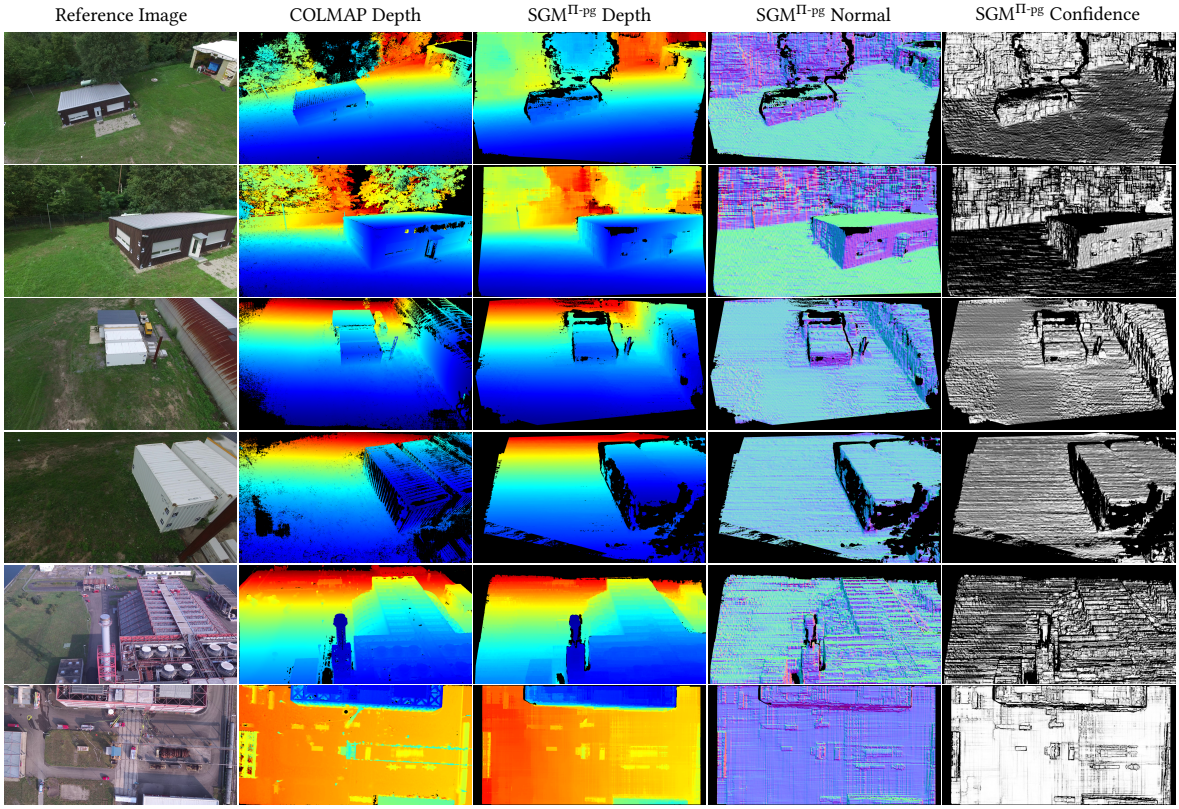


Figure 4.11: Qualitative results of FaSS-MVS with SGM$^{\Pi\text{-pg}}$ and 4 aggregation paths achieved on the two real-world and use-case-specific datasets, namely the TMB dataset and the FB dataset. As comparison, the corresponding depth maps estimated by COLMAP are also visualized. **Rows 1 & 2:** TMB Building scene captured from an altitude of 15 m and 8 m, respectively. **Rows 3 & 4:** TMB Container scene. **Rows 5 & 6:** Two excerpts from the FB dataset. (Ruf et al. 2021a, Fig. 11.)

4.5 Discussion

In the following, the findings of the conducted experiments are discussed with respect to different aspects, namely the overall accuracy and the comparison to offline MVS approaches (Section 4.5.1), the ability of FaSS-MVS to reconstruct slanted surface structures (Section 4.5.2), the run-time and the support for online processing (Section 4.5.3), as well as effects of the employed post-filtering algorithms and the relevance of the confidence estimates (Section 4.5.4). (Ruf et al. 2021a, Sec. 4.)

4.5.1 Overall Accuracy

As the results in Table 4.8 show, the overall accuracies of the depth maps estimated by FaSS-MVS are lower than those of the geometric depth maps from COLMAP, and presumably also lower than the results of other offline MVS approaches as the numbers in Table 4.10 suggest. This is unsurprising, since the procedure and the assumptions involved greatly differ between online and incremental approaches, such as the one presented in this work, and offline approaches. Offline approaches, such as COLMAP, assume that all input images are available at the time of reconstruction, allowing to optimize the set of input images that are considered for the reconstruction of a certain viewpoint. In contrast, online approaches that incrementally perform MVS only consider input images within a temporally confined window, at most all images that were captured up to a certain point in time. Furthermore, offline approaches typically also do not have any time constraints. Nonetheless, the quantitative differences between the results achieved with FaSS-MVS and COLMAP are not that big, less than an order of magnitude, especially when using a geometric consistency filtering in post-processing. Also, a qualitative comparison on use-case-specific input data renders the results of the presented approach very satisfactory. Compared to the geometric depth maps of COLMAP, the depth maps of $\text{SGM}^{\Pi\text{-pg}}$ lack the fine-grained details, such as the roof structures in Rows 5 & 6 of Figure 4.11, which are caused by the coarse-to-fine processing. Bigger structures, however, are well represented and the quality of their reconstruction is comparable, as is the overall density. Yet, even though the fronto-parallel bias of SGM is reduced, some artifacts of the fronto-parallel sampling are still visible, especially in the normal maps of Figure 4.11. (Ruf et al. 2021a, Sec. 4.1.)

Unfortunately, it is hard to make a quantitative comparison against other online approaches, such as the ones presented by Gallup et al. (2007) or Pollefeys et al. (2008), with similar assumption, aims and use-cases. This is mainly due to the lack of appropriate datasets that are publicly available and provide accurate ground truth that in turn allows for a thorough investigation and benchmarking. This holds especially with respect to the task of online and incremental MVS. Current publicly available datasets and benchmarks used to evaluate the performance of algorithms for the task of DIM and MVS aim at the benchmarking of offline approaches, providing a fixed set of input images acquired from predefined viewpoints. However, in case of online approaches, the selection and sampling of images from an input sequence, that are used as input to the approach, have a great effect on the final result. Depending on the configuration of the input images, the range of the scene depth to be sampled can be very large, requiring higher Gaussian pyramids in order to keep the required resources on a reasonable level and not exceed the run-time constraint. However, a higher number of pyramid levels also reduces the initial image size, which in turn will reduce the level-of-detail and the accuracy of the resulting depth map. Thus, a clever assembling of the input bundle is just as important as the correct height of the Gaussian pyramids. (Ruf et al. 2021a, Sec. 4.1.)

4.5.2 Ability to Account for Non-Fronto-Parallel Surfaces

To further increase the accuracy in the reconstruction of slanted, non-fronto-parallel surface structures, this work proposes, apart from SGM^{Π}, two extensions to the SGM algorithm that should reduce the fronto-parallel bias. Namely the incorporation of surface-normals to adjust the zero-cost transition in the SGM path aggregation ($SGM^{\Pi\text{-sn}}$) and the penalization of deviations from the gradient of the minimum cost path ($SGM^{\Pi\text{-pg}}$). The conducted experiments reveal that these extensions only provide a slight quantitative improvement over the standard SGM adaptation (SGM^{Π}) to the plane-sweep sampling. This insight, however, stands in contrast to the experiments conducted by Scharstein et al. (2017). There are at least two reasons that could explain this discrepancy. First, Scharstein et al. (2017) demonstrate their implementation on a two-view stereo dataset, in which the input images are captured by two cameras mounted on a fixed rig and orientated in the same direction. Before being processed, the images are also rectified, i.e. transformed so that both lie on the same image plane and that the epipolar lines coincide with the image rows. Thus, in the process of dense image matching, the images are equidistantly sampled with a step-size of 1 pixel. In case of FaSS-MVS however, the distances of the sampling planes and, in turn, the sampling points are chosen in such a way that the disparity shift along the epipolar line between two consecutive planes is less than or equal to 1. This leads to a sampling with a much higher density, already reducing the stair-casing effect in case of SGM^{Π}. And secondly, Scharstein et al. (2017) propose to use a ground truth normal map for the adjustment of the zero-cost transition, whereas in FaSS-MVS the upscaled normal map of the previous iteration of the hierarchical processing is used. This is bootstrapped with SGM^{Π} on the highest pyramid level, introducing inaccuracies, which probably cannot be fully compensated. The qualitative analysis, however, reveals that $SGM^{\Pi\text{-sn}}$ and $SGM^{\Pi\text{-pg}}$ clearly lead to smoother normal maps and that the stair-casing artifacts in the depth maps are reduced, which is also why in case of the use-case-specific experiments only $SGM^{\Pi\text{-pg}}$ is considered. (Ruf et al. 2021a, Sec. 4.2.)

Apart from reducing the fronto-parallel bias in the SGM path aggregation, the plane-sweep algorithm within FaSS-MVS allows to adjust the image matching to the surface structures in the scene by selecting appropriate normal vectors and sweeping directions. In a short qualitative experiment (cf. Figure 4.9), the effects of a horizontal plane sampling, compared to a fronto-parallel sampling, both in combination with SGM^{Π}, are studied. The results reveal, that the horizontal sampling leads to more consistent depth estimates with little or no stair-casing artifacts in areas where the surface structure coincides with the plane orientation, e.g. the ground plane. In areas, where the surface structures are not horizontal, however, the non-fronto-parallel sampling leads to considerable errors. To overcome this effect, a splitting of the scene into local regions can be considered, which are sampled individually with different plane orientations, similar to the local-plane-sweep approach presented by Sinha et al. (2014). This, however, again comes at the cost of a higher computational complexity. Another remedy is to repeat the plane-sweep image matching multiple times on the whole image domain prior to the SGM optimization, with different sweeping directions and perform a pixel-wise pre-selection of the best plane orientation based on the matching costs, similar to the approach of Pollefeys et al. (2008). This leads to a smaller increase in computational complexity compared to the first option. (Ruf et al. 2021a, Sec. 4.2.)

4.5.3 Run-Time and Online Processing

Given the run-time measurements in Table 4.6 and Table 4.9, the presented approach is obviously not capable of real-time and low-latency processing, in the sense that for each input frame a depth map is computed at similar frame rates as given for the input stream. Considering the nature of the approach and the expected input data, however, the run-time is generally sufficient for online processing, which will be explained in the

following section. The presented approach takes a bundle of three or more input images, with a bundle size of 5 images actually yielding better results, and performs a MVS for one reference image of the input bundle, which is typically the middle one. While these input images could be provided by individual cameras, it is assumed that the images are extracted from an input sequence, which is captured by only one camera that is moving around a static scene. In addition, not every image of the input sequence can be used, since an appropriate baseline needs to be in between each input image in order to allow for the estimation of scene depth. This is obviously dependent on the depth range that is to be sampled and the scene structure. In case of the TMB dataset, the mean distance between the individual input images is 1.8 m and 1.03 m for a flight altitude of 15 m and 8 m, respectively. This increases with higher flight altitudes, due to a larger scene depth. Modern COTS rotor-based UAVs can fly up to a speed of above 10 m/s. The typical flight speed when capturing image data, however, is rather 1-3 m/s (DJI 2020b, DJI 2020a). Thus, if the sets of input images are disjoint, then an estimation needs to be performed at least only every 3 s, considering a low flight altitude, together with a high flight speed of approximately 3 m/s and an input bundle size of 3 images. If a maximum overlap between the input bundles is desired, meaning that the estimation of a new depth map is triggered with every new input frame that is appropriate and that it reuses 4 images from the previous bundle, the required run-time is significantly less. As the use-case-specific experiments for the TMB and FB dataset, however, show, the average processing rate of $\text{SGM}^{\Pi\text{-pg}}$, which is the computationally most complex variant, is between 1-2 Hz, depending on the arrangement of the input images. Another possibility to reduce the run-time is the use of higher Gaussian pyramids, which again comes at the cost of a reduced level-of-detail as already pointed out in the discussion on the overall accuracy (cf. Section 4.5.1). In short, there are a number of possible settings in both the acquisition of the input data, e.g. regarding the flight speed or size and overlap between the input bundles, and the configuration of the presented approach, e.g. regarding the Gaussian pyramid height, depth range or optimization strategy, that allow to tweak the run-time to fit the rate of the input images and, in turn, allow for online processing. All in all, the numbers in Table 4.9 reveal the superiority of the presented approach in terms of run-time compared to state-of-the-art approaches for offline multi-view stereo. (Ruf et al. 2021a, Sec. 4.3.)

The emergence of high-performance SOCs with embedded GPUs, like the NVIDIA Jetson series, allow to bring approaches like FaSS-MVS directly onto the sensor carrier, e.g. the UAV, for on-board processing. To evaluate the feasibility of running FaSS-MVS on board an embedded device, run-time measurements were conducted on the NVIDIA Jetson AGX which is equipped with an 8-core 64-bit ARMv8.2 CPU and a 512-core Volta GPU. Executed on an excerpt of the TMB dataset with an image size of 1920×1080 pixels, FaSS-MVS with SGM^{Π} and parameterized by the final configuration as presented in Section 4.4.7 achieves an average run-time of 727 ms on the Jetson AGX. The average run-time achieved by the same setup on the NVIDIA Titan X is approximately 403 ms. As already discussed, the run-time can further be reduced by increasing the pyramid height to $n = 4$ and $n = 5$, for example, while at the same time accepting a decrease in the quality of the results. This results in an average run-time of 444 ms and 385 ms, respectively. These experiments show, that FaSS-MVS is capable of on-board processing by utilizing a high-performance embedded SOC like the NVIDIA Jetson AGX. This can be of particular interest when considering a deployment on a sensor-carrier that does not suffer from such strong energy constraints as COTS UAVs. (Ruf et al. 2021a, Sec. 4.3.)

4.5.4 Post-Filtering and the Relevance of the Estimated Confidence Values

In the following section, the improvements gained by the post-filtering based on the DOG filter and geometric consistency and the effects the filtering has on the online processing, as well as the relevance and the expressiveness of the confidence estimates are discussed. A comparison of the L1 errors in Table 4.8 to those listed

in Table 4.5, reveals that with the use of post-filtering based on geometric consistency, the mean errors can drastically be reduced by approximately 40 %. The prize for this improvement, however, is a loss in density of the depth map and increase in the latency between the input and the results. The latter one is due to the additional sliding window that is introduced by the geometric-consistency-based filtering. In addition to the bundle of input images, to which only one set of estimates is produced, the geometric filter further requires two or more depth maps for processing. Furthermore, the geometric filter is computationally more complex than the DOG filter. Again, whether to use the DOG or geometric filter depends on the application. If the presented approach is for example used for the task of online 3D reconstruction, meaning that a subsequent depth map fusion step is employed (Hermann et al. 2021a), the geometric-consistency-based filtering is typically done in the fusion of depth maps and can thus be omitted. The DOG filter on the other hand is very efficient and does not introduce additional latency. However, as already mentioned in the experiments, the DOG filter might also remove potentially good estimates, as it is only executed based on the data provided by the input image. Nonetheless, especially when working with input data that contains a lot of homogeneous areas with little to no textures, e.g. a clear or cloudy sky in case of extreme oblique viewpoints, the DOG filter is of great benefit. (Ruf et al. 2021a, Sec. 4.4.)

Lastly, as a third output, FaSS-MVS computes a confidence map containing pixel-wise confidence scores corresponding to the depth estimates. In the scope of this work, these confidence measures are used to perform a comparison between the different SGM extensions based on a ROC analysis (cf. Figure 4.8). As already pointed out, the fact that some of the curves are not monotonically increasing suggests that the confidence values do not represent the certainty of the estimates appropriately. For one, the reason for this might lie in the normal map. Since confidence values are calculated based on the surface orientation stored in the normal map, errors in the normal map inevitably lead to unreliable confidence estimates. However, the qualitative excerpts in Figure 4.6, Figure 4.7 and Figure 4.11 object this explanation. For example, solely the fact that the scene in Row 6 of Figure 4.11 mostly consists of fronto-parallel structures leads to a confidence map with high certainty values, while the confidence map in Row 2 in Figure 4.11 renders the estimation of the building roof, which qualitatively appears very accurate, as fully uncertain. Similar observation can be seen in the confidence maps depicted in Figure 4.6. Only because the ground plane is greatly slanted with respect to the image plane, the confidence of the corresponding estimates is rendered very low even though qualitatively they do not appear more accurate than the estimates on the building facade. The most likely reason is that the modeling of a confidence score based on the surface orientation alone is not very expressive. An incorporation of additional heuristics, that are based on internal characteristics of the algorithm, as done in previous work (Ruf et al. 2019), might improve the certainty estimation, but this still requires a cumbersome empirical study of the hyper-parameters. In recent years, however, the performance of learning-based approaches for the task of confidence estimation (Poggi et al. 2020, Heinrich and Mehltretter 2021) has greatly increased. They are often agnostic to the internals of the algorithm and can be trained on any data for which both estimated and reference depth or disparity maps are available. (Ruf et al. 2021a, Sec. 4.4.)

4.6 Conclusion

In conclusion, with FaSS-MVS an approach is presented for multi-view stereo (MVS) from UAV-borne imagery that allows to facilitate fast, dense and incremental 3D mapping. This approach consists of a hierarchical processing scheme, in which dense depth maps as well as corresponding normal and confidence maps are estimated. For the depth map computation, dense multi-image matching, by utilizing the plane-sweep algorithm, is used to produce pixel-wise depth hypotheses. From these hypotheses, a dense depth map is extracted by

adopting the optimization scheme of the widely used Semi-Global Matching (SGM) algorithm. Here, the SGM algorithm is not only adapted to work with the multi-image matching of the plane-sweep algorithm, but also extended in order to reduce the fronto-parallel bias and, in turn, also account for slanted surface structure, by introducing two additional regularization schemes. The successive normal and confidence map estimation is done separately on the results of the depth estimation. In a final filtering step, geometric consistency over multiple depth maps is enforced, which greatly increases the overall accuracy of the resulting depth maps. (Ruf et al. 2021a, Sec. 5.)

The performance of FaSS-MVS is quantitatively evaluated on two public datasets containing image data of model-scaled scenes, captured from an aerial perspective and providing an accurate ground truth. The experiments show that, for the best configuration, the estimated depth maps have a mean absolute L1 error of only 8.5 mm, that is 1 % with respect to the maximum depth of the reconstructed scene. In comparison, the geometric depth maps from COLMAP, a widely used open-source toolbox for offline MVS, achieve a mean absolute error of 3.8 mm. Thus, even though FaSS-MVS does not have all image data of the input sequence available during the time of reconstruction and is subjected to run-time constraints in order to ensure fast and online processing, its quantitative results are not too far off from state-of-the-art offline approaches. While the quantitative results do not show a significant improvement by the presented SGM extensions to account for slanted surface structures, a qualitative comparison reveals their ability to account for non-fronto-parallel surfaces. Thus, in case of oblique aerial imagery, which contains a lot of slanted surfaces, the presented SGM extension that penalizes deviations from the gradient of the minimum cost path, i.e. $\text{SGM}^{\Pi\text{-pg}}$, is the best choice, despite its computationally higher complexity. Concluding experiments on real-world and use-case-specific datasets have shown, that in terms of run-time the presented approach is well-suited for online processing by achieving a processing rate of 1-2 Hz, meaning that it keeps up with the monocular input stream and allows for incremental 3D mapping, while the input data is being received. A fast 3D mapping, in turn, can facilitate other important applications or tasks, such as the fast assessment of inaccessible areas by emergency forces, e.g. after a flood or earthquake, in order to accomplish disaster relief or SAR missions. (Ruf et al. 2021a, Sec. 5.)

Finally, there are also some aspects to be considered in future work. Even though FaSS-MVS supports different plane orientations in the plane-sweep multi-image matching, to this end, each estimation is done with only one orientation. In future, the approach should be extended to use multiple plane orientations within the computation of a single depth map. This allows to reconstruct large planar surfaces, such as the ground plane more smoothly, but also allows to maintain higher accuracy in other regions by using fronto-parallel sampling. Furthermore, although it is stated that the processing rate is sufficient, a further decrease in run-time and a more efficient use of GPU resources would make up more opportunities for other concurrent tasks, such as the fusion of depth maps or the generation of orthographic photos. Thus, further optimization in terms of run-time and utilization of processing resources is an ongoing task. Moreover, due to the ongoing development and fast advancements of deep-learning-based approaches for the task of MVS, it is to be investigated, whether individual steps or even the whole approach can be substituted by an appropriate learning-based approach, while maintaining the reliability for the use in the scope of critical applications. Lastly, in their work, Nex and Rinaudo (2011) have shown that the complementary use of LiDAR and image-based techniques for photogrammetric tasks has great potential. In addition, with the improvements of LiDAR sensors and the possibility of equipping more and more COTS UAVs with such sensors, like the Zenmuse L1[1] , their use to facilitate fast and incremental 3D mapping is thus inevitably to be considered in future work. (Ruf et al. 2021a, Sec. 5.)

[1] https://www.dji.com/de/zenmuse-l1

As part of the framework that is proposed in this work (cf. Section 1.2), FaSS-MVS aims at providing depth estimates in order to allow a rapid 3D mapping from monocular video data captured by COTS UAVs, which in turn aids first responders in the assessment of a disaster site. The embedding of FaSS-MVS into a processing pipeline for rapid 3D mapping is discussed in Section 6.2. Moreover, the estimated depth data allows to perform 3D change detection for damage assessment, which is discussed in Section 6.3.

A major weakness of monocular MVS approaches, and in turn also of FaSS-MVS, is the inability to handle dynamic objects, such as pedestrians or cars, that are moving around in the scene during data acquisition. A possible remedy is to use approaches for two-view depth estimation from stereo cameras. However, due to the confined baseline of stereo cameras that can be fitted to COTS rotor-based UAVs, the depth range is limited. Another possibility to handle dynamic objects is the prediction of depth estimates from a single image by learning-based approaches. Such an approach for so-called single-view depth estimation is presented and discussed in the following chapter.

5 Learning of Single-View Depth Estimation from Aerial Imagery by Self-Supervision

In the previous chapters, the process of dense depth estimation is formulated by dense image matching (DIM), meaning that a dense correspondence field between two or more input images, which depict the scene from different vantage points, is to be found. From this, depth estimates can be deduced if the relative transformations between the images are known. The process of depth estimation by means of DIM is accompanied by the assumption that all images depict the same scene. In the case of monocular multi-view stereo (MVS) (cf. Chapter 4), this means that the scene geometry is not allowed to change between the individual images, making the reconstruction of scenes containing dynamic objects, e.g. an urban area with moving pedestrians or cars, cumbersome to reconstruct. When using a two-view stereo setup (cf. Chapter 2), on the other hand, the consistency of the depicted scene in the two images is enforced by a time-synchronized image acquisition, thus allowing to also estimate the depth for dynamic objects. However, in this case, the maximum depth that can be estimated is tightly coupled to the baseline between the cameras, which is relatively limited if the system is to be mounted to COTS UAVs (approximately 11.5 cm on the DJI Matrice 210v2 RTK), making such systems only feasibly for close-range photogrammetry and image exploitation.

Motivated by the capabilities of humans to guess relative depth estimates from a single image of a known scene, increasing effort has lately been put into attempts to learn how to estimate depth maps from single input images by means of deep learning (Eigen et al. 2014, Li et al. 2015, Godard et al. 2017). The idea is that, similar to the empirical knowledge of humans, deep convolutional neural networks (CNNs) are able to learn discriminative image cues, from which relative depth information can be inferred. This should especially hold, if the scene of the new image is somehow known to the CNN, i.e. if it is from the same dataset or at least similar to the scenes covered by the training data. In literature, this process is often referred to as *monocular depth estimation*. However, in order to not overload the term "monocular", since in this work it is already used within the context of performing MVS from images of a single moving camera, this process will be referred to *single-view depth estimation (SDE)*.

Apart from the already mentioned advantage of allowing to reconstruct dynamic scenes and possibly larger depth ranges compared to two-view stereo methods, there are a number of other advantages to SDE approaches. For one, the depth estimation by means of SFM and MVS is unstable if the camera has a large focal length, and thus a narrow field-of-view, or if the camera is moving in the direction of the optical axis. Moreover, unlike depth estimation from single images, approaches that rely on DIM typically suffer from occlusions, especially in case of two-view stereo. Lastly, a major advantage of SDE approaches over those relying on monocular MVS is their potentially higher processing speed. While MVS approaches require at least two images that are captured with an appropriate baseline, SDE approaches only require a single image to estimate the scene depth and thus are able to operate at a higher frequency.

Early work on SDE (Eigen et al. 2014, Li et al. 2015, Laina et al. 2016) relies on a supervised training, meaning that a reference depth map is required to evaluate the loss function and, in turn, to train the CNN. These studies show, that state-of-the-art approaches for SDE can outperform conventional approaches if appropriate training

data is available. The dependence on such data is at the same time, however, one of the major drawbacks of SDE approaches. This is due to the fact that the acquisition of datasets, which provide a large set of accurate reference data that is appropriate to perform supervised training of CNNs for the task of single-view depth estimation, is cumbersome and costly. Thus, only a few datasets exist that are appropriate for the desired task, e.g. those presented by Silberman et al. (2012), Menze and Geiger (2015) or Schöps et al. (2017), all of which are made up of terrestrial imagery and thus are not suited to learn to estimate depth from a single aerial image.

In order to overcome this limitation, recent studies have proven that CNNs can also be trained for the task of SDE by self-supervision. Self-supervision means, that in the training process, such approaches only require images from a stereo camera or from a video captured by a single moving camera, similar to the input data required by conventional stereo and multi-view stereo approaches. This can be accomplished by posing the task of depth estimation during training as a view synthesis and image reconstruction problem. In this, one or more neighboring images are transformed into the view of a reference camera, based on a predicted depth map, and matched with the corresponding reference image. The assumption is, that the CNN has learned to predict the depth correctly if the synthesized image and the reference image coincide, i.e. if the matching costs are at their minimum. Thus, such approaches do not require any special training data, because they supervise their training by themselves. Such approaches are referred to as *self-supervised single-view depth estimation (SSDE)* or *self-supervised monocular depth estimation (SMDE)* approaches. The flexibility regarding training data allows to use SSDE on any domain without the need of costly acquired training data, possibly eliminating a major impediment of deep-learning-based approaches for the practical use in the scope of depth estimation.

5.1 Contributions and Outline

In this chapter, it is studied, how modern SSDE or SMDE approaches, which have achieved great results in the context of autonomous driving and driver assistance (Godard et al. 2017, Wang et al. 2018, Mahjourian et al. 2018, Godard et al. 2019), can be adopted for the task of learning single-view depth estimation from aerial imagery captured by a COTS UAV. To this aim,

- an approach is presented that demonstrates the feasibility to train a CNN to estimate a depth map from a single aerial image, solely based on images from a single moving camera,
- the depth maps estimated by the proposed SSDE approach are compared and evaluated against the results from conventional approaches based on DIM and MVS, and
- the ability for generalization of such an SSDE approach, i.e. the performance achieved on images from unknown scenes, is investigated and the possible practical use for the task of depth estimation is discussed.

The contributions presented in these chapters have partly been published in:

- Hermann, M.; Ruf, B.; Weinmann, M., and Hinz, S. (2020): "Self-supervised learning for monocular depth estimation from aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-2-2020, pp. 357–364. **Peer-reviewed** on the basis of the full paper. Cited as (Hermann et al. 2020).

This chapter is structured as follows: In Section 5.2, related work on deep-learning-based approaches for the task of depth estimation from a single image is discussed, both by supervised and self-supervised training. This is followed by a detailed description in Section 5.3 focusing on the self-supervision process that is employed

in this work. Details on the inference of depth maps from a single image by an appropriately trained CNN as well as details on the implementation are given in Section 5.4 and Section 5.5, respectively. The experimental results are presented and discussed in Section 5.6 and Section 5.7, respectively, before providing concluding remarks in Section 5.8.

5.2 Related Work

In the following sections, the related work on deep-learning-based approaches for the task of SDE is presented and briefly discussed. First, Section 5.2.1 introduces related work that is based on supervised learning, i.e. approaches that rely on a reference depth map to train a CNN for the task of single-view depth estimation. This is followed by the presentation of approaches that rely on self-supervised learning in Section 5.2.2, before providing an overview on related work that relies on CNNs, that are trained in a self-supervised manner for the task of estimating depth maps from a single aerial image in Section 5.2.3.

5.2.1 Supervised Learning of Single-View Depth Estimation

As already stated, with the advancements achieved by CNNs, increasing effort is put into trying to train a deep-learning-based model to predict a depth map from a single input image. This is motivated by the ability of humans to deduce the scene structure from a single image based on their empirical knowledge. Early work started off by training deep CNNs in a supervised manner, requiring reference data to generate a training signal. Eigen et al. (2014) present an approach that relies on a cascade of two CNNs, where the first network focuses on the estimation of a coarse depth map based on global prediction, which is refined by the second network to produce the final result. In further work, Li et al. (2015) propose to use a CNN to regress deep features from the input image, which are then refined by a hierarchical conditional random field (CRF) to produce the final output. In their work, Laina et al. (2016), on the other hand, rely on a single CNN, with an encoder-decoder structure based on the ResNet50 (He et al. 2016), that is trained end-to-end without the need of post-processing by conventional methods. A great impediment of such approaches, however, is their dependence on training data containing reference depth maps, of which only a small number exist, e.g. those presented by Silberman et al. (2012), Menze and Geiger (2015), and Schöps et al. (2017), all of which consist of ground-based imagery. One method to overcome this limitation is to use synthetically generated training data (Johnson-Roberson et al. 2017, Mayer et al. 2018). However, the process of creating large amounts of training data, which is realistic enough to allow generalization of the resulting model to real-world scenes, is cumbersome, time-consuming and costly. Moreover, the generalization of a trained model across different domains, be it between synthetic and real-world scenes or between terrestrial and aerial imagery, is still very challenging and does not always achieve satisfying results.

5.2.2 Self-Supervised Learning of Single-View Depth Estimation

Self-supervised techniques provide another possibility to overcome the limitation of requiring appropriate training data. Such approaches formulate the training task as a novel view synthesis and image reconstruction problem and thus supervise themselves in the training for the task of depth prediction. Early works (Flynn et al. 2016, Xie et al. 2016) train a model to synthesize images from new viewpoints by estimating the scene depth and transforming the available imagery into the new vantage points. In this, the prediction of depth is just an auxiliary task in order to correctly sample new images, which requires an understanding of the scene geometry.

However, this technique can also be used to facilitate the learning for the task of depth map prediction by a subsequent comparison between the newly synthesized image and the actual image captured by a camera from the corresponding viewpoint. Khot et al. (2019), for example, employ such a training technique to learn how to predict a depth map from multiple images. But it can also be used to train models for the task of single-view depth estimation, as shown, for example, by Godard et al. (2017) and Godard et al. (2019).

A key requirement to sample synthetic images from a new viewpoint are the intrinsic camera parameters as well as the extrinsic transformation between the individual input images. This makes approaches which are based on images from a fixed stereo camera setup, like the one of Godard et al. (2017), particularly suitable, since the relative transformation between the cameras can be calibrated a priori and stays the same for all images. However, as previously already explained, depending on the use-case and the application, a stereo camera setup is not always suitable, due to the limited baseline. Thus, other approaches (Mahjourian et al. 2018, Wang et al. 2018, Zhao et al. 2016, Godard et al. 2019) rely on the paradigm of monocular SFM and MVS and only use images of a single moving camera during training. While this, in turn, allows to train CNNs for the task of SDE directly from video data, it requires the model to additionally learn how to predict the extrinsic orientations between the input images, increasing the number of parameters to train. Nonetheless, the independence of additional reference data during training makes such approaches especially fit for the learning-based single-view depth estimation from aerial imagery captured by COTS UAVs, since the acquisition of adequate training material is much easier.

5.2.3 Self-Supervised Learning of Single-View Depth Estimation from Aerial Imagery

The majority of approaches for the task of SSDE arise from the context of autonomous driving, which is, amongst other reasons, due to the prominent availability of appropriate datasets, such as the KITTI (Menze and Geiger 2015) or Cityscapes (Cordts et al. 2016) benchmarks. Due to higher degrees of freedom, i.e. varying flight altitude, camera movement and orientation as well as downwards looking perspective, aerial imagery is, however, significantly different compared to recordings of street scenes. In their work, Knöbelreiter et al. (2018) demonstrate, as one of the first, the use of self-supervised learning for stereo reconstruction from aerial imagery. However, they use a rectified stereo image pair as input data and refine the estimated depth map by a subsequent hybrid method, relying on a deep CNN and a CRF (Knöbelreiter et al. 2017).

The approach for SSDE from aerial imagery presented in this work, in contrast, aims to directly use and evaluate the predicted depth maps without any post-filtering or post-optimization. Moreover, similar to the latest approaches that derive from the context of autonomous driving (Mahjourian et al. 2018, Wang et al. 2018, Zhao et al. 2016, Godard et al. 2019), the presented approach only uses image data from a single moving camera mounted to a COTS UAV during training. The approach presented by Madhuanand et al. (2021) also relies on video data captured by a camera mounted to a moving COTS UAV for the self-supervised training of a CNN for SDE from aerial imagery. In contrast to the work presented in this chapter, it uses a different network architecture based on a 2D CNN encoder and a 3D CNN decoder as well as two additional loss functions. This slightly improves the performance of the resulting depth maps compared to the approach that is presented in this chapter and which was published in the paper of Hermann et al. (2020).

5.3 Methodology of Learning Single-View Depth Estimation by Self-Supervision

In the following section, the actual procedure of self-supervised learning for the task of SDE implemented in the scope of this work is described and will be denoted as SSDE. In this, first an overview over the complete procedure is given, before explaining details of each single step.

Similar to the approach described in Chapter 4 for Fast Multi-View Stereo with Surface-Aware Semi-Global Matching (FaSS-MVS), the training of the SSDE approach requires a set of image bundles consisting of 3 or more images from an input sequence that depict a static scene from different vantage points. Just as for approaches that perform depth estimation based on DIM, it is crucial that there is enough parallax, i.e. perspective difference, between the input images in order to allow depth estimation, while ensuring only a little amount of occluded areas which hinder the process of image matching. While the compliance of this criterion can be enforced by an intelligent pre-selection of the image data based on image content and optical flow methods (Hermann et al. 2021b), it is empirically determined in the scope of this work. Again, the center image of the input bundle is referred to as reference image $\mathcal{I}_{\text{ref}}$ for which a depth map $\mathcal{D}$ is to be predicted, while the rest of the images $\mathcal{I}_k$ are grouped into a left ($k < \text{ref}$) and a right ($k > \text{ref}$) subset with respect to $\mathcal{I}_{\text{ref}}$. While it is assumed that the intrinsic camera matrix $\mathbf{K}$ is given, the relative extrinsic transformations $\mathbf{E}_{\text{ref}\to k} = [\mathbf{R}\ \text{T}]$ between the images are not required.

Every single training iteration is subdivided into four consecutive steps as illustrated by Figure 5.1. Step (1) performs single-view depth estimation to predict the depth map $\mathcal{D}$ from $\mathcal{I}_{\text{ref}}$. In step (2), the relative transformations $\mathbf{E}_{\text{ref}\to k}$ consisting of rotation $\mathbf{R}$ and translation T between the reference view (corresponding to $\mathcal{I}_{\text{ref}}$) and its neighboring views are predicted. Given the depth map $\mathcal{D}$ and the relative transformations $\mathbf{E}_{\text{ref}\to k}$, new synthetic images corresponding to the reference view are sampled from the neighboring images in step (3). Finally, the newly sampled reference images are matched against the actual reference image $\mathcal{I}_{\text{ref}}$ in step (4), calculating the reconstruction error and in turn the training loss, which is then backpropagated through the

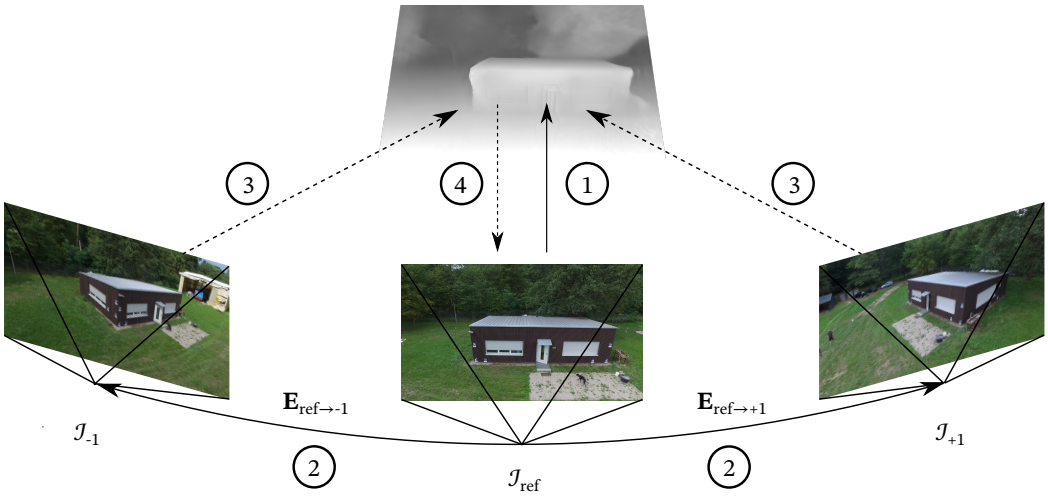


Figure 5.1: Illustration of the self-supervised training process for the task of single-view depth estimation. Step (1) predicts the depth map $\mathcal{D}$ from $\mathcal{I}_{\text{ref}}$. Step (2) predicts the relative transformations $\mathbf{E}_{\text{ref}\to k}$ (with $k = \pm 1$) between the reference view, corresponding to $\mathcal{I}_{\text{ref}}$, and its neighboring views, indicated with $\mathcal{I}_{\text{-}1}$ and $\mathcal{I}_{+1}$ which represent a preceding and following view of the image sequence, respectively. Step (3) samples synthetic images given the neighboring images as well as the predicted depth $\mathcal{D}$ and the predicted relative transformations $\mathbf{E}_{\text{ref}\to k}$. Step (4) matches the synthetic views against $\mathcal{I}_{\text{ref}}$ and calculates the reconstruction error. (Hermann et al. 2020, Fig. 2)

CNN. This procedure is executed in each training iteration and thus repeated thousands of times within the complete training of the CNN for the task of single-view depth estimation. At the beginning of the training, the predictions of $\mathcal{D}$ and $\mathbf{E}_{\text{ref}\to k}$ will obviously be of very low quality, resulting in a high matching score and, in turn, a high training loss. With ongoing training, the quality of the predictions will improve more and more, while the network learns how to minimize the training loss by predicting the corresponding data correctly. When the training loss, and with it the matching cost between $\mathcal{I}_{\text{ref}}$ and the synthetic image sampled from the neighboring views, has reached its minimum, it is assumed that the CNN has learned to predict the data correctly, as a new view can only be convincingly sampled, if the scene geometry and extrinsic camera positions are correct. In the following sections, each step of the outlined procedure is described in detail.

5.3.1 Single-View Depth Estimation

As a network topology for the CNN responsible to estimate a depth map $\mathcal{D}$ from a single input image, the U-Net architecture introduced by Ronneberger et al. (2015) is adopted and illustrated by Figure 5.2. The U-Net is made up of an *encoder*, which transforms the input image into a deep feature representation, from which the output is then generated by the *decoder*. As encoder, the ResNet18 architecture (He et al. 2016) is used, since it achieves a higher accuracy than other architectures, such as the VGG-Net16 (Simonyan and Zisserman 2015), while at the same time having less trainable parameters, as discussed in Section 5.6.4.1. To further reduce the number of trainable parameters, the approach of Godard et al. (2017) is employed, that relies on simple nearest neighbor upsampling instead of a transposed convolution. So-called skip connections between individual layers of the encoder and decoder ensure that features, that have been learned in early stages of the CNN, also have an impact to the final output. In this, the features of the encoder are concatenated with the corresponding features of the decoder. As activation functions in the individual layers of the decoder, exponential linear units (ELUs) are employed. To compute the depth map, a final Sigmoid activation is used.

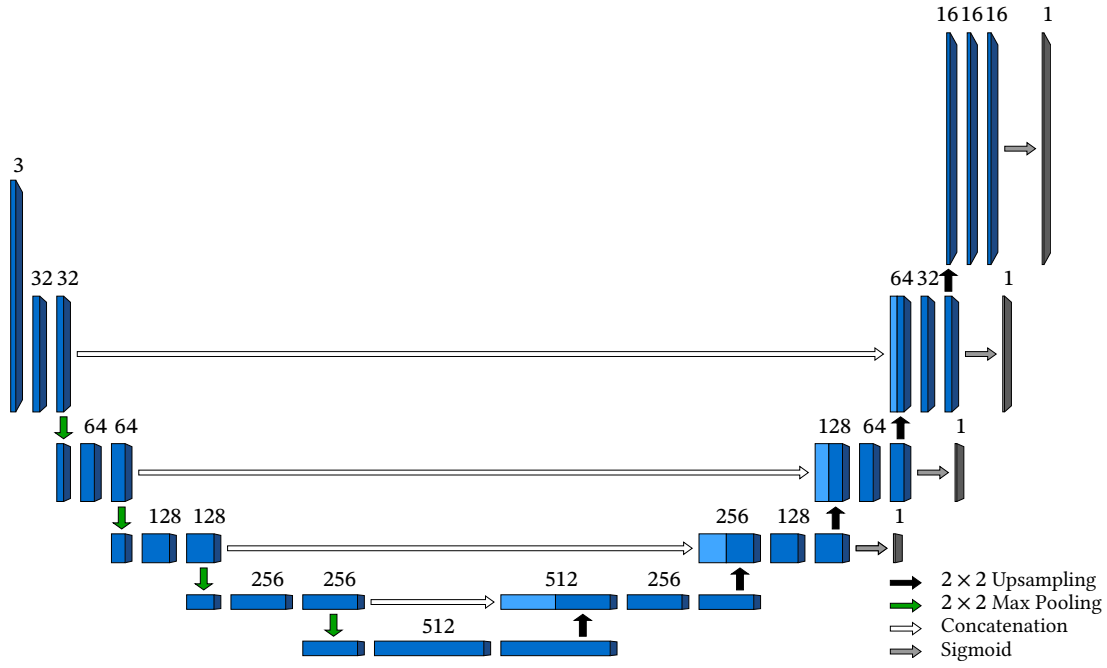


Figure 5.2: Illustration of the structure of the U-Net employed for single-image depth estimation. It comprises an encoder and a decoder. As encoder the ResNet18 architecture is used, while for the decoder a simple 2×2 nearest neighbor upsampling is employed. As activation functions in the individual layers of the decoder, exponential linear units (ELUs) are employed. To compute the depth map, a final Sigmoid activation is used.

5.3.2 Relative Pose Estimation

In the second step of the self-supervised training, the relative extrinsic transformations $\mathbf{E}_{\text{ref}\to k}$, i.e. relative poses, between the view of the reference camera and the neighboring views, and in turn between the corresponding images, are estimated. While this transformation estimation could also be done using image features, the presented approach pursues to rely on deep learning to estimate the relative poses, allowing for a full end-to-end training, similar to that presented by Zhou et al. (2017). In this, each additional input image is also transformed into a deep feature representation by an encoder CNN, just as the reference image in the scope of predicting the depth map (cf. Section 5.3.1). In order to keep the resulting model at a small size, the weights of the depth estimation encoder and the relative pose estimation encoders are shared (cf. Figure 5.3). This means that for the computation of the deep feature representation, the same layers of the U-Net encoder are used for both depth and pose estimation. This also has a positive effect on the time it takes to train the model, since less parameters need to be trained. The sharing of the encoders is also motivated by the fact that the depth and pose estimation are both tightly coupled to the scene geometry, which should ideally be correctly encoded in the deep feature maps. As illustrated by Figure 5.3, the deep features of all input images, calculated by the corresponding encoders, are concatenated and then passed through a sequence of three fully connected layers, followed by a global average pooling in the last layer of the network. The output is a vector containing the rotational and translational parameters of $\mathbf{R}$ and T respectively, both relative to the reference view.

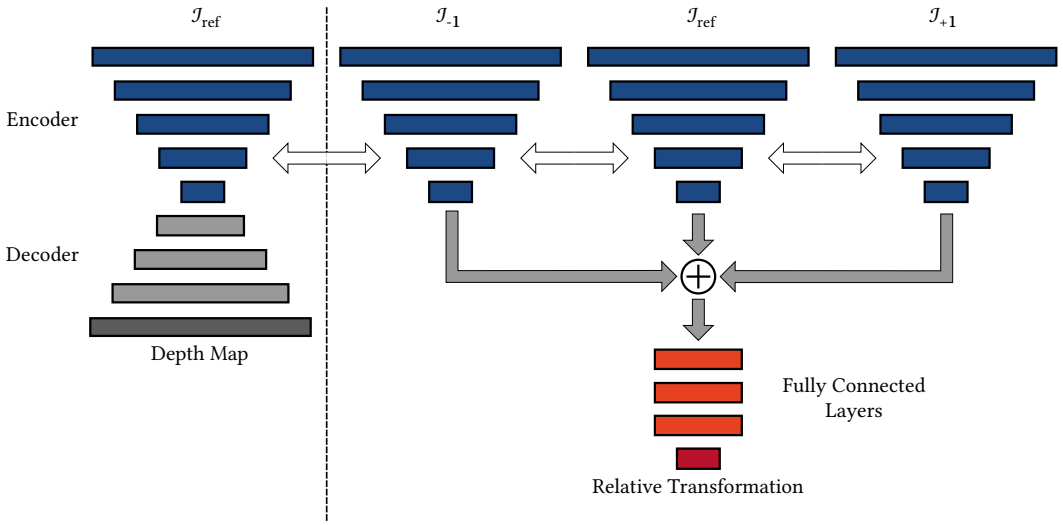


Figure 5.3: Illustration of the network architecture to estimate the relative transformation between images from the reference view and its neighbors. The encoder is shared between the networks responsible for the depth estimation and relative pose estimation, illustrated by the same color and the white arrows. The deep feature maps from the encoders are then concatenated before being passed through three fully connected layers and a final Sigmoid activation layer. (Hermann et al. 2020, Fig. 3)

5.3.3 Image Projection

In order for the approach to compute a training loss and, in turn, for the CNN to learn the parameters for predicting the depth map $\mathcal{D}$ and the relative transformations $\mathbf{E}_{\text{ref}\to k}$, the CNN needs to know how to project the images from the neighboring views $\mathcal{I}_k$ (in the following denoted as matching images) into the view of the reference image $\mathcal{I}_{\text{ref}}$. To facilitate complex processes like image projection and image sampling, Jaderberg et al. (2015) proposed the concept of *spatial transformer networks (STNs)*. Instead of actually training a network to

learn specific algorithm, Jaderberg et al. (2015) propose to use STNs to learn the parameters of a predefined algorithm. For example, in the case of image rotation, the STN does not have to learn how to actually rotate an image but only by how much it is to be rotated. An STN is composed of a *Localization Net*, a *Grid Generator* and a *Sampler*. The Localization Net learns the parameters of a predefined transformation. Within the Grid Generator, the target coordinates of a regular grid are then transformed into destination coordinates using the previously learned parameters. Lastly, given these destination coordinates, the Sampler samples the new image.

Given the intrinsic calibration matrix $\mathbf{K}$, the projection of a matching image $\mathcal{I}_k$ into the view of the reference image $\mathcal{I}_{\text{ref}}$ is formulated as:

$$\mathcal{I}_{k\to\text{ref}} = \mathcal{I}_k \left[P(\mathcal{D}, \mathbf{E}_{\text{ref}\to k}, \mathbf{K})\right]. \tag{5.1}$$

Here, P denotes the predefined projective transformation, for which the parameters in the form of the predicted depth map $\mathcal{D}$ and the predicted relative extrinsic transformations $\mathbf{E}_{\text{ref}\to k}$ are to be learned. As part of the actual sampling $[\cdot]$, that transforms $\mathcal{I}_k$ to create the new image, a bilinear interpolation is employed. As illustrated by Figure 5.2, four individual depth maps with four different resolutions are computed after each upsampling layer of the decoder. The use of multi-scale estimation allows to stabilize the training. In this, instead of downsampling $\mathcal{I}_{\text{ref}}$ to the resolution of $\mathcal{D}$, the approach proposed by Godard et al. (2019) to upsample $\mathcal{D}$ to the resolution of $\mathcal{I}_{\text{ref}}$ is used.

5.3.4 Image Matching and Loss Calculation

Since this approach is not based on supervised learning, it does not rely on a reference depth map with respect to which a loss function is formulated. In contrast, as previously described, the network is trained by a loss function L, which compares the projection of a matching image $\mathcal{I}_k$ into the view of the reference image $\mathcal{I}_{\text{ref}}$, given the predicted depth map $\mathcal{D}$ and relative transformation $\mathbf{E}_{\text{ref}\to k}$. Thus, the employed training loss does not allow a direct reasoning about the quality of the predicted depth map based on a given reference. Nonetheless, induced by the nature of the approach, a good image projection and sampling is only possible if $\mathcal{D}$ and $\mathbf{E}_{\text{ref}\to k}$ are predicted correctly. And thus, the loss function L is formulated to evaluate the quality of the image projection and sampling, similar to the cost function in the scope of the plane-sweep multi-image matching (cf. Section 4.3.2). In particular, it is formulated according to:

$$L = L_{\text{photo}} + \lambda\, L_{\text{smooth}}, \tag{5.2}$$

with L_{photo} being a photometric loss modeling the quality of the image reconstruction, and with L_{smooth} being a smoothness loss enforcing smooth depth maps. The weighting parameter is empirically set to $\lambda = 0.0001$. In the following, the calculation of the photometric and the smoothness loss is described in detail.

Photometric Loss:

Just as in the plane-sweep multi-image matching of FaSS-MVS (cf. Section 4.3.2), the approach of Kang et al. (2001) to account for occlusions is used, by separately calculating the loss for the left and right subset of $\mathcal{I}_{\text{ref}}$ and taking the minimum of both for the final loss, since it can again be assumed that areas, which are occluded in one subset, are visible in the other. Thus, the photometric loss is the minimum of the left and right

reconstruction error s according to:

$$\begin{aligned} L_{\text{photo}} &= \min\left\{ \sum_{k<\text{ref}} s(\mathcal{I}_k,\mathcal{I}_{\text{ref}}) \,,\, \sum_{k>\text{ref}} s(\mathcal{I}_k,\mathcal{I}_{\text{ref}}) \right\} \,, \text{ with} \\ s(\mathcal{I}_a,\mathcal{I}_b) &= \alpha \, \frac{1 - SSIM(\mathcal{I}_a,\mathcal{I}_b)}{2} + (1-\alpha)\,|\mathcal{I}_a - \mathcal{I}_b| \,. \end{aligned} \tag{5.3}$$

Here, the reconstruction error s is comprised of a *structural similarity (SSIM)* measure and a L1-Norm, similar to the work of Godard et al. (2017), Mahjourian et al. (2018), and Madhuanand et al. (2021). The SSIM measure is evaluated by Zhao et al. (2016) to provide visually appealing results for the task of image reconstruction by deep CNNs and is often used in the scope of view synthesis. As in the approach presented by Godard et al. (2017), the Gaussian kernel within SSIM is substituted by a 3×3 box filter, leading to a simpler implementation and a more efficient processing. The c_1 and c_2 parameters of SSIM are set to 0.0001 and 0.0009, respectively. In the experiments, the weighting factor between the SSIM and the L1-Norm is set to $\alpha = 0.15$, yielding a more stable convergence of the training.

Smoothness Loss:

While the smoothness loss aims to enforce a smooth depth map $\mathcal{D}$ and helps to compensate the inability to calculate the photometric loss in weakly-textured areas, it is also desired to preserve depth discontinuities at object boundaries. Thus, in order to not smooth over edges too strongly, an edge-aware smoothness loss is employed, like that presented by Wang et al. (2018). Just as with the adjustment of the SGM penalty φ_2 (cf. Section 4.3.3.2), the aim is to adjust the loss function based on the image gradient in $\mathcal{I}_{\text{ref}}$. Thus, the smoothness loss is formulated according to:

$$L_{\text{smooth}} = \left|\frac{\partial_u \mathcal{D}}{\bar{\mathcal{D}}}\right| \exp\left(-\left|\partial_u \mathcal{I}_{\text{ref}}\right|\right) + \left|\frac{\partial_v \mathcal{D}}{\bar{\mathcal{D}}}\right| \exp\left(-\left|\partial_v \mathcal{I}_{\text{ref}}\right|\right) \,. \tag{5.4}$$

In this, the color gradient in u- and v-direction of $\mathcal{I}_{\text{ref}}$ is used to weight the smoothness and, in turn, the gradient of the depth map $\mathcal{D}$. Moreover, to account for the problem of a degrading depth map, the approach of depth normalization $\mathcal{D}/\bar{\mathcal{D}}$ presented by Wang et al. (2018) is used.

5.4 Inference of Depth from a Single Aerial Image

The presented approach aims at predicting a depth map from a single aerial image. And thus, there is no need for an image bundle to estimate the depth map. Consequently, the fully connected layers, used to estimate the relative transformations $\mathbf{E}_{\text{ref}\to k}$ between the reference image and the neighboring images during training, are not required during the time of inference. Solely, the U-Net, which predicts the depth map $\mathcal{D}$ within the first step of the training routine, is used during inferencing. This again has a positive effect on the processing speed, as the number of parameters and, in turn, the size of the model is further reduced. However, as discussed in Section 5.7.3, when the model is to be used and retrained on a new scene, the whole network, including the part which is responsible for predicting the relative transformations, is needed.

5.5 Implementation Details and Training Procedure

An implementation of the presented approach in Python, using the deep learning framework TensorFlow, is released as open-source on GitHub[1]. During training, the Adam optimizer (Kingma and Ba 2015) is used with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The learning rate is set to 0.0002. While Godard et al. (2019) use a batch size of 4 to train their approach on data of the KITTI benchmark, the training of the presented approach on aerial imagery does not converge with a batch size of 4 (cf. Section 5.6.4.2). Thus, in the scope of this work, a batch size of 20 is used. As input bundle, three input images are used. During training, the model of the presented approach has approximately 21 million parameters. However, since only the depth prediction network is required during the time of inference, provided that no continual learning is required, this can be reduced to 17 million parameters.

During training, the actual input data is augmented with randomly cropped and scaled image copies, in order to increase the versatility of the training data. Moreover, for 50 % of the input bundles, the images were horizontally flipped in order to avoid a bias in the camera movement, as this inverts the relative transformations which are to be predicted by the pose estimation network. Lastly, the brightness, contrast and saturation are adjusted randomly with deviations of up to 20 %, while hue is randomly adjusted up to 10 %.

In the scope of this work, no pre-trained model is used and the network is trained from scratch. It is trained up to the point where the curve of the training loss flattens. During training, by validating the model after each training period, it was noted that the validation error does not improve after approximately one third of the total training time. Thus, the time used to train the model is calculated up to the training epoch, after which the lowest validation error is reached for the first time. For an image size of 384×224 pixels, which yields the best results while, at the same time, ensuring a stable training (cf. Section 5.6.4.3), the proposed network requires around 100 epochs for the full dataset of about 10,000 images (cf. Section 5.6.1), resulting in a training time of roughly 24 hours on a NVIDIA Titan X GPU. The training time changes proportionally when a different image size is used.

With the above-mentioned image resolution, the trained model achieves a frame rate of up 20 FPS on the Titan X GPU without any optimization. This includes the loading of the image data into GPU memory as well as the pre-processing of the images, i.e. resizing them to the input resolution, and writing the output onto the hard drive.

5.6 Experiments

In the following sections, the quantitative and qualitative results from the conducted experiments are presented. First, an overview of the utilized evaluation datasets and the error measures is given in Section 5.6.1 and Section 5.6.2, respectively. Then, the quantitative results achieved on street-level imagery are presented in Section 5.6.3.1, together with results from literature and ReS^{2}tAC on the same datasets as comparison. This is followed by a presentation of the results achieved on aerial imagery in Section 5.6.3.2, which also contains a comparison to FaSS-MVS. Lastly, the results of an ablative study with respect to different aspects of the approach are presented in Section 5.6.4. All experiments, including training and inferencing, were done on a NVIDIA Titan X GPU.

[1] https://github.com/Max-Hermann/SelfSupervisedAerialDepthEstimator

5.6.1 Evaluation Datasets

The overall performance of the presented approach for SSDE is quantitatively evaluated on three different datasets: on the dataset from the KITTI benchmark (Menze and Geiger 2015) comprised of street-level imagery, on a synthetic dataset comprised of aerial imagery with accurate ground truth, and on the TMB dataset (cf. Section 4.4.1) containing real-world aerial imagery. While the two aerial datasets are used to evaluate and demonstrate the ability of the approach for SSDE in a use-case-specific environment and compare its results to that of FaSS-MVS, the KITTI benchmark allows a comparison with respect to other approaches, such as those presented by Godard et al. (2019) and Zhou et al. (2017) as well as ReS^{2}tAC. In case of the KITTI benchmark, ground truth data is captured by a LiDAR sensor providing an accurate point cloud from which the ground truth depth maps are deduced. As described in Section 4.4.1, in case of the TMB dataset, the offline SFM and MVS toolbox COLMAP (Schönberger and Frahm 2016, Schönberger et al. 2016) is used to generate a reference. Moreover, in a qualitatively comparison, depth maps that are predicted by the presented approach on the FB dataset (cf. Section 4.4.1) are compared against depth maps estimated by FaSS-MVS.

While reference data computed by COLMAP is expected and shown to be more accurate than that produced by FaSS-MVS and SSDE, it is still deduced from the same modality, namely the camera images, and thus the depth maps, which are used as reference, will not be perfectly accurate. To overcome this issue, the synthetic dataset is generated by exploiting the capabilities of modern computer graphics and rendering engines in order to extract an accurate ground truth. In particular, the open-source software *GTAVisionExport* (Johnson-Roberson et al. 2017) is adopted to extract color images together with corresponding high-resolution ground truth depth maps from the video game *Grand Theft Auto V (GTA V)*. While the level-of-detail between real-world and synthetic data greatly differs, the rendering of synthetic data is typically more flexible, allowing to manipulate the environment and define a wide range of different camera trajectories. Thus, in case of the synthetic dataset, henceforth referred to as the GTA dataset, a variety of scenes with different flight altitudes and flight trajectories, such as linear fly-overs or circular flights orbiting a building, as well as different weather conditions and daytimes are simulated. The ground truth depth maps are truncated at 100 m.

For all datasets, around 5000 images were used, splitting them with a 90/10 % ratio into training and evaluation set. In case of the two aerial datasets, training was done using a fused set of all training images from both datasets, yielding the same results as when training on the two datasets separately.

5.6.2 Error Measures

Since the depth maps predicted by SSDE are free of a metric scale, only relative error measures are used to assess the quality of the predicted depth maps. In particular, the relative L1-norm (L1-rel) (cf. Equation (4.14)) between the prediction and the reference as well as the completeness (Cpl_θ) (cf. Equation (4.16)) are used, both introduced in the evaluation of FaSS-MVS in Section 4.4.2. In case corresponding estimates exist for all ground truth values, the completeness corresponds to the accuracy δ_θ used by Godard et al. (2019), Zhou et al. (2017) and Hermann et al. (2020). This holds for the depth maps predicted by the SSDE approach due to their density of 100 %. To account for differences in resolution as well as depth scale between the predicted depth maps and the reference data, an image resizing to the resolution of the reference as well as a median scaling of the predicted depth is used.

5.6.3 Experimental Results

In the following, the experimental results achieved on the dataset of the KITTI benchmark (Section 5.6.3.1) and on aerial imagery (Section 5.6.3.2) are presented.

5.6.3.1 Street-Level Imagery

In order to allow a comparison with respect to other corresponding approaches from literature (Godard et al. 2019, Zhou et al. 2017) as well as the approach for Real-Time SGM Stereo Optimized for Embedded ARM and CUDA Devices (ReS^{2}tAC), which is presented in Chapter 3 of this work, an evaluation on the data from the KITTI benchmark (Menze and Geiger 2015) is performed. Even though, the nature and characteristics of the data is not relevant for the use-case considered in the scope of this work. To train the approach on the KITTI data, the raw data of individual sequences is used, allowing to create temporal image bundles from the same camera, instead of the actual stereo data. The model was trained with an image size of 416×128 pixels and a batch size of 4. In the evaluation, the ground truth depth maps are truncated at 80 m.

Table 5.1 lists the quantitative results achieved by the presented approach for SSDE on the dataset of the KITTI benchmark together with the results of two other approaches for SSDE as well as those achieved by ReS^{2}tAC. The results reveal that the presented approach achieves similar results as related approaches from literature. However, compared to a conventional depth estimation approach based on two-view stereo, such as ReS^{2}tAC, the SSDE approaches are inferior. Figure 5.4 shows an excerpt of the depth maps predicted by the presented approach on data of the KITTI benchmark, together with the corresponding input image and ground truth. Since the LiDAR sensor, from which the ground truth is generated, only covers the lower part of the camera image, there is a big area in the ground truth depth map for which no reference is provided, as depicted with black in Figure 5.4.

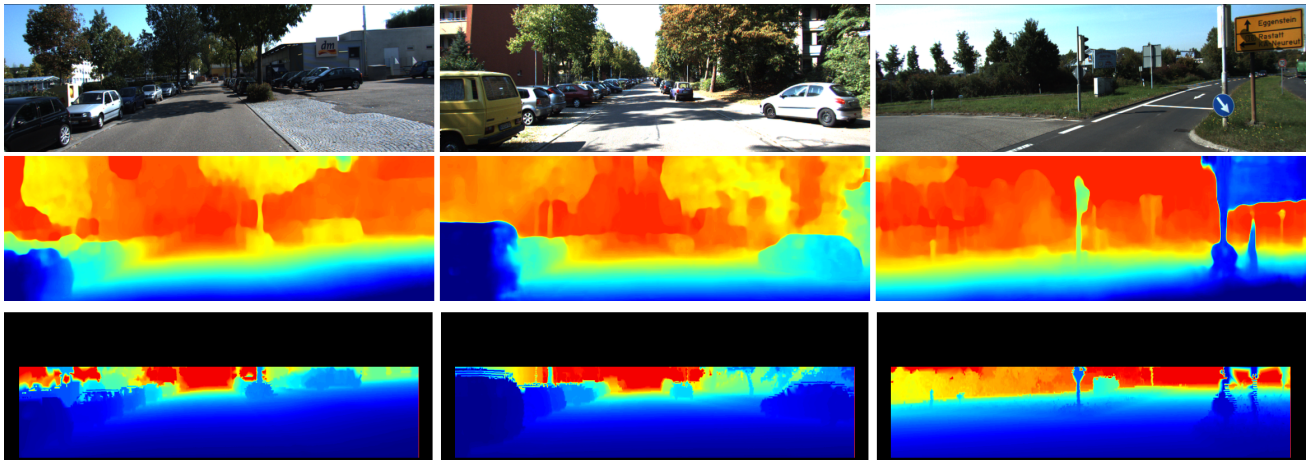


Figure 5.4: Qualitative results of the approach for self-supervised single-view depth estimation on data from the KITTI benchmark. **Row 1:** Reference images. **Row 2:** Estimated depth maps. **Row 3:** Ground truth depth maps generated from the LiDAR point clouds. The depth is color-coded, going from blue (near) via yellow to red (far). Since the scaling between the predicted depth maps and the reference differs, the color gradients between the two modalities do not coincide.

Table 5.1: Quantitative results achieved by the presented approach for self-supervised single-view depth estimation (SSDE) on the dataset of the KITTI benchmark. As evaluation metrics, the L1-rel as well as the completeness at a threshold of **25** % with respect to the ground truth depth is used. In addition, the results of two other approaches for SSDE as well as those achieved by ReS^{2}tAC are listed for comparison.

Approach	L1-rel	$\text{Cpl}_{1.25}$ (in %)
Zhou et al. (2017)	0.183	73.4
Godard et al. (2019)	0.133	84.1
ReS^{2}tAC	<u>0.043</u>	<u>88.6</u>
SSDE	0.272	83.9

5.6.3.2 Aerial Imagery

Table 5.2 lists the quantitative results of the presented approach for SSDE achieved on the two aerial image datasets, namely TMB and GTA. While the L1-rel error achieved by SSDE on the GTA dataset is lower than that achieved on the TMB dataset, the completeness achieved on the TMB dataset is higher than that achieved on the GTA dataset. This irregularity is assumed to arise from the higher scene depth of the GTA dataset, since a larger ground truth depth will have a greater normalization effect on the L1-rel error. Thus, for comparison between different datasets, the completeness is more meaningful and, in turn, reveals that the SSDE approach achieves the better results on the TMB dataset. The reason for this could lie in the lower scene complexity and variety of the TMB dataset compared to that of the GTA dataset. The strength of SSDE approaches on datasets with lower scene complexity is also discussed by Madhuanand et al. (2021).

In addition to the sole evaluation of the results from the SSDE approach with respect to the ground truth, a comparison to the results achieved by FaSS-MVS with SGM$^{\Pi\text{-pg}}$ (cf. Chapter 4) is done in Table 5.2. As expected, the results of the SSDE approach are inferior to those achieved by approaches that use two or more views. In

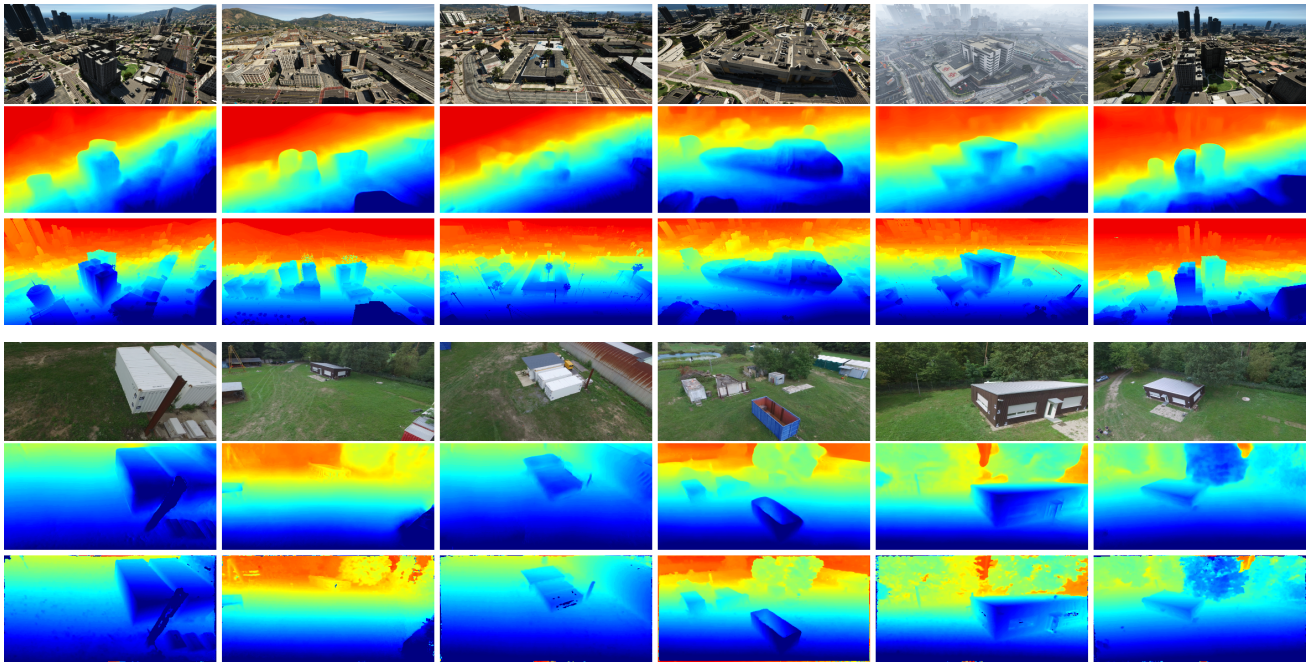


Figure 5.5: Qualitative results of the approach for SSDE on aerial imagery from the GTA V (Rows 1-3) and TMB (Rows 4-6) dataset. **Rows 1 & 4:** Reference images. **Rows 2 & 5:** Estimated depth maps. **Rows 3 & 6:** Reference depth maps rendered from the game engine in case of the GTA V dataset and generated by COLMAP in case of the TMB dataset. The depth is color-coded, going from blue (near) via yellow to red (far). Since the scaling between the predicted depth maps and the reference differs, the color gradients between the two modalities do not coincide. (Hermann et al. 2020, Fig. 4.)

Table 5.2: Quantitative results achieved by the presented approach for SSDE on aerial imagery. As evaluation metrics, the L1-rel as well as the completeness at a threshold of 25 % and 5 % with respect to the ground truth depth is used. For comparison, the results of FaSS-MVS with $\text{SGM}^{\Pi\text{-pg}}$ optimization are listed.

Dataset	Approach	L1-rel	$\text{Cpl}_{1.25}$ (in %)	$\text{Cpl}_{1.05}$ (in %)	Density (in %)
TMB	SSDE	0.171	93.5	69.8	100
	FaSS-MVS-$\text{SGM}^{\Pi\text{-pg}}$	0.037	72.0	66.4	70.4
GTA	SSDE	0.081	91.0	57.8	100
	FaSS-MVS-$\text{SGM}^{\Pi\text{-pg}}$	0.045	91.4	76.1	93.8

particular, the L1-rel error of FaSS-MVS is significantly lower than that of the SSDE approach. In terms of completeness, however, the SSDE approach clearly outperforms FaSS-MVS on the TMB dataset. This is due to the lower density of the depth maps produced by FaSS-MVS (cf. Table 5.2, Column 6), as the post-filtering removes quite a number of inconsistent pixels. However, the smaller difference between $\text{Cpl}_{1.25}$ and $\text{Cpl}_{1.05}$ of FaSS-MVS compared to the SSDE approach shows that the depth maps of FaSS-MVS are generally more accurate. Nonetheless, the use of only one view for depth estimation has a number of advantages as further discussed in Section 5.7.1.

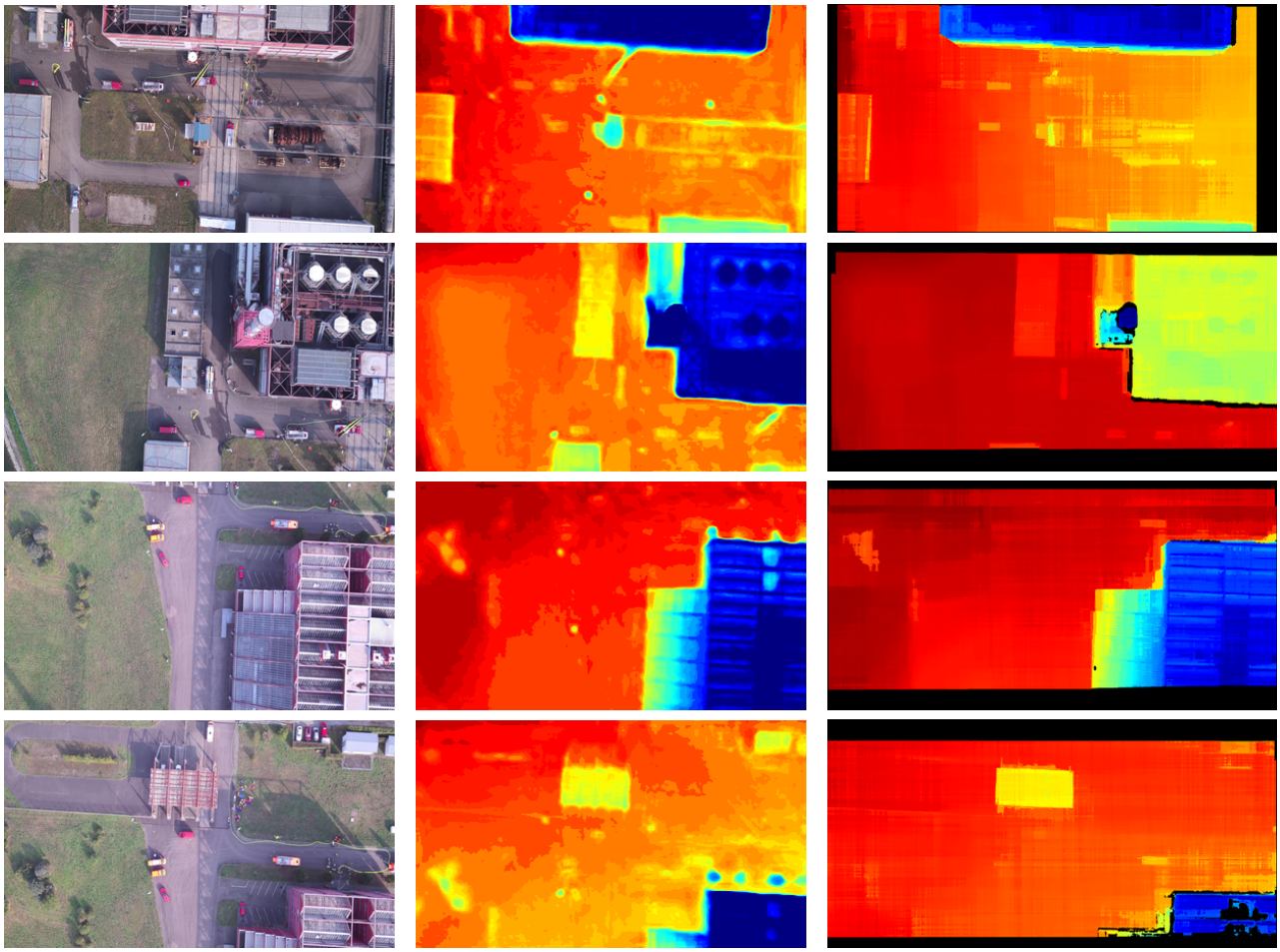


Figure 5.6: Qualitative results of the approach for SSDE on the use-case-specific FB dataset together with depth maps of FaSS-MVS as comparison **Column 1:** Reference image. **Column 2:** Depth maps predicted by the SSDE approach. **Column 3:** Depth maps predicted by FaSS-MVS. The depth is color-coded, going from blue (near) via yellow to red (far). Since the scaling between the predicted depth maps and the reference differs, the color gradients between the two modalities do not coincide.

For a qualitative comparison, an excerpt of the predicted depth maps from the two aerial datasets is shown in Figure 5.5 and Figure 5.6 together with the input images and reference depth maps. As before, the depth is color-coded, going from blue (near) via yellow to red (far). Since the scaling between the predicted depth maps and the reference differs, the color gradients between the two modalities do not coincide.

5.6.4 Ablation Study

In the following, the results of an ablative study with respect to different aspects of the presented approach for self-supervised single-view depth estimation are presented and discussed. Here, the study is focused on the use of different encoder topologies (Section 5.6.4.1) within the U-Net for depth estimation as well as the use of different batch sizes (Section 5.6.4.2), resolution of the input images (Section 5.6.4.3) and input bundle size (Section 5.6.4.4) during training.

5.6.4.1 Encoder Topologies

In the scope of this work, the impact of three different CNN topologies for the use as encoder within the U-Net is investigated, namely the VGG-Net16 (Simonyan and Zisserman 2015), the ResNet18 (He et al. 2016) and the DenseNet (Huang et al. 2017). Among others, the most important requirements for a CNN topology to be used as an encoder within the presented approach is the possibility to extract four layers of the CNN to facilitate the skip connections between the encoder and the decoder, as well as the capability of an end-to-end training. Due to the different configurations and the training speed of the different CNNs, all topologies are trained until the error curve during training flattens, in order to allow a fair comparison. This results in a long training time, which is why the input resolution is reduced to 192×96 pixels in the scope of this study.

The results in Table 5.3 show that all three considered architectures are able to predict meaningful depth maps on the TMB dataset, with the ResNet18 achieving the best results. At the same time, the ResNet18 has the lowest number of parameters, resulting in a faster training, which it why it is chosen for the presented SSDE approach. Deeper variants of the ResNet, i.e. ResNet32 and ResNet50, were also investigated. However, they have a much greater number of parameters that are to be trained, requiring a reduction in the batch size, which, in turn, led to a more unstable training.

Table 5.3: Comparison of different CNN architectures for the use as encoder within the U-Net. Results are computed on the TMB dataset with a reduced image size of 192×96 pixels.

Architecture	L1-rel	$\text{Cpl}_{1.25}$ (in %)
VGG-Net16 (Simonyan and Zisserman 2015)	0.133	<u>92.7</u>
ResNet18 (He et al. 2016)	<u>0.120</u>	92.6
DenseNet (Huang et al. 2017)	0.184	91.2

5.6.4.2 Batch Size

On the street-level imagery from the KITTI benchmark a batch size of 4 is chosen, similar as by Godard et al. (2019). For the final training configuration on the aerial image datasets, however, the batch size is empirically set to 20 (cf. Section 5.5), due to an unstable training when smaller batch sizes are used. The instability of the training leads to a divergence of the training in early iterations, as the optimization gets stuck in a local minimum. This divergence is manifested by the network predicting a depth of 0 for a significant amount

of pixels in the depth maps, from which it does not recover. This is presumably due to the inability of the pose estimation network to reliably predict the relative transformations if the batch size is too small. This assumption is backed by the fact, that sequences with complex camera movements, such as a high amount of rotation without significant translational movement, are particularly affected. A possible explanation might lie in the sharing of the encoder between the depth and pose estimation network, since both are dependent on each other, which could possibly create a dead lock in which the prediction of the depth maps degenerates and the training gets stuck. With an increase in batch size, the possibility of processing a higher variety of camera movement within one batch also rises, preventing a divergence in early iterations. However, once reasonable depth maps are predicted by the network, the batch size can be reduced. Further experiments have shown, that a batch size larger than 20 does not achieve better results. It rather restricts the resolution that can be used for the input images, since it increased the memory consumption which is limited in terms of the upper-bound by the available memory on the GPU.

5.6.4.3 Image Size

The quality of the resulting depth maps is greatly influenced by the image size used as input. The image size, on the other hand, is limited by the memory available on the GPU, depending on what batch size is used. While the results listed in Table 5.2 are achieved using an input resolution of 384×224 pixels, an ablative study on the TMB dataset with respect to different sizes of the input images has shown, that even a small resolution of 192×96 pixels achieves reasonable results. In this, it is also found that the time required to train a network is directly proportional to the image size used. Training the network with an image size of 192×96 pixels instead of 384×224 pixels is $4\times$ faster, while at the same time reducing the quality of the quantitative results by only 2 %. A use of even larger image sizes, i.e. 768×448 pixels, however, is restricted by the limited memory of the GPU. To overcome this restriction, the network is pre-trained on a smaller image size, which is followed by a training using a higher image resolution but a smaller batch size. This, however, does not lead to superior results, while at the same time greatly increasing the training time.

5.6.4.4 Size of the Input Bundle

In a final ablative study, it was investigated, whether the approach benefits from using more than three input images during training. In this, five input images, with two images to either side of the reference image are used. The study, however, shows that this does not lead to any superior results, both quantitatively and qualitatively. It is assumed that, with more matching images being projected into the reference view, the probability of false positives within the process of image sampling and image matching is increased, due to a higher probability of pixels that presumably have a high similarity but are sampled from wrong locations in the image. This, in turn, hinders the training process.

5.7 Discussion

Before providing some concluding remarks, three different aspects of the presented approach for the task of SSDE are discussed in the following. First, the overall results are put into context and discussed in Section 5.7.1 with respect to a practical use of the approach. This is followed by a discussion on the run-time of the approach in Section 5.7.2. Finally, the generalization and self-improving capabilities are discussed in Section 5.7.3, which is very important in order to conclude whether the approach is actually ready to be used in practice.

5.7.1 Overall Results

Overall, the results presented in Table 5.2 as well as in Figure 5.5 show that the presented approach for SSDE is well capable of learning how to predict a depth map from a single aerial image. Both quantitative and qualitative results show that the quality of the resulting depth maps is high with respect to the reference data. Moreover, the comparisons in Table 5.1 and Table 5.2 show that it is also capable to keep up with conventional approaches that estimate depth maps from two or more images as well as related work from literature. Presumably due to the low image resolution with which the approach is trained and which is also used for inference, the model is not able to reconstruct fine structures like vegetation and sharp edges. However, the essential geometry of the scene and objects is predicted correctly.

Furthermore, as already stated, approaches for self-supervised single-view depth estimation have a number of advantages over methods that rely on conventional dense image matching. For example, since SSDE approaches only rely on one input image, they are not sensitive to camera movement at the time of inference and, thus, do not suffer from instabilities, if the camera movement degenerates. And during training, in which the approach still greatly depends on a good choice of input images, the setup of the input data can be controlled more easily, at least if the model is not to be retrained on-the-fly. Another advantage is that approaches for SSDE produce fully dense depth maps, which typically also contain estimates in areas in which conventional DIM approaches often fail due to a weak texture or occlusions. Lastly, since SSDE approaches use only one input image to predict a depth map, they can usually operate at a higher frequency compared to monocular MVS approaches, which first have to wait until an appropriate set of input images is acquired.

Nonetheless, there is a significant drawback of the presented approach that is to be discussed when looking at the overall results. The training of the approach is done on a great number of disjoint image bundles consisting of one reference image and one or more matching images to either side of the reference image. To this end, when these bundles are constructed, there is no consistency with respect to the baseline between the individual images or a temporal coherency so that the CNN is actually able to learn a depth scale which is consistent over all predicted depth maps. In fact, the resulting CNN predicts a different depth scale for every depth map, making their use in subsequent steps, like obstacle detection and collision avoidance or 3D mapping, difficult.

5.7.2 Run-Time

A key characteristic of the presented approach for self-supervised single-view depth estimation, and with it its greatest strength, is its capability to predict a depth map from only one aerial image. This allows the SSDE approach to operate at a much higher frequency, since there is no need to wait until a bundle of two or more images with an appropriate baseline is acquired. This, in turn, also results in a significantly lower latency depending on the time it takes to predict a single depth map. As noted in Section 5.5, the approach reaches 20 FPS on a NVIDIA Titan X during inference. In this, the inference is done using TensorFlow and without any optimization regarding run-time. In order to reason about the actual processing rate during a productive use

Table 5.4: Average run-time of the SSDE approach to predict a depth map from a single aerial image of different image sizes executed on different hardware architectures.

Hardware	Embedded SOC	Image Size	
		384×224 pixels	768×448 pixels
NVIDIA RTX 2070 Super		~ 4 ms	~ 12 ms
NVIDIA Jetson AGX	✓	~ 16 ms	~ 45 ms

of the presented approach, a more optimized processing pipeline was implemented using OpenCV, which only covers the prediction of a depth map from a single aerial image, given a trained model. Table 5.4 states the average run-time required by the more optimized processing pipeline to predict a single depth map on different hardware architectures, i.e. a desktop GPU in form of a NVIDIA RTX 2070 Super and the embedded Tegra GPU deployed on the NVIDIA Jetson AGX. The measurements clearly show that the processing is capable of real-time and low-latency processing, reaching theoretical frame rates of up to 250 FPS and 62 FPS on the desktop and embedded hardware, respectively. Note that, even though the model is trained with an image size of 384×224 pixels, it is possible to process images with a larger image size as the convolution kernel of the CNN is iteratively moved over the input image. However, the greater the difference in image size between the training data and inference data, the less accurate the results are. The suitability of the SSDE approach for real-time and low-latency processing suggests that it is a good alternative to conventional two-view stereo approaches for the task of obstacle detection and collision avoidance. However, as further discussed in Section 6.1, the possibly insecure handling of unknown data by learning-based approaches impedes their use for safety-critical applications.

5.7.3 Generalization and Self-Improving Capabilities

An important criterion of deep-learning-based approaches with respect to the use and deployment in an actual application is their generalization capabilities, meaning how well they can cope with unknown data. In the evaluation of the generalization capabilities of the presented SSDE approach, a test setup with two disjoint video sequences (Sequence A & B) is used. The two sequences do not have any overlap or common objects but were captured under similar conditions with respect to camera angle, weather and daytime. In the conducted experiment, the CNN is first trained until convergence on Sequence A and then evaluated on Sequence B. As depicted in Figure 5.7a at epoch 0, when switching to a new, yet similar sequence, the completeness $\mathrm{Cpl}_{1.25}$ drops by around 25 %. This suggests that the trained model generalizes rather poorly to unknown data and, in turn, makes the approach rather impractical for a use in actual applications. In contrast, however, the presented approach for self-supervised training does not require any special reference data during training, which makes an on-the-fly fine-tuning with respect to a new sequence possible, resulting in so-called *self-improving capabilities*.

To evaluate how fast a pre-trained model can be fine-tuned to a new sequence, an experiment was conducted in which the pose estimation network was not removed and the weights of the model were not fixed after initial training on Sequence A, allowing to continue training on Sequence B. As illustrated by the green curve in Figure 5.7a, it takes around 2-4 epochs to fine-tune a pre-trained model to a new sequence, reaching the initial completeness again. With the configuration described in Section 5.5, one training epoch takes around 2 minutes, making the fine-tuning around 10× faster than the initial training from scratch. This obviously, however, greatly depends on the similarity between the different sequences. The fine-tuning on a sequence, which greatly differs from the initial sequence in terms of content and environment, is most likely to take longer.

The curves in Figure 5.7a also suggest, that a network that is fine-tuned to a new sequence, tends to forget the initial sequence, as with each epoch the completeness achieved on Sequence A drops. Thus, in a further experiment, a fine-tuning with images from both sequences was conducted, showing that the completeness on Sequence A could be maintained while at the same time causing a similar increase in completeness on Sequence B (cf. Figure 5.7b). Since, in this case, the training dataset is bigger, the time it takes to train for one epoch is, however, also increased. Thus, depending on the application, it has to be considered whether a fast fine-tuning to the current surrounding is more important, considering that the data used for initial training is forgotten.

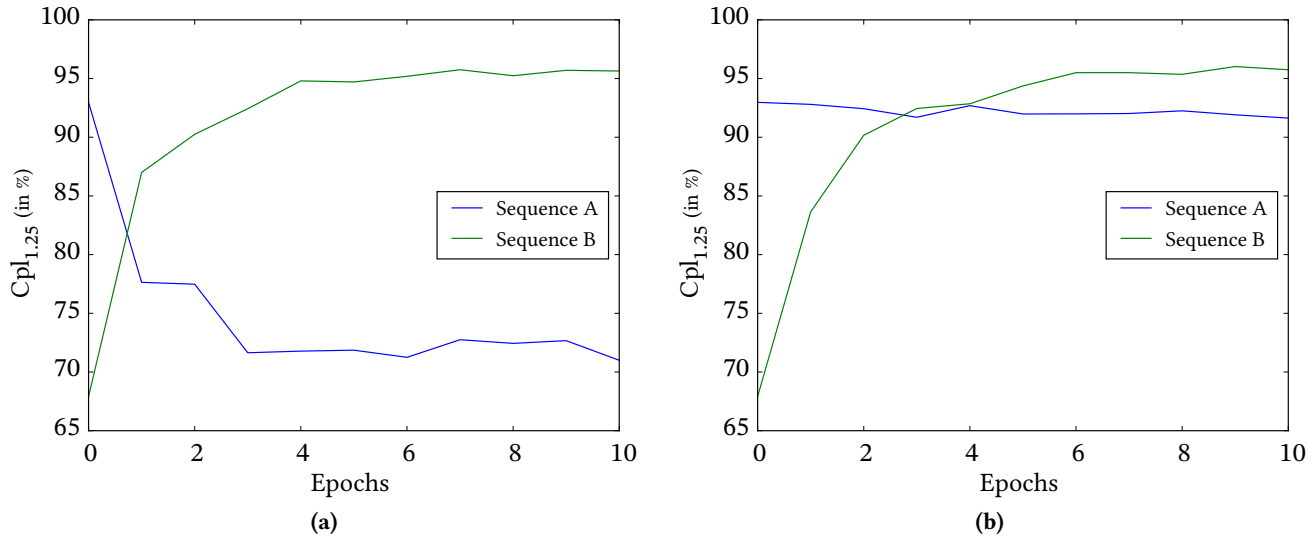


Figure 5.7: Experimental results on the generalization and self-improving capabilities of the presented approach for the task of self-supervised single-view depth estimation. In this, the CNN is first trained until convergence on Sequence A and then evaluated and fine-tuned on Sequence B, which is similar to Sequence A but was not used during the initial training of the CNN. **(a)** Completeness achieved on Sequence A and Sequence B after x epochs of fine-tuning only on Sequence B (Hermann et al. 2020, Fig. 5). **(b)** Completeness achieved on Sequence A and Sequence B after x epochs of fine-tuning on a mixture of Sequence A and Sequence B.

To sum up, due to the fact that a self-supervised training does not require special training data, the presented approach for the task of self-supervised single-view depth estimation can relatively quickly be fine-tuned on new image data, if it is pre-trained on an appropriate dataset. Here it is important, that the dataset that is used for the initial training is kind of similar to the environment in which the approach is later to be used. The method of self-improvement, however, also has some disadvantages. When the CNN is to adapt itself to the current environment, the pose estimation network cannot be omitted and is needed in addition to the network responsible for depth estimation. This results in a larger model. Moreover, more than one input image is needed to estimate the depth. And since not only the forward pass through the network but also the back-propagation of the training loss, and with it the adjustments of the weights, is required in the processing, the run-time is increased.

5.8 Conclusion

In conclusion, this chapter presents an approach that allows to flexibly train a convolutional neural network for the task of SDE from aerial imagery in a self-supervised manner. The use of self-supervised training eliminates the need for appropriate reference depth maps which, in the case of aerial imagery, are especially cumbersome and costly to acquire. In contrast, the approach can be trained on any image data from a video sequence, depicting a static scene, captured by a single moving camera. By formulating the training task as novel view synthesis and image reconstruction process, the network learns how to predict a depth map from a single input image as well as the relative transformations between the input images, as an image can only be correctly synthesized if the scene geometry and the camera setup are learned correctly. While the approach requires three or more input images during training, it only requires a single input image for the depth estimation during inference, allowing it to also be used for the task of reconstructing scenes with dynamic objects. For the task of depth estimation from a single image, only a portion of the CNN is required, allowing to reduce the size of the model and with it the processing time.

The experimental results suggest that the approach can possibly be used to facilitate a number of applications, such as real-time 3D mapping, autonomous navigation or scene understanding. Although the quality of depth maps obtained from a single image are unsurprisingly inferior to those obtained by means of DIM, and although the depth maps produced by the presented approach lack a consistent and metric scale, depth estimation from a single image can operate with a higher frequency than approaches based on MVS. Moreover, SDE does not suffer from typical drawbacks, such as occlusions, ambiguous matching in weakly-textured areas and inability to reconstruct dynamic objects. In comparison to multi-camera DIM approaches, which overcome the inability to reconstruct dynamic objects by a synchronized image capture, the depth range supported by SDE approaches is not limited by the confined baseline between the cameras, as demonstrated in Figure 5.8. Thus, self-supervised single-view depth estimation approaches are well-suited to complement, and in some applications even substitute, depth estimation methods relying on DIM and MVS.

A practical use of deep-learning-based methods, however, is typically hindered by uncertainties with respect to their operation on unknown image data and generalization. While experimental results show, that a pre-trained model generalizes rather poorly to a new and yet still similar sequence, resulting in a drop in completeness by about 25 %, it is also demonstrated that due to the fact, that the approach does not require any special training data, it can quickly be adapted and fine-tuned to the new sequence, reaching the initial completeness after only few training epochs. To this end, the self-improving capabilities have only been demonstrated using offline learning. In future work, an investigation with respect to on-the-fly learning, meaning that the model is retrained during the actual application, is still to be done. Furthermore, investigations on the generalization capability to a change in the scene for which the CNN has been trained, e.g. other weather or lighting conditions or even modifications to scene geometry, are to be conducted. This is particularly important for applications which repeatedly visit the same scene, such as 3D change detection or monitoring.

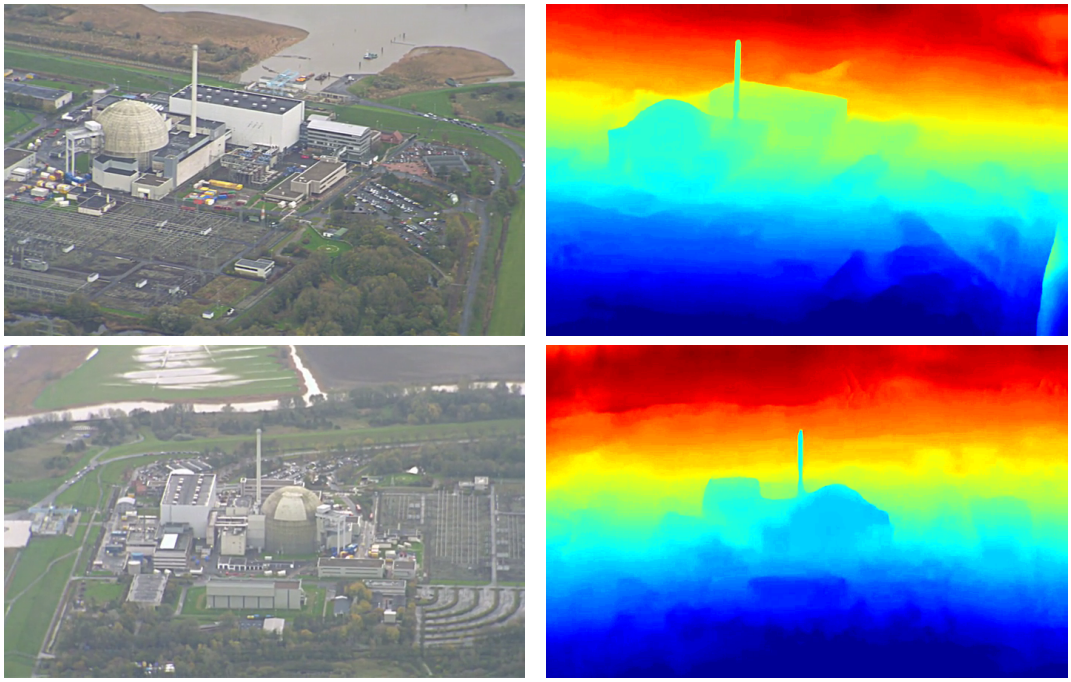


Figure 5.8: Exemplary results of the SSDE approach to estimate depth maps from images with a very large focal length. **Column 1:** Reference image. **Column 2:** Predicted depth maps. The depth is color-coded, going from blue (near) via yellow to red (far).

Lastly, it is very important that the depth maps obtained from single-view depth estimation are to be enhanced with a metric scale in order to facilitate applications that rely on accurate distance measurements, such as autonomous driving and navigation. Solutions to this are manifold. For one, the training data could be enhanced with a few metric depth maps in order to regularize the training. Another possibility is to remove the pose estimation network and use a calibrated pose estimation approach, such as visual inertial SLAM (Campos et al. 2021), which would also remedy the drawback of unstable training in case of complex camera movement. The integration of an external method, however, would reduce the benefit of an end-to-end training.

6 Discussion on the Importance of the Presented Approaches for Possible High-Level Applications

The presented approaches for fast and efficient dense depth estimation based on two-view stereo (Chapter 3) and multi-view stereo (Chapter 4) as well as estimating depth from a single view by means of deep learning (Chapter 5) are embedded into a high-level framework (Section 1.2), which aims to facilitate emergency forces, in particular first responders, in case of disaster relief by aerial reconnaissance using commercial off-the-shelf (COTS) UAVs. In this, the focus lies on the facilitation of three high-level applications, namely real-time obstacle detection and collision avoidance (Section 6.1) for the safe and autonomous use of the UAVs, as well as fast and incremental 3D mapping (Section 6.2) and 3D change detection (Section 6.3) for the assessment of the disaster site. Details on how the presented approaches for dense depth estimation aid the execution of these high-level applications are presented and discussed in the following sections.

6.1 Reactive Obstacle Detection and Collision Avoidance for UAVs Based on Dense Disparity Maps

As illustrated by Figure 1.1, modern COTS UAVs are equipped with a number of sensors to monitor the close vicinity and flight path of the UAV, allowing to detect obstacles and avoid collisions which, in turn, is crucial for a safe and autonomous flight. This is particularly important, if the UAV is to be flown beyond visual line-of-sight (BVLOS). In addition, it can further increase the benefit of the UAV, as the operating personnel can concentrate on the data produced by the system rather than flying the UAV itself. For the task of monitoring the close surrounding of the UAV, typically ultrasonic as well as visual sensors are used, due to their low weight, cost and power consumption. While the strength of ultrasonic sensors lies in the reliable detection of obstacles in a very close vicinity, i.e. a few tens of centimeters, visual stereo cameras allow to monitor the flight path up to a few tens of meters in front of the UAV. To detect obstacles and calculate an appropriate evasion maneuver based on data from a stereo vision sensor, as illustrated by Figure 6.1, first a dense disparity or depth map is to be calculated from which the obstacles can be detected and a flight path around them can be calculated. In this, it is of great importance that the whole processing pipeline runs in real-time and with

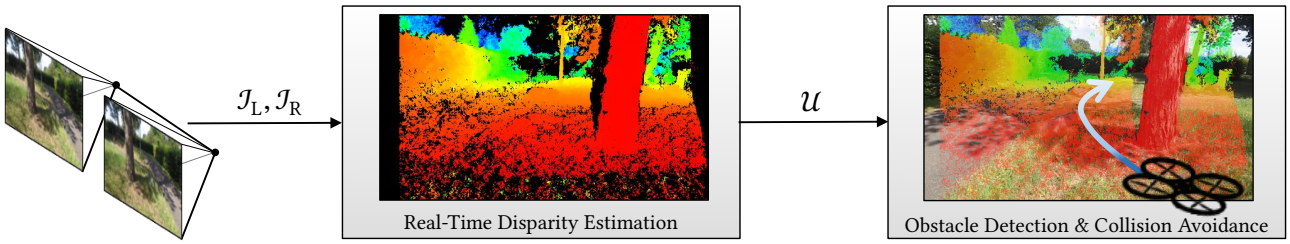


Figure 6.1: Overview of the discussed approach for reactive obstacle detection and collision avoidance based on dense disparity maps. Given an image pair from a stereo camera, $\mathcal{I}_L$ and $\mathcal{I}_R$, a disparity map $\mathcal{U}$ is calculated in the first step, for example by using ReS^{2}tAC (Chapter 3). In a subsequent step, the disparity map is used to detect obstacles in the flight path and to calculate an appropriate evasion maneuver. (Ruf et al. 2018b, Fig. 1.)

low latency to ensure a quick and efficient reaction, thus requiring an execution on an on-board processing unit. With ReS^{2}tAC (Chapter 3), an approach for real-time disparity estimation by means of two-view stereo on an embedded ARM and CUDA device, such as the DJI Manifold 2-G, is presented and can, in turn, be used as the first stage within the processing pipeline illustrated in Figure 6.1. To demonstrate the suitability of the disparity maps estimated by ReS^{2}tAC for the task of detecting obstacles and calculating a collision avoidance maneuver, a simple and yet effective approach is implemented for the second stage of the pipeline, which has also been published in the paper of Ruf et al. (2018b).

In contrast to using the disparity maps to incrementally construct a representative map of the environment in which the UAV is operated, allowing for an autonomous exploration of unknown areas, the considered approach only operates on a single disparity map to calculate the free space in front of the stereo camera and allow for reactive collision avoidance. It relies on the transformation of the disparity map $\mathcal{U}$ into more simpler representations, the so-called *U-* and *V-Maps*. They can be interpreted as a top-down- and side-view of the area covered by the disparity map and are often used for the task of reactive obstacle detection and collision avoidance, e.g. in the work of Labayrade et al. (2002), Li and Ruichek (2014), and Oleynikova et al. (2015). In the

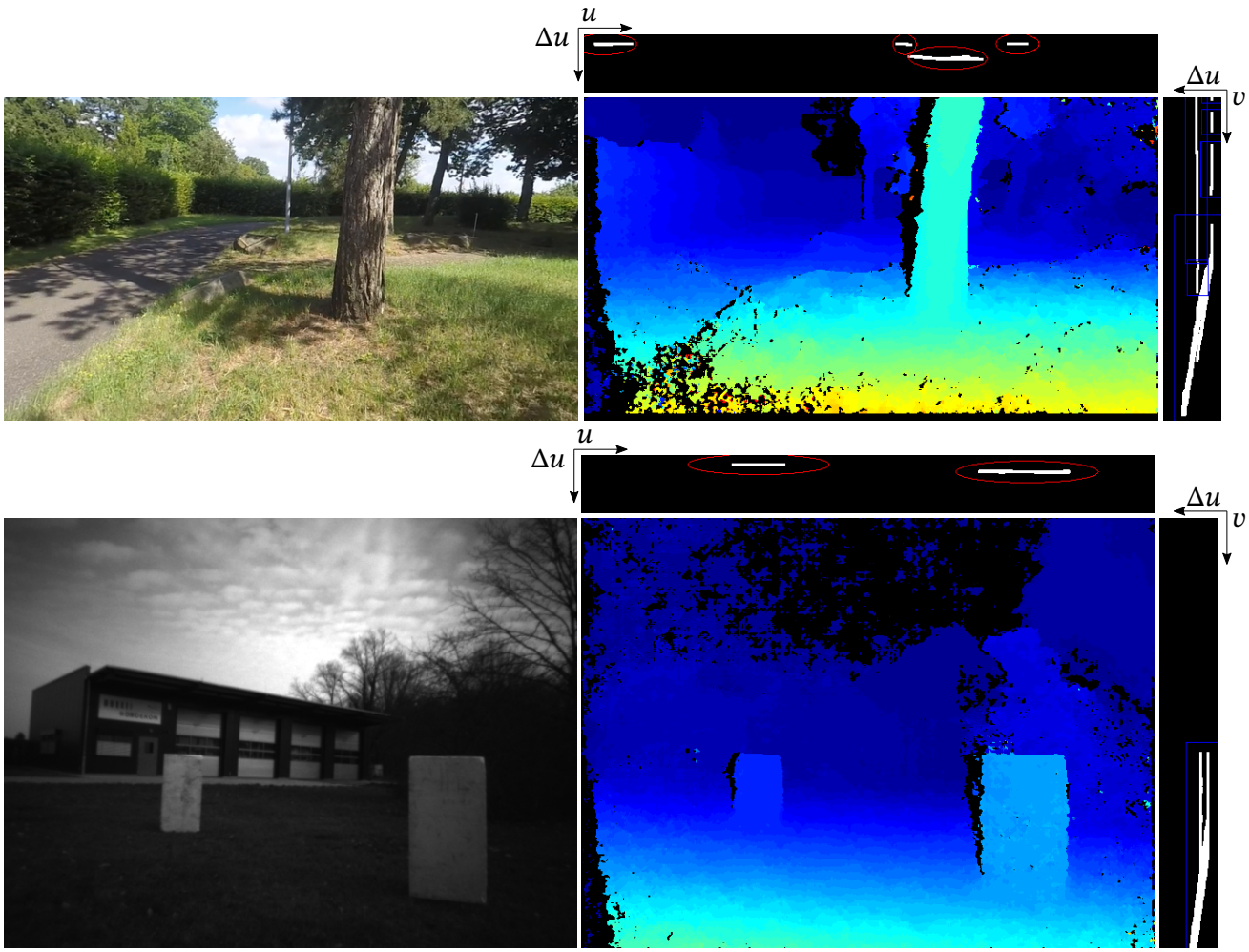


Figure 6.2: Qualitative results of the U- and V-Maps computed for reactive obstacle detection and collision avoidance based on the disparity maps of ReS^{2}tAC. The upper example depicts a scene captured by a hand-held stereo camera, while the bottom example is taken from the stereo data captured by the DJI Matrice 210v2 RTK. The images on the left represent the corresponding reference image. The images on the right are the disparity maps computed by ReS^{2}tAC, together with the U-Map on top and the V-Map to the right. The disparity in the disparity maps is color-coded, going from yellow (high disparity) via green to blue (low disparity). The detected obstacles are marked by red ellipses in the U-Maps and with blue rectangles in the V-Maps. The slanted line in the V-Maps represents the ground plane.

construction of the U-Map, a histogram of disparity occurrences is calculated for every column of $\mathcal{U}$ and stored inside a map of size $W \times |\Delta u|$. Here, W represents the width of the disparity map and $|\Delta u| = \Delta u_{\max} - \Delta u_{\min}$ denotes the number of disparities inside the disparity range with which $\mathcal{U}$ is calculated. The V-Map is calculated analogously by computing a histogram for every row of $\mathcal{U}$ and storing these histograms inside a map of size $|\Delta u| \times H$. Here, $|\Delta u|$ again denotes the number of disparities and H represents the height of the disparity map. For every possible obstacle, the corresponding disparities will accumulate in the histograms. Thus, the U-Map encodes the depth and width of each object, while the V-Map shows the height of each object as well as the ground plane. To further increase robustness with respect to small outliers and suppress uncertainties, the U- and V-Maps are binarized by applying a threshold filter. Further processing by means of dilation as well as cutting off small disparity values to remove the cluttered background reveals prominent objects in the foreground. Here, a kernel of size 7×3 and 3×7 is used for the dilation of the U- and V-Map, respectively. Next, the approach of Suzuki and Abe (1985), which is implemented in OpenCV, is used to find the contours of each object from which cylindrical models are abstracted by drawing ellipses around the contours in the U-Map and rectangles around the contours in the V-Map. From these cylindrical models, the depth of the obstacles as well as the closest waypoint to the left, right, top or bottom can be calculated.

Two exemplary results of the U-/V-Map calculation are illustrated in Figure 6.2. While the data in the upper example was captured by a hand-held stereo camera, the data of the bottom example is taken from the stereo data captured by the DJI Matrice 210v2 RTK. The qualitative examples show, that the disparity maps produced by ReS^{2}tAC are well-suited for a reactive obstacle detection based on the computation of U- and V-Maps, as the obstacles are clearly marked by the red ellipses in the U-Maps and the blue rectangles in the V-Maps. In both examples, the ground plane is clearly revealed by the slanted line inside the V-Map. In the bottom example, the V-Map clearly shows that it is also an option to fly over the detected obstacles as the vertical line only goes up to half the height of the disparity map. As discussed by Ruf et al. (2018b), the integrity of the algorithm to evade the obstacles and avoid collisions was validated by means of hardware-in-the-loop testing in combination with the flight simulators of DJI and Microsoft AirSim (Shah et al. 2017). In this, the flight controller of the UAV is connected via USB to the PC which is running the simulator allowing the flight controller to control the simulation instead of controlling the aircraft itself.

As previously discussed, safety-critical applications, such as obstacle detection and collision avoidance, typically have strict run-time requirements, in the sense that they have to run in real-time and with low latency in order to allow a quick reaction. Figure 3.15 illustrates that ReS^{2}tAC is well capable of reaching these requirements. It achieves frame rates of over 30 FPS on the embedded GPU of the NVIDIA Jetson AGX when processing stereo images of a size of 640×480 pixels and 64 disparities, which is also the case for the examples depicted Figure 6.2. To this end, the considered approach for obstacle detection and collision avoidance has not been optimized for parallel execution. Instead, it is executed on the CPU of the NVIDIA Jetson AGX, increasing the run-time of the full pipeline by approximately 10 %, which results in a loss of 2-3 FPS. This is acceptable, especially since large parts of the construction of the U- and V-Maps, such as the computation of the disparity histograms and the subsequent filtering, are well-suited for massively parallel execution by means of GPGPU and can thus be further optimized. The run-times listed in Table 5.4 corresponding to the execution of the third approach that is proposed in this work for the task of depth estimation, namely the approach for single-view depth estimation, show that it achieves similar frame rates as ReS^{2}tAC. However, as already noted in the corresponding discussion, approaches based on deep learning are of limited suitability for safety-critical applications, due to their possibly unreliable handling of unknown data. Nonetheless, the research field of explainable AI has grown significantly in the last years (Barredo Arrieta et al. 2020), further increasing the reliability of deep-learning-based approaches in safety-critical environments.

6.2 Rapid 3D Mapping Based on Dense Depth Maps

While the first part of the proposed and embedding framework (Section 1.2) focuses on the actual operation of the UAV, the second part aims to facilitate the operator, which in case of the considered use-case are emergency forces, in the rapid assessment of a geographical area by means of 3D mapping. Given the image data from the payload camera, a fast and online 3D mapping is to be performed, meaning that the 3D model is created while the image data is acquired or at least received by the ground control station (GCS). If an appropriate data-link between the UAV and the GCS is established, the processing can be done while the UAV is flying over the area of interest. Since the use-case aims at fast and online processing in order to aid first responders, software toolboxes, such as COLMAP (Schönberger and Frahm 2016, Schönberger et al. 2016), that are aimed for offline 3D reconstruction are not an option for this use-case, due to their higher run-time. Nonetheless, highly accurate 3D models produced by such toolboxes are still helpful for an offline assessment and their generation could be triggered anyway, once all image data is acquired.

Conventional approaches for fast and online 3D reconstruction and model generation based on monocular image data are typically divided into three subsequent steps, namely extrinsic camera pose estimation, dense depth estimation and depth map fusion (cf. Figure 6.3). Given data from an image sequence captured by a single moving camera which has been calibrated a priori, meaning that the intrinsic camera matrix $\mathbf{K}$ is known, the first step performs an estimation of the camera trajectory together with a sparse mapping of the scene by using a visual SLAM (V-SLAM) algorithm. In the second stage, dense depth maps are computed by means of monocular multi-view stereo from a bundle of input images with corresponding camera poses. The estimated depth maps are then incrementally fused into a three-dimensional model in the third stage.

With FaSS-MVS (Chapter 4), this work presents an approach for an efficient processing of the computationally most expensive part of the pipeline, i.e. the dense depth estimation. To demonstrate that FaSS-MVS is capable of facilitating rapid 3D mapping, an approach based on the processing pipeline illustrated by Figure 6.3 was implemented and evaluated. In this, the state-of-the-art and open-source SLAM system *ORB-SLAM2* (Mur-Artal and Tardós 2017) is used for sparse mapping and camera trajectory estimation in the first stage. While ORB-SLAM2 continuously tracks the movement of the camera for each frame of the image sequence, it also defines a subset of these input images as so-called key-frames. These key-frames are used to create a sparse map of the environment by matching and triangulating the ORB features (Rublee et al. 2011) between the key-frames. In doing so, the extrinsic camera poses of the key-frames are simultaneously estimated and updated with every new key-frame that is added to the list. For the approach of fast 3D mapping, the most recent group of five key-frames, consisting of the input images together with the corresponding camera poses, are repeatedly selected and passed on to the second stage. By relying on the internal logic of ORB-SLAM2 to select the key-frames, it is ensured that only images with sufficiently novel content are used for further processing. In the second

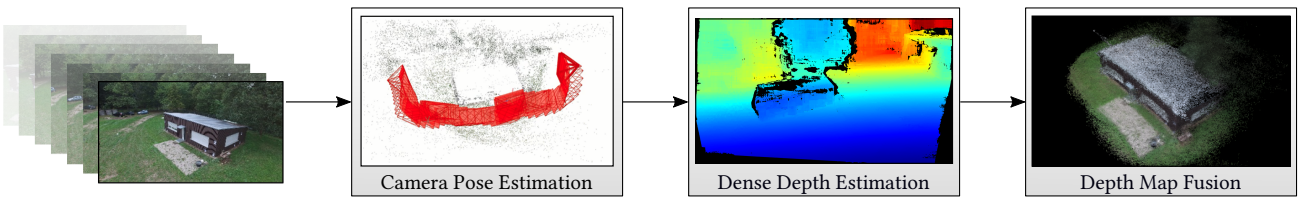


Figure 6.3: Overview of a generic pipeline for online 3D reconstruction and model generation from data of a monocular image sequence. The first stage performs an estimation of the camera trajectory together with a sparse mapping of the scene by using a visual SLAM (V-SLAM) algorithm. In this, it is assumed that the intrinsic matrix of the camera that is capturing the image sequence is known. In the second stage, dense depth maps are computed by means of monocular multi-view stereo from a bundle of input images with corresponding camera poses. The estimated depth maps are then incrementally fused into a three-dimensional model in the third stage.

stage, FaSS-MVS is used to estimate a dense depth map for the middle image of each input bundle, and the resulting depth maps can then be fused into single 3D model in the third stage. For the depth map fusion, the open-source system *ElasticFusion*, presented by Whelan et al. (2015) and Whelan et al. (2016), is used. It relies on both geometric and photometric cues to register and fuse the input data into the already established model and thus incrementally constructs a 3D model of the covered area. While it uses a point-to-plane error measure as a geometric cue, it constructs a photometric cost by projecting the model into the view of the new input frame and matches it with the corresponding input image. Just as FaSS-MVS, ElasticFusion is optimized and accelerated for real-time processing by utilizing general-purpose computation on a GPU (GPGPU). Available geographical information provided by the on-board GNSS of the UAV can either be used to geographically reference the input images and thus propagate the geographical information through the pipeline, or it can be used to calculate an appropriate transformation which is applied on the resulting model in a post-processing. In either way, the geographical referencing of the processed data is vital for the application in focus.

This approach for fast dense 3D mapping from monocular image data captured by COTS UAVs was thoroughly evaluated and published in the work of Hermann et al. (2021a). In the following, the findings presented by Hermann et al. (2021a) will be discussed with respect to the considered use-case of facilitating first responders by providing rapid 3D mapping of the operational area. The evaluation was conducted on a synthetic dataset extracted from the video game Grand Theft Auto V (GTA V) (cf. Section 5.6.1) as well as on the TMB dataset (cf. Section 4.4.1) consisting of real-world data captured by a COTS UAV. The accuracy of the resulting 3D model is calculated by computing the root-mean-square error (RMSE) between the points of the estimated 3D model and the ground truth, with both entities being aligned in such a way that the distance between each point in the estimated 3D model and the closest counterpart in the ground truth is minimal. The qualitative results depicted in Figure 6.4 suggest a high quality of the resulting point clouds. This is supported by the quantitative results listed in Table 6.1, with a RMSE of less than 1 m on both datasets. The RMSE on the GTA dataset is much higher than that achieved on the TMB dataset. However, the fairly high standard deviation of 0.616 m suggests that there are great quality differences between the individual sequences. The qualitative excerpt in Figure 6.4 shows, that even though the considered approach has difficulties in reconstructing fine-grained structures and high level-of-detail, it is well capable to reconstruct larger structures and the general geometry of the scene.

However, the evaluation also reveals that the approach is not capable of successfully processing all input sequences but instead failed on 7.85 % of the synthetic image sequences. The majority of these failures arise from errors in the registration and fusion of depth maps into a single model when covering challenging scenes. Single and isolated misalignments are compensated by the outlier filtering of ElasticFusion, as there will still be enough correctly registered estimates that reduce the confidence score of the outliers. If a number of depth maps are repeatedly and incorrectly registered, however, the resulting model becomes very noisy, making it difficult to fuse subsequent depth maps and, in turn, resulting in a failure of reconstruction. The other cause of failure lies in the camera movement and the flight trajectory of the UAV. If the camera is moved too rapidly, with high amount of rotation, the feature tracking within the SLAM algorithm fails, requiring a re-initialization of the trajectory estimation. While this re-initialization typically does not pose a major problem, it still leads to a stop in the estimation of the depth maps for a certain period of time, making the registration of the new depth maps difficult due to a smaller or missing overlap. This, however, can be compensated if the pose estimation

Table 6.1: Quantitative results of the 3D models computed by the discussed pipeline for fast 3D mapping. The results are computed by evaluation on a synthetic dataset (GTA) and a real-world dataset (TMB). (Hermann et al. 2021a, Tab. 4.)

	GTA	**TMB**
RMSE	0.774	0.141
(in m)	±0.616	±0.033

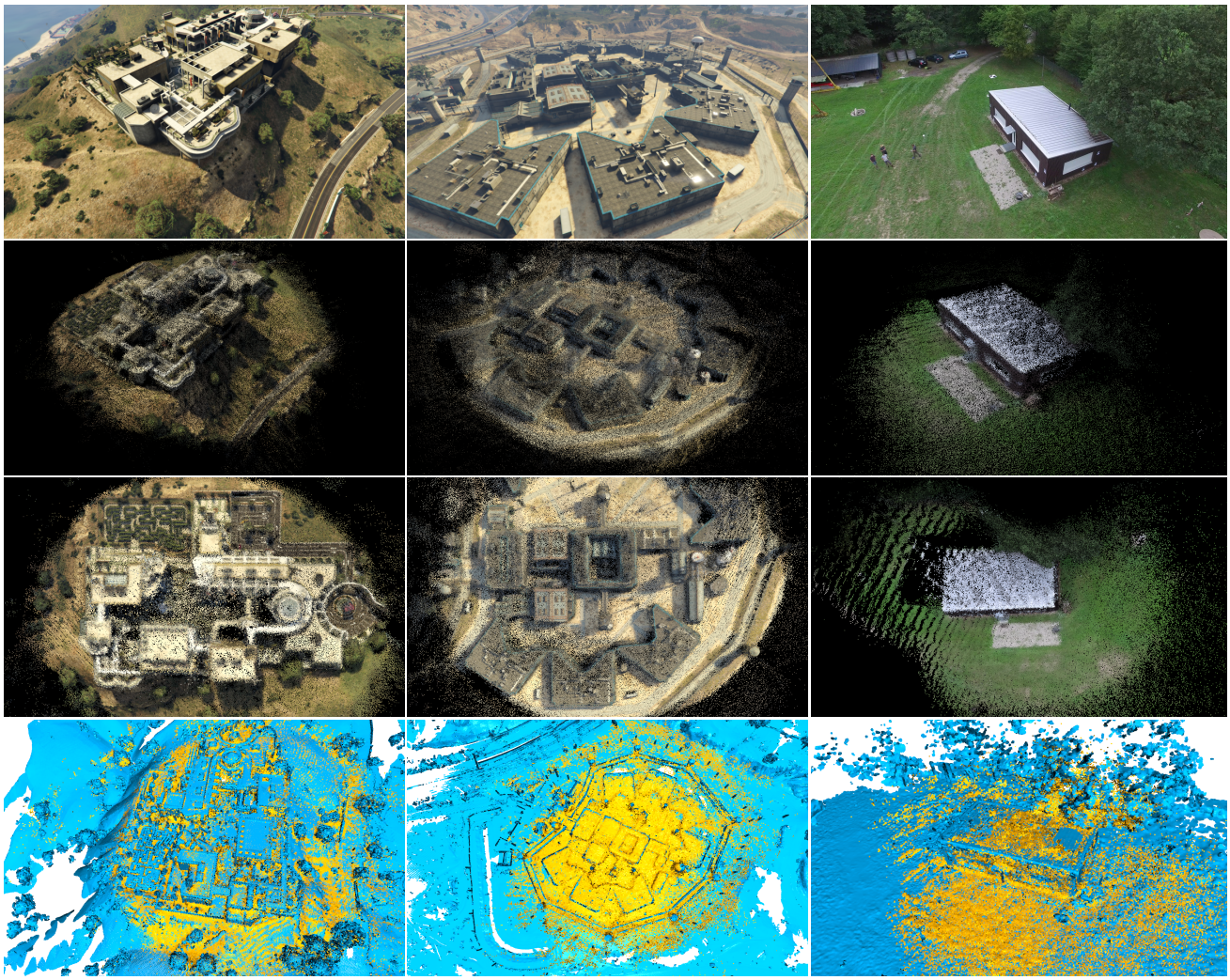


Figure 6.4: Exemplary results of the 3D models computed by the discussed pipeline for online 3D reconstruction. **Row 1:** Exemplary input images from the datasets. **Rows 2 & 3:** Images of the colored point clouds computed from the image sequences corresponding to the images in Row 1. **Row 4:** Visual comparison between the estimated point cloud and the reference model. The reference model is depicted in blue, while the estimated point cloud is depicted in yellow. In some areas, the estimated data lies beneath the reference data and is thus occluded. (Hermann et al. 2021a, Fig. 3.)

is regularized by the augmentation of the input images with data from the GNSS and IMU substituting the visual SLAM system by a visual-inertial SLAM (VI-SLAM) system, such as ORB-SLAM3 (Campos et al. 2021). In addition, if the UAV is not operated manually but instead automatically follows a predefined and specifically planned flight path, rapid camera movements are typically avoided as the flight controller of modern COTS UAVs smoothly updates the camera position based on the predefined configuration.

Despite these weaknesses and instabilities, the biggest strength of the considered approach lies in its run-time. While ORB-SLAM2 and ElasticFusion are capable of running in real-time, with ORB-SLAM2 running on a commodity CPU and ElasticFusion on a GPU, FaSS-MVS by itself is not capable of real-time processing, as already discussed in Section 4.5.3. But since, FaSS-MVS is not executed for every frame of the input stream, the whole pipeline is still capable of keeping up with an input video with 30 FPS, making it capable of online processing. Furthermore, as previously discussed in Section 4.5.3, the flight speed and configuration have a great impact on the processing speed. The experiments conducted by Hermann et al. (2021a) show, that in case

of the GTA dataset, the average subsampling ratio of the considered approach is around 2 %, meaning that for only 2 % of the input images depth maps were computed.

In summary, it has been shown that with a processing pipeline as presented in Figure 6.3, the approach for Fast Multi-View Stereo with Surface-Aware Semi-Global Matching (FaSS-MVS) presented in this work is well capable for rapid 3D mapping of a geographical area by aerial reconnaissance with COTS UAVs. In doing so, the generated 3D map can facilitate emergency forces in a quick assessment of the situation, either by means of providing a large-scale 3D model or by providing an overview with the help of up-to-date orthographic maps, as illustrated in Row 2 of Figure 6.4.

What still needs to be discussed, however, is whether the presented approach for self-supervised single-view depth estimation (SSDE) from aerial imagery can be used within the dense depth estimation stage of the considered pipeline for rapid 3D mapping. The greatest advantage of SSDE over conventional monocular MVS approaches is its high processing speed, the ability to predict depth maps from images containing dynamic objects, and the ability to predict depth from images with a very large focal length. However, its lack of a metric scale, temporal inconsistency, and its insecure handling of unknown objects or scenes as well as the small image size of the resulting depth maps make the presented SSDE approach unfit to be used by itself within the considered pipeline for rapid 3D mapping. The different depth scales and the temporal inconsistency of the depth maps as well as possibly wrong predictions due to a lack of generalization are likely to induce an error in the registration of newly estimated data into the 3D model which, in turn, will lead to a failure in the processes of depth map fusion. The approach of SSDE is rather suited for a complementary use in combination with a monocular MVS approach, such as FaSS-MVS. Due to its high processing speed, SSDE can easily be executed in parallel to FaSS-MVS. Given the depth maps from both approaches, depicting the same scene from the same viewpoint, the multi-view depth map, on the one hand, can then be used to adjust the scale of the single-view depth map. The single-view depth map, in turn, allows to fill in image regions in the multi-view depth map that are missing due to occlusions or due to the existence of dynamic objects.

Lastly, for the task of 3D mapping, image-based techniques can evidently only be used with appropriate visibility and lighting conditions. Thus, the complementary use of active sensing, such as LiDAR, is to be considered in future work, especially since the technological advancements allow to equip more and more COTS UAVs with such sensors.

6.3 Change Detection in 3D Model Data

The third application considered in the proposed framework aims at supporting the emergency forces by means of 3D change detection, which can be of great importance in facilitating damage assessment or an updated route and mission planning for the emergency forces. According to the taxonomy introduced by Qin et al. (2016), 3D change detection methodologies can be divided into two categories, namely “difference-based” and “projection-based” methodologies. As the name suggests, methodologies in the first category used a simple difference metric between two 3D models to detect changes. In this, 2.5D model data, such as depth maps or digital elevation models, as well as complete 3D models can be used. Change detection based on 2.5D data is performed by simply calculating the point-wise difference between the two models along the depth or elevation direction. While this is a very simple methodology that scales well to large-scale models, it is, however, very sensitive to errors in the co-registration of the two models. If the model data is given in the form of complete 3D models, meaning that the models also contain convex structures, change detection based on a simple depth or height difference is not appropriate, since this would not consider structures that are occluded by overhanging

structures. To circumvent this, Qin et al. (2016) propose change detection based on the minimum Euclidean distance between the two models. This approach is more robust to small errors in the co-registration, but at the same time requires the finding of point correspondences between the models.

On the other hand, methods that fall into the second category of the taxonomy introduced by Qin et al. (2016) use data from an existing 3D model, against which the change is to be detected, to project one or more neighboring views into a reference view and match the projected images to the reference image in order to determine inconsistencies. In this, it is assumed that, if a change has occurred with respect to the reference model, the corresponding image regions between the projected and the reference view does not coincide anymore, introducing inconsistencies which, in turn, indicate a change in the scene structure. The advantage with projection-based methodologies is that, in order to perform change detection, no explicit depth estimation or 3D reconstruction is required. According to Qin et al. (2016), these approaches are particularly efficient when the existing 3D model as well as the co-registration between the image data and the model are highly accurate. At the same time, however, such approaches have difficulties when the 3D model has a significantly lower level-of-detail than the image data or if the changes occur in homogeneous image regions.

For the 3D change detection from aerial imagery, this work considers a processing pipeline which is illustrated by Figure 6.5 and was published, together with preliminary results, by Ruf and Schuchert (2016). Again, a very strong focus is put on the run-time of the considered approach in order to achieve online processing. Thus, the change detection is performed only by comparison of two depth maps and thus falls into the first category of the taxonomy introduced by Qin et al. (2016). Here, one depth map, i.e. $\mathcal{D}_\mathrm{I}$, is computed from the image data captured by a UAV flying over an area of interest, while the other one, i.e. $\mathcal{D}_\mathrm{S}$, is synthetically rendered from the reference 3D model against which the change is to be determined. Even though, in the scope of the proposed framework (Section 1.2), it is assumed that $\mathcal{D}_\mathrm{I}$ is computed either by FaSS-MVS (Chapter 4) or the approach for SSDE (Chapter 5), it can also be computed by an approach, such as ReS^{2}tAC (Chapter 3), that relies on two-view stereo, as the preliminary results in Ruf and Schuchert (2016) demonstrate. The rendering of the synthetic depth map $\mathcal{D}_\mathrm{S}$ can efficiently be implemented by relying on the real-time rendering pipeline of OpenGL. By considering the depth information alone, the methodology does not rely on the comparison of texture information between the reference model and the image data. This allows to even perform change detection with respect to data which has been acquired long time ago and possibly at a different time of the year, or which originates from different sensor modalities. However, this approach requires an accurate localization of the image data with respect to the reference 3D model. Even though the pipeline presented in Figure 6.5

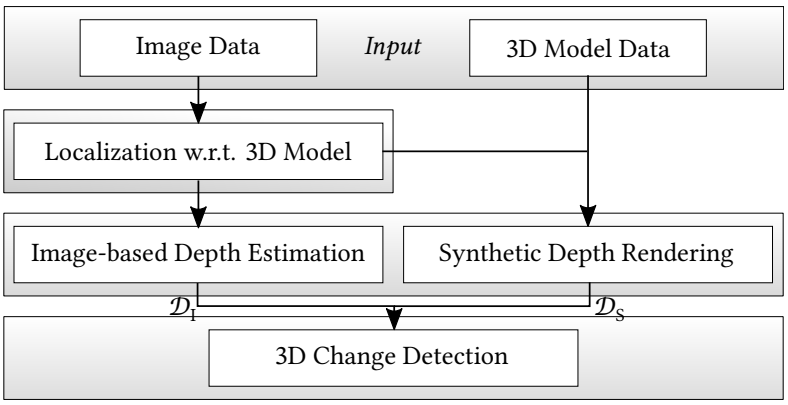


Figure 6.5: Overview of the processing pipeline for rapid 3D change detection based on depth difference. Up-to-date image data is used to compute depth maps ($\mathcal{D}_\mathrm{I}$) of a scene for which the changes are to be determined. These are compared against reference depth maps ($\mathcal{D}_\mathrm{S}$) which are synthetically rendered from a reference model with respect to which the 3D changes are to be detected. For this, it is important that the image data is localized and co-registered with respect to the 3D model data.

suggests that the co-registration between the image and model data is to be performed before the generation of $\mathcal{D}_\mathrm{I}$, it can also be executed after the image-based depth estimation, which, in turn, allows to also consider $\mathcal{D}_\mathrm{I}$ in the registration process. Experiments with respect to both options for the task of automatic co-registration of aerial image data with respect to a 3D model were conducted and published by Schmitz et al. (2019). In both cases, the co-registration is required in order to render the 3D model in such a way that $\mathcal{D}_\mathrm{S}$ depicts the same perspective as $\mathcal{D}_\mathrm{I}$.

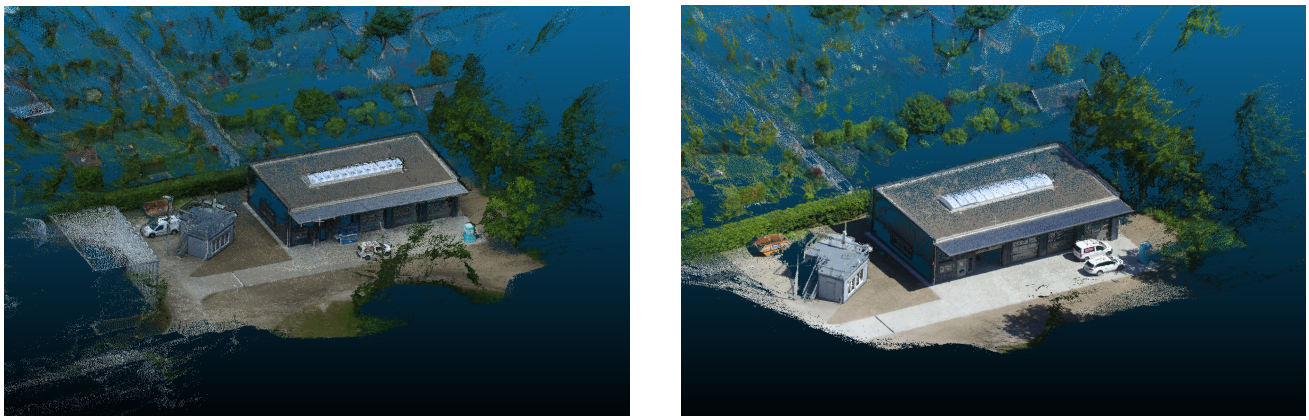
In the process of change detection, a simple difference metric is considered, namely the pixel-wise absolute difference between the logarithmic, normalized depth according to:

$$\mathcal{M}(\mathrm{p}) = \begin{cases} 1 & \text{, if } \left|\log\left(\mathcal{D}'_\mathrm{S}(\mathrm{p})\right) - \log\left(\mathcal{D}'_\mathrm{I}(\mathrm{p})\right)\right| \geq \tau \\ 0 & \text{, otherwise.} \end{cases} \tag{6.1}$$

Here, $\mathcal{M}$ denotes a binary change mask, marking changed image regions with 1, while $\mathcal{D}'_\mathrm{S}$ and $\mathcal{D}'_\mathrm{I}$ represent the normalized depth maps. The threshold, for which the difference between the two depth maps is considered as change, is defined by τ. With the use of the logarithmic depth, variations in regions with a large depth have less influence on the change detection. This is necessary, as the image-based depth estimation typically becomes less accurate with increasing distance. In order to account for noise, the change mask $\mathcal{M}$ is additionally processed by an appearance-based Gaussian smoothing, analogous to that introduced by Equation (4.11). In order to additionally store information about whether a structure has been added or removed with respect to the existing model, Equation (6.1) is extended to:

$$\mathcal{M}(\mathrm{p}) = \begin{cases} 1 & \text{, if } \min\left\{\log\left(\mathcal{D}'_\mathrm{I}(\mathrm{p})\right) - \log\left(\mathcal{D}'_\mathrm{S}(\mathrm{p})\right), 0\right\} \leq -\tau \\ -1 & \text{, if } \min\left\{\log\left(\mathcal{D}'_\mathrm{S}(\mathrm{p})\right) - \log\left(\mathcal{D}'_\mathrm{I}(\mathrm{p})\right), 0\right\} \leq -\tau \\ 0 & \text{, otherwise.} \end{cases} \tag{6.2}$$

By not considering the absolute difference, it is now possible to determine whether a positive or negative change has occurred. Since the logarithm is negative for the range $[0, 1]$, a negative threshold must be used.



(a) Reference Scene **(b)** Matching Scene

Figure 6.6: Illustration of the reference and matching scene used for the qualitative evaluation of the considered approach for rapid 3D change detection. Here, images of an industrial hall were captured on two different days. The visual appearance between the scenes greatly differs, which is due to different weather conditions. Between the two scenes, the cars were differently positioned and the blue lifting platform, visible in in the reference scene, was removed. The co-registration of the data was done manually.

For a qualitative evaluation and test of the considered approach for change detection from aerial imagery, images of an industrial hall were captured by a DJI Phantom 3 Advanced on two different days, as illustrated in Figure 6.6. This figure clearly reveals that the visual appearance between the scenes greatly differs, which is due to different weather conditions. The matching scene is significantly brighter and more colorful than the reference scene. Between the two scenes, the cars were differently positioned and the blue lifting platform, visible in in the reference scene, was removed. The co-registration of the data was done manually using the CloudCompare[1] software.

In Figure 6.7, qualitative results of the considered approach for 3D change detection are illustrated. Here, the image-based depth maps $\mathcal{D}_\mathrm{I}$ were generated based on images from the input sequence. For the reference depth maps $\mathcal{D}_\mathrm{S}$, a reference model was first calculated using COLMAP (Schönberger and Frahm 2016, Schönberger et al. 2016) from which the depth maps were synthetically rendered. Using the formula explained in Equation (6.2), change masks were computed based on $\mathcal{D}_\mathrm{I}$ and $\mathcal{D}_\mathrm{S}$. In the illustration of the depth maps in Figure 6.7, positive changes (structures that are not present in the reference model) are marked with blue and negative changes (structures that have been removed) are marked with red. A qualitative evaluation reveals that the two vehicles in the background are correctly identified and marked as positive changes. In case of negative changes, the change masks show many false positives. In addition, the real changes, such as the removed car or the removed lift, have not been detected. This can be attributed to a possibly different scaling between reference and matching depth maps, which can be deduced by the different coloring between $\mathcal{D}_\mathrm{S}$ and $\mathcal{D}_\mathrm{I}$. As already discussed by Qin et al. (2016), this form of 3D change detection, i.e. based on the point-wise difference along the depth direction, is particularly prone to errors in the co-registration. However, a major advantage of the methodology is that it does not rely on a texture comparison between the reference and matching data. Thus, different weather conditions, times of day, or seasons typically do not have an influence on the change detection.

The run-time of the actual change detection is negligible compared to the image-based depth estimation based on monocular MVS. When using approaches that achieve higher processing rates, such as ReS^{2}tAC or SSDE, the change detection will increase the latency of the complete processing pipeline. However, since it is only a computation based on depth difference, it is not very computationally expensive. As already discussed in the

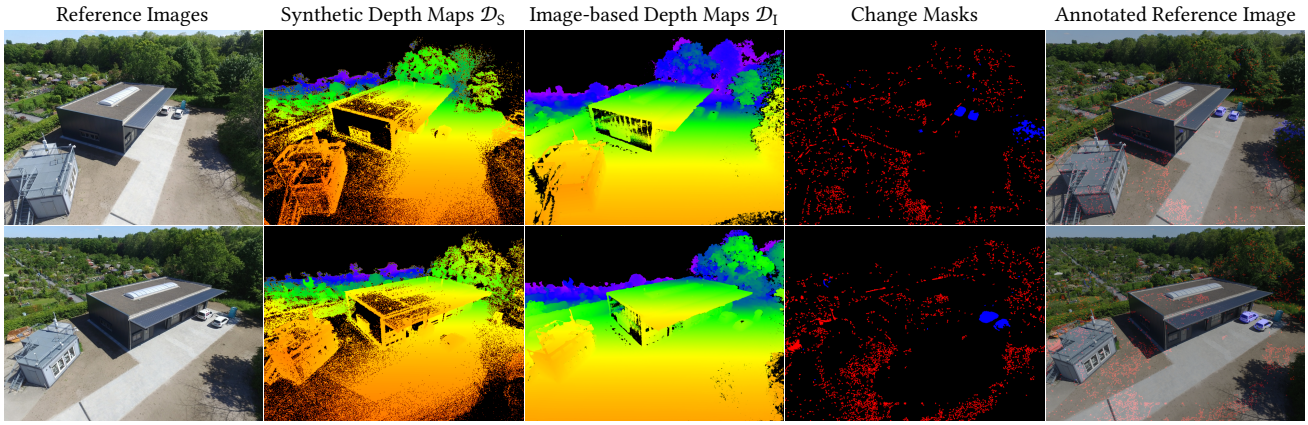


Figure 6.7: Qualitative results of the approach for rapid 3D change detection based on depth difference. **Column 1:** Reference images from the input sequence, in which 3D changes with respect to the reference model are to be detected. **Column 2:** Reference depth maps synthetically rendered from the reference model given the corresponding viewpoint. **Column 3:** Matching depth maps computed from the images of the input sequence. **Column 4:** Computed change masks with blue marking new structures and red marking removed structures. **Column 5:** Reference images overlaid with the corresponding change masks.

[1] http://www.cloudcompare.org/

previous chapter on rapid 3D mapping, the lack of scale and the difficulty in generalization of an approach based on SSDE inhibit its sole use within the presented pipeline. Nonetheless, a complementary use in order to enhance the depth maps computed by conventional methods can be very beneficial.

Lastly, just as in the case of the discussion on obstacle detection and avoidance, this is a very simple approach for 3D change detection that still has quite a number of weaknesses as illustrated in Figure 6.7. It is to be further improved in order to increase its performance, for example by using a more sophisticated difference measure, or by introducing temporal consistency in order to reduce false positives, as well as enhancing it with data driven methods relying on CNNs. Nonetheless, the results demonstrate the feasibility of using approaches for fast image-based depth estimation in order to facilitate rapid 3D change detection, which, in turn, aids emergency forces in the assessment of the situation in case of disaster relief.

7 Conclusions and Future Work

This work addresses the use of commercial off-the-shelf (COTS) UAVs by emergency forces, in particular first responders, to facilitate disaster relief or search-and-rescue (SAR) missions by aerial reconnaissance. It proposes a framework that aims at facilitating the rapid assessment of a disaster site based on aerial image data captured by UAVs, which are ideally operated fully autonomously in conjunction with a ground control station (GCS). This framework is divided into two parts: (i) The first part aims at increasing the safe and autonomous use of the UAV by implementing an approach for real-time and low-latency two-view stereo depth estimation that is executed on a high-performance on-board processing unit (PU). It estimates disparity maps from the on-board stereo vision sensor, which serve as an input to an algorithm for reactive obstacle detection and collision avoidance. (ii) The second part of the proposed framework addresses the use of image data from the visual payload camera to perform fast multi-view stereo (MVS) depth estimation and single-view depth estimation (SDE) to allow rapid 3D mapping and 3D change detection. Here, it is assumed that the image data is streamed down to the GCS, equipped with more powerful hardware in order to allow for high resolution online processing. The approaches for two-view, multi-view and single-view depth estimation are individually and thoroughly evaluated on appropriate datasets, clearly demonstrating their performance. The feasibility of their use for the corresponding applications within the proposed framework is shown and discussed by embedding them into greater processing pipelines and performing experiments on use-case-specific datasets.

With the three different approaches for fast depth estimation, this work presents a number of scientific contributions. The key contributions of this work are summarized as follows:

- **Implementation-specific optimization of the SGM algorithm for real-time two-view stereo on embedded ARM and CUDA devices:** The so-called Semi-Global Matching (SGM) algorithm is one of the most widely used algorithms for embedded and real-time dense depth estimation based on two-view stereo images. This is due to its good trade-off between accuracy and computational complexity. At the same time, when it comes to embedded high-performance computing, the use of FPGA-based systems, for which the development is often cumbersome and costly, was for a long time the only option. In recent years, however, more and more COTS UAVs can be equipped with SOCs that provide embedded GPUs, such as the NVIDIA Jetson series, which, in turn, can be utilized by the CUDA-API for massively parallel processing. With ReS^{2}tAC, this work proposes an approach in which the SGM algorithm has been optimized for embedded CUDA GPUs, such as the NVIDIA Tegra, by utilizing the CUDA-API, as well as for embedded ARM CPUs by utilizing the NEON intrinsics for vectorized SIMD processing. The optimization of the SGM algorithm with NEON intrinsics for SIMD processing on embedded ARM CPUs is the first of its kind. It is shown, that ReS^{2}tAC achieves state-of-the-art accuracies on public stereo benchmarks, while the CUDA-based implementation reaches real-time performance on modern embedded SOCs. Furthermore, it is demonstrated that it can be used for real-time stereo depth estimation on COTS UAVs that are equipped with appropriate hardware. Even though GPU-based SOCs have a worse FPS/W ratio than those that rely on FPGA technology, it is shown that in case of rotor-based UAVs, their power consumption is negligible compared to that of the rotors during flight.

- **A hierarchical approach for fast dense depth estimation based on monocular multi-view stereo with a surface-aware improvement of the SGM algorithm:** For high resolution and fast dense depth estimation that facilitates rapid 3D mapping and 3D change detection, this work proposes the so-called Fast Multi-View Stereo with Surface-Aware Semi-Global Matching (FaSS-MVS) approach. It relies on a hierarchical processing to efficiently perform dense depth estimation by means of monocular MVS from aerial imagery. In the depth estimation, a plane-sweep algorithm for dense multi-image matching and generation of depth hypotheses is used, which is followed by an adapted and improved variant of the SGM algorithm to perform a surface-aware regularization and depth map computation. The surface-aware regularization reduces the fronto-parallel bias of the SGM algorithm and allows to account for slanted surfaces which are particularly prominent in oblique aerial imagery. The quantitative and qualitative evaluation show that the depth maps of FaSS-MVS achieve a high quality with an L1 error of only 1 % of the maximum depth. As the name suggests, the focus of FaSS-MVS lies in a fast and yet accurate computation that allows for online processing, meaning that it should be fast enough to keep up with the frame rate of the input data, while not being subjected to hard run-time or latency constraints as it is not intended for safety-critical applications. Apart from the methodical optimization, FaSS-MVS has also been optimized for massively parallel processing on a GPU, allowing to reach a processing rate of 1-2 Hz, depending on the configuration of the input data. This is enough to keep up with the input data stream, since in case of monocular MVS, the depth estimation is not performed for each input image.
- **Investigation of the use of self-supervised learning for the task of single-view depth estimation from aerial imagery:** A major disadvantage of the approaches presented with ReS^{2}tAC and FaSS-MVS, which rely on dense image matching (DIM), is that they require two or more images for the task of depth estimation. This, however, limits their processing rate as well as their ability to reconstruct dynamic objects, in case of monocular MVS. Furthermore, if depth estimation is performed based on images of a stereo camera, the depth range is limited by the baseline between the cameras. Motivated by the ability of humans to reason about depth from a single input image based on their empirical knowledge, a data-driven approach for single-view depth estimation (SDE) based on CNNs is investigated. In particular, an approach that is trained by means of self-supervision in order to not depend on specific training data, which is typically cumbersome and costly to acquire. By formulating the training task as novel view synthesis and image reconstruction problem, the approach can be trained on any image data from a video sequence that depicts a static scene and is captured by a single moving camera. The independence of specific training data, i.e. reference depth maps, makes the deployment of a self-supervised single-view depth estimation approach more flexible with respect to the practical use. The experimental results suggest that the approach can possibly be used to facilitate a number of applications, such as real-time 3D mapping, autonomous navigation or scene understanding. However, the lack of a metric scale and temporal inconsistency in the resulting depth maps as well as the insecure handling of unknown objects or scenes by learning-based approaches makes the presented SSDE approach rather unfit to be used by itself. It is rather suggested to use the approach for single-view depth estimation complementary to an approach based on DIM, such as ReS^{2}tAC or FaSS-MVS. Due to its high processing speed, SSDE can easily be executed in parallel. Given the depth maps from both approaches depicting the same scene from the same viewpoint, the DIM depth map, on the one hand, can be used to adjust the scale of the single-view depth map, which, in turn, allows to fill image regions that are missing in the DIM depth map due to occlusions or due to dynamic objects.

- **Demonstration of the use of dense depth maps estimated by means of two-view stereo for the task of on-board obstacle detection and collision avoidance on COTS UAVs:** To demonstrate the suitability of ReS^{2}tAC for real-time and low-latency obstacle detection and collision avoidance, it is combined with an approach that transforms the disparity or depth maps into a top-down- and side-view of the area in front of the UAV, represented by the so-called U- and V-Maps. From these U- and V-Maps, cylindrical models of the obstacles can be extracted by subsequent filtering, contour detection, and fitting of ellipses and rectangles. Qualitative experiments show that obstacle detection and collision avoidance based on the U- and V-Map is simplistic and yet very efficient in the detection of obstacles. Its computationally low complexity allows, in conjunction with ReS^{2}tAC, for real-time and low-latency processing on board the UAV. Yet, its lack of temporal consistency makes the approach sensitive to false positive detection and possible noise inside the disparity or depth maps.
- **Demonstration of the use of dense depth maps estimated by means of multi-view stereo for the task of rapid 3D mapping and 3D change detection:** The second part of the proposed framework aims at rapid 3D mapping and 3D change detection based on dense depth maps computed by FaSS-MVS. The separate evaluation of FaSS-MVS has shown that the estimated depth maps have a high quality and that it can reach a processing rate of 1-2 Hz, depending on the configuration of the input data. To demonstrate the use of FaSS-MVS for rapid 3D mapping from aerial imagery, it is integrated into a bigger processing pipeline that relies on open-source real-time visual SLAM system for the estimation of the camera trajectory as well as on an open-source system for real-time depth map fusion and 3D reconstruction. A thorough evaluation, which is published in other work, shows that for a great majority of the investigated image sequences, this pipeline produces accurate 3D models with a mean RMSE of less than 1 m and that it is capable of running simultaneously to the input stream and keeping up with an input frame rate of 30 FPS. For a small percentage of the investigated sequences, however, the considered approach fails due to rapid camera movement with a high degree of rotation, which typically arises if the UAV is operated manually. This leads to errors in the depth estimation which, in turn, result in a misalignment and wrong registration in the fusion of the depth maps. Thus, it is important to make use of an autonomous flight based on predefined waypoints enforcing a smoother camera movement.

  To investigate the ability to use FaSS-MVS for rapid 3D change detection, a processing pipeline is evaluated in which the 3D change is determined by comparison of two depth maps, namely a depth map that is estimated from the image data of an input stream, and a depth map which is synthetically rendered from a reference model against which the change is to be detected. The qualitative experiments show that this approach is well capable of detecting changes with respect to a 3D model. Yet, it is very sensitive to an inaccurate co-registration of the image data and the 3D model as well as a difference in the depth scale of the two depth maps, leading to a number of false-positive and false-negative detections. Detecting 3D changes based on depth difference, however, is computationally efficient and is thus particularly suited for fast and rapid processing.

The focus of this work lies in three different methodical approaches for fast dense depth estimation based on image data captured by COTS UAVs. In addition, these approaches are embedded into a bigger framework focusing on three high-level applications. Consequently, the future work addresses both methodical aspects with respect to the different approaches, and conceptional aspects with respect to the proposed framework. While some of the future work that address the different methodical aspects is already discussed in the corresponding chapters, the most important and overlapping aspects are discussed below:

- **Further optimization with respect to run-time and required hardware resources:** The approaches presented in this work are intended for real-time and online processing and have been optimized accordingly. While they already reach the desired performance, thorough evaluation has shown that they still require more optimization with respect to run-time and the hardware resources needed. ReS^{2}tAC, for example, does not reach the same throughput as comparable approaches from literature. FaSS-MVS, on the other hand, requires a large amount of hardware resources which also has a negative effect on the run-time. And the maximum frame-size of the approach for SSDE is limited due to a large batch size that is required for a stable training. Thus, further implementation-specific optimization is required to further increase the performance of the presented approaches for dense depth estimation with respect to processing speed, which, in turn, increases their benefit.
- **Benchmarking and use-case-specific evaluation:** A great difficulty in the evaluation of the approaches presented in this work is the lack of public benchmark datasets for the task of fast and online depth estimation, which in turn inhibits a direct comparison to similar approaches from literature. In particular, appropriate datasets for the task of fast monocular multi-view stereo from aerial imagery, as tackled by FaSS-MVS, are missing. Furthermore, the availability of datasets that focus on the use-case of obstacle detection and collision avoidance as well as fast 3D mapping and 3D change detection is not given. This raises the need for further use-case-specific evaluation and benchmarking in future work. In this, the acquisition and publishing of appropriate datasets with high-quality ground truth is essential in order to facilitate a benchmarking with respect to similar approaches from literature.
- **Extensive investigation of learning-based approaches with respect to generalization and practical use:** In recent years, the performance of techniques that rely on deep learning has increased significantly. For the task of depth estimation, deep-learning-based approaches stand at the top of the majority of public benchmarks. Thus, their use seems very promising with respect to highly accurate and fast processing. Nonetheless, their performance “in the wild”, meaning outside of carefully crafted datasets, is still an open issue and needs further investigation. In particular, investigations on whether such approaches are suited for a practical use and how they generalize to unknown scenes and objects are to be conducted. As discussed, the approach for self-supervised single-view depth estimation, for example, has certain self-improving capabilities, which need further evaluation with respect to more variety and novelty in the considered scenes.
- **Complementary use of DIM and SDE approaches:** The experiments conducted in the scope of this work suggest a complementary use of conventional approaches that rely on DIM as well as learning-based approaches such as the one for self-supervised single-view depth estimation. This would allow to make use of the individual strengths of the different types of approaches, while at the same time eliminating prominent weaknesses. The limitation of DIM-based approaches with respect to occluded areas and dynamic objects, for example, could be compensated by approaches that only rely on one input image like the one for SSDE presented in this work. At the same time, the depth maps computed by DIM can help to induce a scale and consistency into the single-view depth maps. This complementary use, however, is to this end only a suggestion and theoretical discussion, which needs to be implemented and further investigated in future work.
- **Extension of the proposed framework towards the complementary use of image-based techniques and active sensing methods:** Lastly, this work focuses on depth estimation from image data captured by visual cameras. This is motivated by the fact that appropriate cameras are available on a

majority of COTS UAVs by default and are significantly cheaper than other sensors, such as LiDAR systems for example. However, the use of images from visual cameras is subjected to suitable visibility and lighting conditions which, in turn, depend on the environmental conditions, e.g. weather and daytime. In recent years, more and more sensors that rely on active sensing, in particular LiDAR systems, are available for the use and deployment on UAVs, allowing for a complementary use of image-based techniques and active sensing methods. Especially the high-level applications considered in the second part of the proposed framework, i.e. fast, accurate and reliable 3D mapping and 3D change detection, could greatly benefit from an additional use of active sensing techniques. Thus, in future work, the proposed framework is to be extended and evaluated to possibly also integrate data from active sensors, such as LiDAR systems, if available.

Bibliography

ARM (2013): NEON programmers guide (cit. on pp. 27, 33, 34).

ARM (2017): ARM architecture reference manual for ARMv8 architecture profile (cit. on p. 33).

Aanæs, H.; R. Jensenand; G. Vogiatzis; E. Tola and A. B. Dahl (2016): "Large-scale data for multiple-view stereopsis". In: *International Journal of Computer Vision* 120, pp. 153–168 (cit. on p. 74).

Arndt, O. J.; D. Becker; C. Banz and H. Blume (2013): "Parallel implementation of real-time semi-global matching on embedded multi-core architectures". In: *Proceedings of the IEEE International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, pp. 56–63 (cit. on pp. 26, 27).

Banz, C.; S. Hesselbarth; H. Flatt; H. Blume and P. Pirsch (2010): "Real-time stereo vision system using semi-global matching disparity estimation: Architecture and FPGA-implementation". In: *Proceedings of the IEEE International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 93–101 (cit. on pp. 24, 26, 47, 86).

Banz, C.; H. Blume and P. Pirsch (2011): "Real-time semi-global matching disparity estimation on the GPU". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 514–521 (cit. on pp. 25, 26, 60).

Barredo Arrieta, A.; N. Díaz-Rodríguez; J. Del Ser; A. Bennetot; S. Tabik; A. Barbado; S. Garcia; S. Gil-Lopez; D. Molina; R. Benjamins; R. Chatila and F. Herrera (2020): "Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI". In: *Information Fusion* 58, pp. 82–115 (cit. on p. 121).

Barry, A. J.; H. Oleynikova; D. Honegger; M. Pollefeys and R. Tedrake (2015): "FPGA vs. pushbroom stereo vision for MAVs". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Workshop on Vision-based Control and Navigation of Small Lightweight UAVs* (cit. on pp. 25, 26, 60).

Bay, H.; T. Tuytelaars and L. V. Gool (2006): "SURF: Speeded up robust features". In: *Proceedings of the European Conference on Computer Vision*, pp. 404–417 (cit. on p. 13).

Campos, C.; R. Elvira; J. J. Gómez; J. M. M. Montiel and J. D. Tardós (2021): "ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM". In: *IEEE Transactions on Robotics* 37.6, pp. 1874 –1890 (cit. on pp. 117, 124).

Chang, Q.; A. Zha; W. Wang; X. Liu; M. Onishi and T. Maruyama (2020): "Z2-ZNCC: ZigZag scanning based zero-means normalized cross correlation for fast and accurate stereo matching on embedded GPU". In: *Proceedings of the IEEE International Conference on Computer Design*, pp. 597–600 (cit. on pp. 26, 41, 46, 47, 49, 53).

Chen, A. Y.; Y.-N. Huang; J.-Y. Han and S.-C. J Kang (2014): "A review of rotorcraft unmanned aerial vehicle (UAV) developments and applications in civil engineering". In: *Smart Structures and Systems* 13.6, pp. 1065–1094 (cit. on p. 3).

Cheng, S.; Z. Xu; S. Zhu; Z. Li; L. E. Li; R. Ramamoorthi and H. Su (2020): "Deep stereo using adaptive thin volume representation with uncertainty awareness". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2524–2534 (cit. on p. 61).

Collins, R. T. (1996): "A space-sweep approach to true multi-image matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 358–363 (cit. on pp. 15, 64).

Cordts, M.; M. Omran; S. Ramos; T. Rehfeld; M Enzweiler; R. Benenson; U. Franke; S. Roth and B. Schiele (2016): "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3213–3223 (cit. on p. 100).

Cui, H. and N. Dahnoun (2019): "Real-time stereo vision implementation on Nvidia Jetson TX2". In: *Proceedings of the IEEE Mediterranean Conference on Embedded Computing*. 18833784 (cit. on pp. 41, 46, 47, 49, 53).

DJI (2020a): Matrice 2 Pro/Zoom - User Manual (cit. on p. 93).

DJI (2020b): Matrice 200 V2-Serie - User Manual (cit. on pp. 54, 93).

DJI (2021): DJI onboard SDK advanced sensing - stereo camera. Available online: https://developer.dji.com/onboard-sdk/documentation/guides/component-guide-advanced-sensing-stereo-camera.html (accessed on 19th March 2021) (cit. on p. 51).

Davison, A. J.; J. D. Reid; N. D. Molton and O. Stasse (2007): "MonoSLAM: Real-time single camera SLAM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6, pp. 1052–1067 (cit. on p. 59).

Dean, J. and S. Ghemawat (2008): "MapReduce: Simplified data processing on large clusters". In: *Communications of the ACM* 51.1, pp. 107–113 (cit. on p. 32).

Doherty, P. and P. Rudol (2007): "A UAV search and rescue scenario with human body detection and geolocalization". In: *Advances in Artificial Intelligence* (cit. on p. 3).

DroneDeploy (2018): Drones assess the aftermath of Indonesia's destructive earthquake. Available online: https://medium.com/aerial-acuity/drones-assess-the-aftermath-of-a-indonesias-destructive-earthquake-1e60611d0abd (accessed on 4th November 2021) (cit. on p. 4).

Eade, E. and T. Drummond (2006): "Scalable monocular SLAM". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 469–476 (cit. on p. 59).

Eastern Kentucky University (2017): 5 ways drones are being used for disaster relief. Available online: https://safetymanagement.eku.edu/blog/5-ways-drones-are-being-used-for-disaster-relief/ (accessed on 4th November 2021) (cit. on p. 4).

Eigen, D.; C. Puhrsch and R. Fergus (2014): "Depth map prediction from a single image using a multi-scale deep network". In: *Proceedings of the Advances in Neural Information Processing Systems*, pp. 2366–2374 (cit. on pp. 97, 99).

Ernst, I. and H. Hirschmüller (2008): "Mutual information based semi-global stereo matching on the GPU". In: *Proceedings of the International Symposium on Advances in Visual Computing*, pp. 228–239 (cit. on p. 25).

Estrada, M. A. R. and A. Ndoma (2019): "The uses of unmanned aerial vehicles–UAV's-(or drones) in social logistic: Natural disasters response and humanitarian relief aid". In: *Procedia Computer Science* 149, pp. 375–383 (cit. on p. 3).

Ferris-Rotman, A. (2015): How drones are helping Nepal recover from the earthquake. Ed. by The Huffington Post. Available online: https://www.huffpost.com/entry/nepal-earthquake-drones_n_7232764 (accessed on 4th November 2021) (cit. on pp. 4, 5).

Finn, R. L. and D. Wright (2012): "Unmanned aircraft systems: Surveillance, ethics and privacy in civil applications". In: *Computer Law & Security Review* 28.2, pp. 184–194 (cit. on p. 3).

Flynn, J.; I. Neulander; J. Philbin and N. Snavely (2016): "DeepStereo: Learning to predict new views from the world's imagery". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5515–5524 (cit. on p. 99).

Furukawa, Y. and J. Ponce (2010): "Accurate, dense, and robust multi-view stereopsis". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.8, pp. 1362–1376 (cit. on pp. 58, 89).

Furukawa, Y.; B. Curless; S. M. Seitz and R. Szeliski (2009): "Manhattan-world stereo". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1422–1429 (cit. on p. 59).

Furutani, T. and M. Minami (2021): "Drones for disaster risk reduction and crisis response". In: *Emerging Technologies for Disaster Resilience*, pp. 51–62 (cit. on pp. 3, 4, 57).

Galliani, S.; K. Lasinger and K. Schindler (2015): "Massively parallel multi-view stereopsis by surface normal diffusion". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 873–881 (cit. on p. 89).

Gallup, D.; J.-M. Frahm; P. Mordohai; Q. Yang and M. Pollefeys (2007): "Real-time plane-sweeping stereo with multiple sweeping directions". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 9738134 (cit. on pp. 16, 59, 91).

Gallup, D.; J.-F. Frahm and M. Pollefeys (2010): "Piecewise planar and non-planar stereo for urban scene reconstruction". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1418–1425 (cit. on p. 59).

Gehrig, S. K. and C. Rabe (2010): "Real-time semi-global matching on the CPU". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 85–92 (cit. on pp. 25–27).

Gehrig, S. K.; F. Eberli and T. Meyer (2009): "A real-time low-power stereo vision engine using semi-global matching". In: *Proceedings of the International Conference on Computer Vision Systems*, pp. 134–143 (cit. on pp. 24, 26).

Geiger, A.; M. Roser and R. Urtasun (2011): "Efficient large-scale stereo matching". In: *Proceedings of the Asian Conference on Computer Vision*, pp. 25–38 (cit. on p. 27).

Godard, C.; O. M. Aodha and G. J. Brostow (2017): "Unsupervised monocular depth estimation with left-right consistency". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 270–279 (cit. on pp. 97, 98, 100, 102, 105).

Godard, C.; O. M. Aodha; M. Firman and G. J. Brostow (2019): "Digging into self-supervised monocular depth estimation". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3828–3838 (cit. on pp. 98, 100, 104, 106–109, 111).

Goesele, M.; N. Snavely; B. Curless; H. Hoppe and S. M. Seitz (2007): "Multi-view stereo for community photo collections". In: *Proceedings of the IEEE International Conference on Computer Vision*. 9848978 (cit. on p. 58).

Gu, X.; Z. Fan; S. Zhu; Z. Dai; F. Tan and P. Tan (2020): "Cascade cost volume for high-resolution multi-view stereo and stereo matching". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2495–2504 (cit. on p. 61).

Haala, N.; M. Rothermel and S. Cavegn (2015): "Extracting 3D urban models from oblique aerial images". In: *Proceedings of the Joint Urban Remote Sensing Event*. 15201729 (cit. on pp. 60, 61, 64).

Han, X.; T. Leung; Y. Jia; R. Sukthankar and A. C. Berg (2015): "MatchNet: Unifying feature and metric learning for patch-based matching". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3279–3286 (cit. on p. 60).

Hartley, R. and A. Zisserman (2004): Multiple view geometry in computer vision. Cambridge University Press (cit. on pp. 12, 15).

Hartmann, W.; S. Galliani; M. Havlena; L. Van Gool and K. Schindler (2017): "Learned multi-patch similarity". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1586–1594 (cit. on p. 60).

He, K.; X. Zhang; S. Ren and J. Sun (2016): "Deep residual learning for image recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778 (cit. on pp. 99, 102, 111).

Heinrich, K. and M. Mehltretter (2021): "Learning multi-modal features for dense matching-based confidence estimation". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2021, pp. 91–99 (cit. on p. 94).

Hermann, M.; B. Ruf; M. Weinmann and S. Hinz (2020): "Self-supervised learning for monocular depth estimation from aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-2-2020, pp. 357–364 (cit. on pp. 8, 9, 98, 100, 101, 103, 107, 109, 115, 161).

Hermann, M.; B. Ruf and M. Weinmann (2021a): "Real-time dense 3D reconstruction from monocular video data captured by low-cost UAVs". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2021, pp. 361–368 (cit. on pp. 8, 9, 94, 123, 124, 161).

Hermann, M.; T. Pollok; D. Brommer and D. Zahn (2021b): "iVS3D: An open source framework for intelligent video sampling and preprocessing to facilitate 3D reconstruction". In: *Advances in Visual Computing*, pp. 441–454 (cit. on p. 101).

Hermann, S.; R. Klette and E. Destefanis (2009): "Inclusion of a second-order prior into semi-global matching". In: *Proceedings of the Pacific-Rim Symposium on Image and Video Technology*, pp. 633–644 (cit. on p. 60).

Hernandez-Juarez, D.; A. Chacón; A. Espinosa; D. Vázquez; J. C. Moure and A. M. López (2016): "Embedded real-time stereo estimation via semi-global matching on the GPU". In: *Procedia Computer Science* 80, pp. 143–153 (cit. on pp. 25, 26, 41, 46–50, 53, 60, 86).

Hirschmüller, H. (2005): "Accurate and efficient stereo processing by semi-global matching and mutual information". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 807–814 (cit. on pp. 17, 23, 24, 55, 57, 60, 63, 69, 72).

Hirschmüller, H. (2008): "Stereo processing by semi-global matching and mutual information". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.2, pp. 328–341 (cit. on pp. 17, 23, 24, 41, 43, 55, 57, 60, 63, 69, 72).

Hofmann, J.; J. Korinth and A. Koch (2016): "A scalable high-performance hardware architecture for real-time stereo vision by semi-global matching". In: *Proceeding of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 845–853 (cit. on pp. 25, 26).

Honegger, D.; H. Oleynikova and M. Pollefeys (2014): "Real-time and low latency embedded computer vision hardware based on a combination of FPGA and mobile CPU". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4930–4935 (cit. on pp. 25, 26).

Huang, B.; H. Yi; C. Huang; Y. He; J. Liu and X. Liu (2021): "M3VSNET: Unsupervised multi-metric multi-view stereo network". In: *Proceedings of the IEEE International Conference on Image Processing*, pp. 3163–3167 (cit. on p. 61).

Huang, G.; Z. Liu; L. Van Der Maaten and K. Q. Weinberger (2017): "Densely connected convolutional networks". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4700–4708 (cit. on p. 111).

Huang, P.-H.; K. Matzen; J. Kopf; N. Ahuja and J.-B. Huang (2018): "DeepMVS: Learning multi-view stereopsis". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2821–2830 (cit. on p. 61).

Jaderberg, M.; K. Simonyan; A. Zisserman and K. Kavukcuoglu (2015): "Spatial transformer networks". In: *Proceedings of the Advances in Neural Information Processing Systems*, pp. 2017–2025 (cit. on pp. 103, 104).

Jensen, R.; A. Dahl; G. Vogiatzis; E. Tola and H. Aanæs (2014): "Large scale multi-view stereopsis evaluation". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 406–413 (cit. on pp. 61, 74).

Ji, M.; J. Gall; H. Zheng; Y. Liu and L. Fang (2017): "SurfaceNet: An end-to-end 3D neural network for multi-view stereopsis". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2307–2315 (cit. on p. 61).

Johnson-Roberson, M.; C. Barto; R. Mehta; S. N. Sridhar; K. Rosaen and R. Vasudevan (2017): "Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks?" In: *IEEE International Conference on Robotics and Automation*, pp. 746–753 (cit. on pp. 99, 107).

Kalb, T.; L. Kalms; D. Göhringer; C. Pons; F. Marty; A. Muddukrishna; M. Jahre; P. G. Kjeldsberg; B. Ruf; T. Schuchert; I. Tchouchenkov; C. Ehrenstråhle; F. Christensen; A. Paolillo; C. Lemer; G.

Bernard; F. Duhem and P. Millet (2016): "Tulipp: Towards ubiquitous low-power image processing platforms". In: *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 306–311 (cit. on p. 162).

Kalb, T.; L. Kalms; D. Göhringer; C. Pons; A. Muddukrishna; M. Jahre; B. Ruf; T. Schuchert; I. Tchouchenkov; C. Ehrenstråhle; M. Peterson; F. Christensen; A. Paolillo; B. Rodriguez and P. Millet (2019): "Developing low-power image processing applications with the TULIPP reference platform instance". In: *Hardware Accelerators in Data Centers*, pp. 181–197 (cit. on pp. 25, 162).

Kang, S. B.; R. Szeliski and J. Chai (2001): "Handling occlusions in dense multi-view stereo". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 103–110 (cit. on pp. 64, 65, 104).

Kerle, N.; F. Nex; M. Gerke; D. Duarte and A. Vetrivel (2020): "UAV-based structural damage mapping: A review". In: *ISPRS International Journal of Geo-Information* 9.1, 14 (cit. on p. 3).

Khot, T.; S. Agrawal; S. Tulsiani; C. Mertz; S. Lucey and M. Hebert (2019): "Learning unsupervised multi-view stereopsis via robust photometric consistency". In: *arXiv preprint arXiv:1905.02706* (cit. on pp. 61, 100).

Kingma, D. P. and J. Ba (2015): "Adam: A method for stochastic optimization". In: *Proceedings of the International Conference on Learning Representations* (cit. on p. 106).

Klein, G. and D. Murray (2007): "Parallel tracking and mapping for small AR workspaces". In: *Proceedings of the IEEE/ACM International Symposium on Mixed and Augmented Reality*, pp. 225–234 (cit. on p. 59).

Knapitsch, A.; J. Park; Q.-Y. Zhou and V. Koltun (2017): "Tanks and temples: Benchmarking large-scale scene reconstruction". In: *ACM Transactions on Graphics* 36.4, 78 (cit. on p. 77).

Knöbelreiter, P.; C. Reinbacher; A. Shekhovtsov and T. Pock (2017): "End-to-end training of hybrid CNN-CRF models for stereo". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2339–2348 (cit. on p. 100).

Knöbelreiter, P.; C. Vogel and T. Pock (2018): "Self-supervised learning for stereo reconstruction on aerial images". In: *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium*, pp. 4379–4382 (cit. on p. 100).

Knuth, D. E. (1998): The art of computer programming: sorting and searching. Pearson Education (cit. on p. 38).

Kolev, K.; P. Tanskanen; P. Speciale and M. Pollefeys (2014): "Turning mobile phones into 3D scanners". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3946–3953 (cit. on p. 73).

Kuschk, G. and D. Cremers (2013): "Fast and accurate large-scale stereo reconstruction using variational methods". In: *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pp. 700–707 (cit. on p. 60).

Labayrade, R.; D. Aubert and J. P. Tarel (2002): "Real time obstacle detection in stereovision on non flat road geometry through 'v-disparity' representation". In: *Proceedings of the IEEE Intelligent Vehicle Symposium*, pp. 646–651 (cit. on p. 120).

Laina, I.; C. Rupprecht; V. Belagiannis; F. Tombari and N. Navab (2016): "Deeper depth prediction with fully convolutional residual networks". In: *Proceedings of the IEEE International Conference on 3D Vision*, pp. 239–248 (cit. on pp. 97, 99).

Li, B.; C. Shen; Y. Dai; A. Van Den Hengel and M. He (2015): "Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1119–1127 (cit. on pp. 97, 99).

Li, Y. and Y. Ruichek (2014): "Occupancy grid mapping in urban environments from a moving on-board stereo-vision system". In: *MDPI Sensors* 14.6, pp. 10454–10478 (cit. on p. 120).

Lowe, D. G. (2004): "Distinctive image features from scale-invariant keypoints". In: *International Journal of Computer Vision* 60.2, pp. 91–110 (cit. on p. 13).

Madhuanand, L.; F. Nex and M. Y. Yang (2021): "Self-supervised monocular depth estimation from oblique UAV videos". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 176, pp. 1–14 (cit. on pp. 100, 105, 109).

Mahjourian, R.; M. Wicke and A. Angelova (2018): "Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5667–5675 (cit. on pp. 98, 100, 105).

Markets-and-Markets (2021): Unmanned aerial vehicle (UAV) market by point of sale, systems, platform, function, end use, application, type, mode of operation, MTOW, range, and region - Global forecast to 2026. Report (cit. on p. 3).

Mayer, N.; E. Ilg; P. Fischer; C. Hazirbas; D. Cremers; A. Dosovitskiy and T. Brox (2018): "What makes good synthetic training data for learning disparity and optical flow estimation?" In: *International Journal of Computer Vision* 126.9, pp. 942–960 (cit. on p. 99).

Menze, M. and A. Geiger (2015): "Object scene flow for autonomous vehicles". In: *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pp. 3061–3070 (cit. on pp. 39–42, 45, 98–100, 107, 108).

Michael, M.; J. Salmen; J. Stallkamp and M. Schlipsing (2013): "Real-time stereo vision: Optimizing semi-global matching". In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 1197–1202 (cit. on pp. 25, 26).

Mur-Artal, R. and J. D. Tardós (2017): "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras". In: *IEEE Transactions on Robotics* 33.5, pp. 1255–1262 (cit. on p. 122).

Newcombe, R. A. and A. J. Davison (2010): "Live dense reconstruction with a single moving camera". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1498–1505 (cit. on p. 59).

Newcombe, R. A.; S. J. Lovegrove and A. J. Davison (2011): "DTAM: Dense tracking and mapping in real-time". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2320–2327 (cit. on p. 59).

Nex, F. and F. Rinaudo (2011): "LiDAR or photogrammetry? Integration is the answer". In: *Italian Journal of Remote Sensing* 43.2, pp. 107–121 (cit. on pp. 23, 95).

Ni, J.; Q. Li; Y. Liu and Y. Zhou (2018): "Second-order semi-global stereo matching algorithm based on slanted plane iterative optimization". In: *IEEE Access* 6, pp. 61735–61747 (cit. on p. 60).

Oezdemir, E.; I. Toschi and F. Remondino (2019): "A multi-purpose benchmark for photogrammetric urban 3D reconstruction in a controlled environment". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1/W2, pp. 53–60 (cit. on pp. 74, 76).

Oleynikova, H.; D. Honegger and M. Pollefeys (2015): "Reactive avoidance using embedded stereo vision for MAV flight". In: *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 50–56 (cit. on p. 120).

OpenMVS (2022): OpenMVS: open Multi-View Stereo reconstruction library. Available online: https://github.com/cdcseacave/openMVS (accessed on 1st April 2022) (cit. on p. 58).

Perz, R. and K. Wronowski (2019): "UAV application for precision agriculture". In: *Aircraft Engineering and Aerospace Technology* 91.2, pp. 257–263 (cit. on p. 3).

Poggi, M.; F. Tosi and S. Mattoccia (2020): "Learning a confidence measure in the disparity domain from O(1) features". In: *Computer Vision and Image Understanding* 193, 102905 (cit. on p. 94).

Pollefeys, M.; D. Nistér; J.-M. Frahm; A. Akbarzadeh; P. Mordohai; B. Clipp; C. Engels; D. Gallup; S.-J. Kim; P. Merrell; C. Salmi; S. Sinha; B. Talton; L. Wang; Q. Yang; H. Stewénius; R. Yang; G. Welch and H. Towles (2008): "Detailed real-time urban 3D reconstruction from video". In: *International Journal of Computer Vision* 78, pp. 143–167 (cit. on pp. 16, 59, 61, 64, 66, 68, 91, 92).

Pollok, T.; L. Junglas; B. Ruf and A. Schumann (2019): "UnrealGT: Using unreal engine to generate ground truth datasets". In: *International Symposium on Visual Computing*, pp. 670–682 (cit. on p. 162).

Qin, R.; J. Tian and P. Reinartz (2016): "3D change detection - approaches and applications". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 122, pp. 41–56 (cit. on pp. 125, 126, 128).

Rahnama, O.; T. Cavalleri; S. Golodetz; S. Walker and P. Torr (2018a): "R3SGM: Real-time raster-respecting semi-global matching for power-constrained systems". In: *Proceedings of the IEEE International Conference on Field-Programmable Technology*, pp. 102–109 (cit. on pp. 25, 26, 41, 46, 49, 50, 53).

Rahnama, O.; D. Frost; O. Miksik and P. H. Torr (2018b): "Real-time dense stereo matching with ELAS on FPGA-accelerated embedded devices". In: *IEEE Robotics and Automation Letters* 3.3, pp. 2008–2015 (cit. on p. 27).

Restas, A. (2015): "Drone applications for supporting disaster management". In: *World Journal of Engineering and Technology* 3, pp. 316–321 (cit. on pp. 3, 57).

Ronneberger, O.; P. Fischer and T. Brox (2015): "U-Net: Convolutional networks for biomedical image segmentation". In: *Proceedings of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241 (cit. on pp. 61, 102).

Rosenberg, I. D.; P. L. Davidson; C. M. Muller and J. Y. Han (2006): "Real-time stereo vision using semi-global matching on programmable graphics hardware". In: *ACM SIGGRAPH 2006 Sketches*, 89–es (cit. on p. 25).

Roth, L. and H. Mayer (2019): "Reduction of the fronto-parallel bias for wide-baseline semi-global matching". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W5, pp. 69–76 (cit. on p. 61).

Rothermel, M.; K. Wenzel; D. Fritsch and N. Haala (2012): "SURE: Photogrammetric surface reconstruction from imagery". In: *Proceedings of the LowCost3D Workshop* 8.2 (cit. on pp. 58, 60).

Rublee, E.; V. Rabaud; K. Konolige and G. Bradski (2011): "ORB: An efficient alternative to SIFT or SURF". In: *Proceedings of the International Conference on Computer Vision*, pp. 2564–2571 (cit. on pp. 13, 122).

Ruf, B. and T. Schuchert (2016): "Towards real-time change detection in videos based on existing 3D models". In: *Proceedings of the SPIE Image and Signal Processing for Remote Sensing XXII*, pp. 489 –502 (cit. on pp. 8, 9, 126, 162).

Ruf, B.; B. Erdnüß and M. Weinmann (2017): "Determining plane-sweep sampling points in image space using the cross-ratio for image-based depth estimation". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W6, pp. 325–332 (cit. on pp. 9, 58, 59, 161).

Ruf, B.; L. Thiel and M. Weinmann (2018a): "Deep cross-domain building extraction for selective depth estimation from oblique aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-1, pp. 125–132 (cit. on pp. 9, 58, 161).

Ruf, B.; S. Monka; M. Kollmann and M. Grinberg (2018b): "Real-time on-board obstacle avoidance for UAVs based on embedded stereo vision". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1, pp. 363–370 (cit. on pp. 6, 7, 25, 26, 41, 46, 49, 50, 60, 119–121, 161).

Ruf, B.; T. Pollok and M. Weinmann (2019): "Efficient surface-aware semi-global matching with multi-view plane-sweep sampling". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W7, pp. 137–144 (cit. on pp. 8, 9, 58, 94, 161).

Ruf, B.; M. Weinmann and S. Hinz (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821* (cit. on pp. xiii, xiv, 7–9, 11, 15, 16, 20, 22, 57–95, 161).

Ruf, B.; J. Mohrs; M. Weinmann; S. Hinz and J. Beyerer (2021b): "ReS^{2}tAC – UAV-borne real-time SGM stereo optimized for embedded ARM and CUDA devices". In: *MDPI Sensors* 21.11, 3938 (cit. on pp. 6, 7, 11–14, 17–19, 21, 23–55, 60, 80, 161).

Saidi, T. E.; A. Khouas and A. Amira (2020): "Accelerating stereo matching on mutlicore ARM platform". In: *Proceedings of the IEEE International Symposium on Circuits and Systems*. 20727693 (cit. on p. 27).

Sankarasrinivasan, S; E Balasubramanian; K Karthik; U Chandrasekar and R. Gupta (2015): "Health monitoring of civil structures with integrated UAV and image processing system". In: *Procedia Computer Science* 54, pp. 508–515 (cit. on p. 3).

Scharstein, D. and R. Szeliski (2002): "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms". In: *International Journal of Computer Vision* 47, pp. 7–42 (cit. on pp. 13, 17).

Scharstein, D.; H. Hirschmüller; Y. Kitajima; G. Krathwohl; N. Nešić; X. Wang and P. Westling (2014): "High-resolution stereo datasets with subpixel-accurate ground truth". In: *Proceedings of the German Conference on Pattern Recognition*, pp. 31–42 (cit. on pp. 40, 43–45).

Scharstein, D.; T. Taniai and S. N. Sinha (2017): "Semi-global stereo matching with surface orientation priors". In: *Proceedings of the International Conference on 3D Vision*, pp. 215–224 (cit. on pp. 15, 60, 61, 64, 69, 71, 72, 78, 92).

Schmid, K.; T. Tomic; F. Ruess; H. Hirschmüller and M. Suppa (2013): "Stereo vision based indoor/outdoor navigation for flying robots". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3955–3962 (cit. on p. 25).

Schmitz, S.; M. Weinmann and B. Ruf (2019): "Automatic co-registration of aerial imagery and untextured model data utilizing average shading gradients". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W13, pp. 581–588 (cit. on pp. 127, 162).

Schönberger, J. L.; E. Zheng; J.-M. Frahm and M. Pollefeys (2016): "Pixelwise view selection for unstructured multi-view stereo". In: *Proceedings of the European Conference on Computer Vision*, pp. 501–518 (cit. on pp. xiv, 22, 57, 58, 74, 77, 88, 89, 107, 122, 128).

Schönberger, J. L.; S. N. Sinha and M. Pollefeys (2018): "Learning to fuse proposals from multiple scanline optimizations in semi-global matching". In: *Proceedings of the European Conference on Computer Vision*, pp. 739–755 (cit. on pp. 41, 43).

Schönberger, J. L. and J.-M. Frahm (2016): "Structure-from-motion revisited". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4104–4113 (cit. on pp. xiv, 57, 76, 77, 107, 122, 128).

Schöps, T.; J. Schönberger; S. Galliani; T. Sattler; K. Schindler; M. Pollefeys and A. Geiger (2017): "A multi-view stereo benchmark with high-resolution images and multi-camera videos". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern*, pp. 3260–3269 (cit. on pp. 77, 98, 99).

Shah, S.; D. Dey; C. Lovett and A. Kapoor (2017): "AirSim: High-fidelity visual and physical simulation for autonomous vehicles". In: *Field and Service Robotics*, pp. 621–635 (cit. on p. 121).

Shakhatreh, H.; A. H. Sawalmeh; A. Al-Fuqaha; Z. Dou; E. Almaita; I. Khalil; N. S. Othman; A. Khreishah and M. Guizani (2019): "Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges". In: *IEEE Access* 7, pp. 48572–48634 (cit. on p. 5).

Silberman, N.; D. Hoiem; P. Kohli and R. Fergus (2012): "Indoor segmentation and support inference from RGBD images". In: *Proceedings of the European Conference on Computer Vision*, pp. 746–760 (cit. on pp. 98, 99).

Silvagni, M.; A. Tonoli; E. Zenerino and M. Chiaberge (2017): "Multipurpose UAV for search and rescue operations in mountain avalanche events". In: *Geomatics, Natural Hazards and Risk* 8.1, pp. 18–33 (cit. on pp. 3, 4).

Simonyan, K. and A. Zisserman (2015): "Very deep convolutional networks for large-scale image recognition". In: *Proceedings of the International Conference on Learning Representations*, pp. 1–14 (cit. on pp. 102, 111).

Sinha, S. N.; D. Steedly and R. Szeliski (2009): "Piecewise planar stereo for image-based rendering". In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1881–1888 (cit. on p. 59).

Sinha, S. N.; D. Scharstein and R. Szeliski (2014): "Efficient high-resolution stereo matching using local plane sweeps". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1582–1589 (cit. on pp. 15, 16, 60, 64, 92).

Spangenberg, R.; T. Langner; S. Adfeldt and R. Rojas (2014): "Large scale semi-global matching on the CPU". In: *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 195–201 (cit. on pp. 26, 27, 37, 60).

Suzuki, S. and K. Abe (1985): "New fusion operations for digitized binary images and their applications". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 7.6, pp. 638–651 (cit. on p. 121).

Szeliski, R. and D. Scharstein (2004): "Sampling the disparity space image". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26.3, pp. 419–425 (cit. on p. 66).

Tsouros, D. C.; S. Bibi and P. G. Sarigiannidis (2019): "A review on UAV-based applications for precision agriculture". In: *Information* 10.11, 349 (cit. on p. 3).

Wang, C.; J. M. Buenaposada; R. Zhu and S. Lucey (2018): "Learning depth from monocular videos using direct methods". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2022–2030 (cit. on pp. 98, 100, 105).

Wang, W.; J. Yan; N. Xu; Y. Wang and F. Hsu (2015): "Real-time high-quality stereo vision system in FPGA". In: *IEEE Transactions on Circuits and Systems for Video Technology* 25.10, pp. 1696–1708 (cit. on pp. 25, 26).

Wenzel, K. (2016): "Dense image matching for close range photogrammetry". PhD thesis. University of Stuttgart, Germany (cit. on p. 20).

Wenzel, K.; M. Rothermel; D. Fritsch and N. Haala (2013a): "Image acquisition and model selection for multi-view stereo". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 40.5W, pp. 251–258 (cit. on p. 76).

Wenzel, K.; M. Rothermel; N. Haala and D. Fritsch (2013b): "SURE – The IFP software for dense image matching". In: *Proceedings of the Photogrammetric Week*, pp. 59–70 (cit. on pp. 58, 60).

Whelan, T.; S. Leutenegger; R. F. Salas-Moreno; B. Glocker and A. Davison (2015): "ElasticFusion: Dense SLAM without a pose graph". In: *Proceedings of Robotics: Science and Systems* (cit. on p. 123).

Whelan, T.; R. F. Salas-Moreno; B. Glocker; A. J. Davison and S. Leutenegger (2016): "ElasticFusion: Real-time dense SLAM and light source estimation". In: *The International Journal of Robotics Research* 35.14, pp. 1697–1716 (cit. on p. 123).

Xie, J.; R. Girshick and A. Farhadi (2016): "Deep3D: Fully automatic 2D-to-3D video conversion with deep convolutional neural networks". In: *Proceedings of the European Conference on Computer Vision*, pp. 842–857 (cit. on p. 99).

Yan, J.; Z. Wei; H. Yi; M. Ding; R. Zhang; Y. Chen; G. Wang and Y.-W. Tai (2020): "Dense hybrid recurrent multi-view stereo net with dynamic consistency checking". In: *Proceedings of the European Conference on Computer Vision*, pp. 674–689 (cit. on p. 61).

Yao, Y.; Z. Luo; S. Li; T. Fang and L. Quan (2018): "MVSNet: Depth inference for unstructured multi-view stereo". In: *Proceedings of the European Conference on Computer Vision*, pp. 767–783 (cit. on pp. 61, 89).

Yao, Y.; Z. Luo; S. Li; T. Shen; T. Fang and L. Quan (2019): "Recurrent MVSNet for high-resolution multi-view stereo depth inference". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5525–5534 (cit. on p. 61).

Yi, H.; Z. Wei; M. Ding; R. Zhang; Y. Chen; G. Wang and Y.-W. Tai (2020): "Pyramid multi-view stereo net with self-adaptive view aggregation". In: *Proceedings of the European Conference on Computer Vision*, pp. 766–782 (cit. on p. 61).

Zabih, R. and J. Woodfill (1994): "Non-parametric local transforms for computing visual correspondence". In: *Proceedings of the European Conference on Computer Vision*, pp. 151–158 (cit. on pp. 14, 64).

Zbontar, J.; Y. LeCun et al. (2016): "Stereo matching by training a convolutional neural network to compare image patches." In: *Journal of Machine Learning Research* 17.1, pp. 2287–2318 (cit. on p. 60).

Zhang, Z. (2000): "A flexible new technique for camera calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22, pp. 1330–1334 (cit. on p. 14).

Zhao, H.; O. Gallo; I. Frosio and J. Kautz (2016): "Loss functions for image restoration with neural networks". In: *IEEE Transactions on Computational Imaging* 3.1, pp. 47–57 (cit. on pp. 100, 105).

Zhao, J.; T. Liang; L. Feng; W. Ding; S. Sinha; W. Zhang and S. Shen (2020): "FP-Stereo: Hardware-efficient stereo vision for embedded applications". In: *Proceedings of the IEEE International Conference on Field-Programmable Logic and Applications*, pp. 269–276 (cit. on pp. 25, 26, 41, 46, 49, 50, 53, 60).

Zhou, T.; M. Brown; N. Snavely and D. G. Lowe (2017): "Unsupervised learning of depth and ego-motion from video". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1851–1858 (cit. on pp. 103, 107–109).

List of Figures

List of Tables

List of Algorithms

Acronyms

FaSS-MVS	Fast Multi-View Stereo with Surface-Aware Semi-Global Matching
ReS^{2}tAC	Real-Time SGM Stereo Optimized for Embedded ARM and CUDA Devices
ADAS	advanced driver assistance system
AI	artificial intelligence
AR	augmented reality
BPU	branch prediction unit
BVLOS	beyond visual line-of-sight
CAGR	compound annual growth rate
CNN	convolutional neural network
COTS	commercial off-the-shelf
CPU	central processing unit
CRF	conditional random field
CT	census transform
DIM	dense image matching
DOG	difference-of-Gaussian
EASA	European Aviation Safety Agency
ELU	exponential linear unit
ENU	east-north-up
FOV	field-of-view
FPGA	field-programmable gate array
FPS	frames per second

FPS/W	FPS per watt
FPV	first person view
GCS	ground control station
GNSS	global navigation satellite system
GPGPU	general-purpose computation on a GPU
GPU	graphic processing unit
GTA V	Grand Theft Auto V
HLS	high-level synthesis
IMU	inertial measurement unit
IR	infrared
KLT	Kanade-Lucas-Tomasi
LiDAR	light detection and ranging
MDE/s	million disparity estimations per second
MGM	More-Global Matching
MRF	Markov random field
MVS	multi-view stereo
NCC	normalized cross correlation
ORB	Oriented FAST and rotated BRIEF
PU	processing unit
RMSE	root-mean-square error
ROC	receiver operating characteristic
RTK	real-time kinematic positioning
SAR	search-and-rescue
SDE	single-view depth estimation
SFM	structure-from-motion

SGBM	Semi-Global Block Matching
SGM	Semi-Global Matching
SIFT	Scale-Invariant Feature Transform
SIMD	single-instruction-multiple-data
SISD	single-instruction-single-data
SLAM	simultaneous localization and mapping
SMDE	self-supervised monocular depth estimation
SOC	system-on-a-chip
SSDE	self-supervised single-view depth estimation
SSE	Streaming SIMD Extensions
SSIM	structural similarity
STN	spatial transformer network
SURF	Speeded Up Robust Features
UAV	unmanned aerial vehicle
USD	US Dollars
VI-SLAM	visual-inertial SLAM
V-SLAM	visual SLAM
WTA	winner-takes-it-all

List of Publications

The following lists the publications which have been published by or with the author Boitumelo Ruf in the period of 2016 - 2021. The publications are grouped into "main publications" and "further publications". The former group holds all publications that are directly related to the content of this work. The publications are sorted in descending order with respect to the year of publication.

Main Publications

- Ruf, B.; Weinmann, M., and Hinz, S. (2021a): "FaSS-MVS – Fast multi-view stereo with surface-aware semi-global matching from UAV-borne monocular imagery". In: *arXiv preprint arXiv:2112.00821*. Published as preprint. Cited as (Ruf et al. 2021a).
- Ruf, B.; Mohrs, J.; Weinmann, M.; Hinz, S., and Beyerer, J. (2021b): "ReS^{2}tAC – UAV-borne real-time SGM stereo optimized for embedded ARM and CUDA devices". In: *MDPI Sensors* 21.11, 3938. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2021b).
- Hermann, M.; Ruf, B., and Weinmann, M. (2021a): "Real-time dense 3D reconstruction from monocular video data captured by low-cost UAVs". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLIII-B2-2021, pp. 361–368. Cited as (Hermann et al. 2021a).
- Hermann, M.; Ruf, B.; Weinmann, M., and Hinz, S. (2020): "Self-supervised learning for monocular depth estimation from aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* V-2-2020, pp. 357–364. **Peer-reviewed** on the basis of the full paper. Cited as (Hermann et al. 2020).
- Ruf, B.; Pollok, T., and Weinmann, M. (2019): "Efficient surface-aware semi-global matching with multi-view plane-sweep sampling". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-2/W7, pp. 137–144. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2019).
- Ruf, B.; Thiel, L., and Weinmann, M. (2018a): "Deep cross-domain building extraction for selective depth estimation from oblique aerial imagery". In: *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences* IV-1, pp. 125–132. **Peer-reviewed** on the basis of the full paper. Cited as (Ruf et al. 2018a).
- Ruf, B.; Monka, S.; Kollmann, M., and Grinberg, M. (2018b): "Real-time on-board obstacle avoidance for UAVs based on embedded stereo vision". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-1, pp. 363–370. Cited as (Ruf et al. 2018b).
- Ruf, B.; Erdnüß, B., and Weinmann, M. (2017): "Determining plane-sweep sampling points in image space using the cross-ratio for image-based depth estimation". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W6, pp. 325–332. Cited as (Ruf et al. 2017).

- Ruf, B. and Schuchert, T. (2016): "Towards real-time change detection in videos based on existing 3D models". In: *Proceedings of the SPIE Image and Signal Processing for Remote Sensing XXII*, pp. 489 –502. Cited as (Ruf and Schuchert 2016).

Further Publications

- Schmitz, S.; Weinmann, M., and Ruf, B. (2019): "Automatic co-registration of aerial imagery and untextured model data utilizing average shading gradients". In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W13, pp. 581–588. Cited as (Schmitz et al. 2019).
- Pollok, T.; Junglas, L.; Ruf, B., and Schumann, A. (2019): "UnrealGT: Using unreal engine to generate ground truth datasets". In: *International Symposium on Visual Computing*, pp. 670–682. **Peer-reviewed** on the basis of the full paper.
- Kalb, T.; Kalms, L.; Göhringer, D.; Pons, C.; Muddukrishna, A.; Jahre, M.; Ruf, B.; Schuchert, T.; Tchouchenkov, I.; Ehrenstråhle, C.; Peterson, M.; Christensen, F.; Paolillo, A.; Rodriguez, B., and Millet, P. (2019): "Developing low-power image processing applications with the TULIPP reference platform instance". In: *Hardware Accelerators in Data Centers*, pp. 181–197. Cited as (Kalb et al. 2019).
- Kalb, T.; Kalms, L.; Göhringer, D.; Pons, C.; Marty, F.; Muddukrishna, A.; Jahre, M.; Kjeldsberg, P. G.; Ruf, B.; Schuchert, T.; Tchouchenkov, I.; Ehrenstråhle, C.; Christensen, F.; Paolillo, A.; Lemer, C.; Bernard, G.; Duhem, F., and Millet, P. (2016): "Tulipp: Towards ubiquitous low-power image processing platforms". In: *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation*, pp. 306–311.