Dissertation
submitted to the
Combined Faculties for the Natural Sciences and for Mathematics
of the Ruperto-Carola University of Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

Put forward by

Dipl.-Phys.: **Jens-Malte Gottfried**
Born in: **Worms, Germany**

Oral examination: 22.11.2016

Handling Artifacts in Dynamic Depth Sequences

Referees:	**Priv.-Doz. Dr. rer. nat.**	**Christoph S. Garbe**
	Prof. Dr. rer. nat.	**Karl-Heinz Brenner**

Behandlung von Artefakten in dynamischen Sequenzen von Tiefenbildern

Bildsequenzen bewegter Szenen, aufgenommen mit verschiedenen Tiefenkameras, die dabei auftretenden Artefakte und deren Handhabung sind das übergreifende Thema dieser Arbeit. Zunächst wird ein Verfahren zur Bestimmung des Tiefenflusses auf Daten der *Kinect* Kamera von *Microsoft* vorgestellt. Alle wichtigen Schritte, beginnend mit der Kamerakalibrierung über die Ausrichtung der Tiefen- und Bilddaten bis zur Einführung eines neuartigen multi-modalen Tiefenflussalgorithmus, der sich robust gegenüber typischen (technologiebedingten) Artefakten verhält, werden dabei behandelt. Anschließend werden Daten aus Lichtlaufzeitkameras und die Bewegungsartefakte, die durch die sequenzielle Natur des Aufnahmeprozesses hervorgerufen werden, betrachtet. Bisher wurden viele Methoden vorgeschlagen, solche Fehler zu beheben, allerdings gibt es noch keinen aussagekräftigen Vergleich. Dies wird hier behoben, indem nicht nur die Ergebnisse der vorhandenen Methoden verglichen, sondern zusätzlich auch technische Eigenschaften und eine nötige Tiefenkorrektur der aufgenommenen Daten als Basis für zukünftige Forschung beleuchtet werden. Ein Austausch des Kalibrierungsmodells der verschiedenen Aufnahmeeinheiten auf dem Sensor, das von den vorgestellten Methoden gebraucht wird, durch ein anderes, das die Realität besser widerspiegelt, verbessert die Ergebnisse aller damit zusammenhängenden Methoden ohne Leistungseinbußen.

Handling Artifacts in Dynamic Depth Sequences

Image sequences of dynamic scenes recorded using various depth imaging devices and handling the artifacts arising within are the main scope of this work. First, a framework for range flow estimation from *Microsoft's* multi-modal imaging device *Kinect* is presented. All essential stages of the flow computation pipeline, starting from camera calibration, followed by the alignment of the range and color channels and finally the introduction of a novel multi-modal range flow algorithm which is robust against typical (technology dependent) range estimation artifacts are discussed. Second, regarding Time-of-Flight data, motion artifacts arise in recordings of dynamic scenes, caused by the sequential nature of the raw image acquisition process. While many methods for compensation of such errors have been proposed so far, there is still a lack of proper comparison. This gap is bridged here by not only evaluating all proposed methods, but also by providing additional insight in the technical properties and depth correction of the recorded data as base-line for future research. Exchanging the tap calibration model necessary for these methods by a model closer to reality improves the results of all related methods without any loss of performance.

Contents

Chapter 1.

Introduction

Figure 1.1.: Overview of the physical depth imaging devices used within this work: Microsoft *Kinect* (**left**), PMD *CamCube3* (**center**) and Fotonic *E40* (**right**[1])

Motion and 3D depth perception are essential key-concepts of human vision, greatly enhancing it over classic photography. Starting in the 17th century, the first stereoscopic images using image pairs drawn by hand or recorded with single cameras shifted between exposures or with multiple lenses tried to simulate the impression of three-dimensional vision to the viewer. For the field of computer vision, things became interesting starting in the late 1960s with the introduction of digital photography based on CCD and CMOS imaging sensors [Nob68; DW68; BS70]. Roughly the same time, several techniques of capturing images with depth information have been developed (cf. survey of the stereo methods also starting about 1968 [BF82]).

So called *depth imaging devices*, sometimes also depicted as *range imaging devices*, are digital recording systems consisting of single or multiple digital cameras and possibly software post-processing not only capable of capture images of objects or scenes as in photography but also measure the distance between camera and the surface of recorded objects. So ideally, the output consists of two dense images, a gray scale or color image of the scene and a so called *depth map* with the measured distances for each pixel. Depending on the context, this map is also called *range image*, *surface profile* or *topographic map*. This information may be used to determine the 3D coordinates of the object surface pixels and visualize them as a 3D *point cloud* (*xyz* point list). But it is important to note that this kind of data is fundamentally different from volumetric data as recorded e.g. via *computer tomo-*

[1] Picture from the manufacturer's data-sheet [Fot15]

graphy (CT) where information about the whole 3D space is available, not only the object surfaces. So despite manufacturers promote depth imaging as "3D" technology, one may argue that the recorded mapping $\mathbb{R}^2 \to \mathbb{R}^3$ of a 2D pixel space to 3D point coordinates may be considered as "2.5D" technique.

During the last few years, several low cost depth imaging devices hit the consumer market, most notably the *Kinect* sensor by Microsoft in 2010. Although intended to be used with the video game console *Xbox 360*, the availability of an USB adapter and drivers to access the device via PC made it attractive for the computer vision as well as the robotics community and the field of human computer interaction. Based on the *structured light* approach [Bes88] which is also related to *stereo* triangulation it is a representative of a large class of devices with similar properties and artifacts arising during usage.

After the *Kinect* hype, recently the trend moved to depth imaging devices based on the *Time-of-Flight* (ToF) principle, where depth and intensity images may be recorded with one single camera and hence the calibration step to align the images of multiple sensors may be skipped. Especially the successor *Kinect 2* for *Xbox One* released 2014 is based on this principle. Recently, Lenovo and PMD Technologies announced the release of a smart phone ("phablet") with included ToF camera [PMD16] in the curse of 2016. This shows, that this technology is not only used for industrial applications but also present on the consumer market and available at low cost. Since the mentioned devices do not provide access to the recorded raw data, this work puts focus on the *CamCube* (V2 and V3) ToF-devices by PMD. These are of special interest because the captured raw data is accessible using the manufacturer's SDK and so may be processed and analyzed. For comparison, data recorded with a *Fotonic E40* device has been considered as well, but due to the lack of captured raw data, the presented methods could not be applied to this recorded data.

In this thesis, the combination of depth imaging and motion is considered. With the *Kinect* device, recorded sequences of moving objects like hands are used to apply motion estimation methods from 2D *optical flow* to the depth sequences which is called *range flow*. The artifacts in the depth maps caused by the *Kinect* principle need special attention and adaption of the *optical flow* algorithms.

Using ToF devices, the multiple exposures needed by this camera principle leads to strong artifacts when objects move in the time between. In the second part of this thesis, an overview of the existing methods dealing with these artifacts is given as well as a combination and enhancement of the different ideas to reduce or eliminate the artifacts.

1.1. Related Work

1.1.1. Range Flow Estimation on *Kinect* Data

To best of the authors knowledge, there has not been any publication on range flow estimation from *Kinect* before the conference paper [GFG11]. However, there have been several approaches to solve some of the algorithmic steps in the presented method independently. The open source driver [Mar10] is used to access and capture *Kinect* data. Calibration and alignment of color and depth channels have been addressed by [Bur]. The drawback of this method is, that the resulting data set still contains large areas of invalid values which results in poor flow estimation results (as shown later in Section 3.5). The proposed method for the computation of the actual range flow is based on ideas introduced by [SJB02], combined with ideas from [SRB10].

The conference paper [GFG11] has drawn a lot of attention in the community because it was one of the first papers at all dealing with this topic. Based on this work, Herbst et al. [HRF13] greatly improved the performance by application of more state-of-the art optical flow methods like using a multi-scale approach and more robust (Charbonnier) penalty terms. Recently, Jaimez et al. [Jai+15] took this even further by proposing a real-time primal-dual variant with total variation (TV-L^1) regularization.

1.1.2. Compensation of Motion Artifacts in ToF Data

To fix the motion artifacts arising in dynamic scenes, there are basically two kinds of methods. The first kind works on the (computed) *depth data* which is provided in some kind by all ToF devices or SDKs. Gokturk et al. [GYB04] assume all depth values at object borders to be influenced by motion artifacts and uses morphologic operations on foreground-/background segmented data to replace all depth values at foreground borders with spatially near valid ones. Regarding this type of methods it is difficult to fix artifact with the available depth data only, since in affected regions, all depth information is lost. Spatial/temporal near valid depth values are the only source of information so this is kind of an inpainting [Ber+00] or image restoration [GG84] problem to replace the artifact regions, which is itself an own field of research. This restoration is largely simplified since depth images are usually not as structured as the texture of objects and scenes in 2D images.

If it is possible to access the raw data captured by the ToF device, even in the case of motion artifacts, at least parts of these data are still valid so this is an additional source of information which can be used by the second kind of methods working directly on this *raw data*. Lottner et al. [Lot+07] combine edge detection of a high resolution 2D sensor to identify regions affected by artifacts with a reduction of used sub-exposures for depth computation within these regions. Hussmann and Edler [HE10] call this *pseudo-four-phase-shift algorithm*. The latter explicitly applies this method to difference images of the correlated data which may be extracted from a larger number of ToF cameras, even if the individual raw frames are not

available. This approach was reformulated more generally as *framerate increase* method by Schmidt [Sch11] where the imaging hardware problems also mentioned by Lottner et al. [Lot+07] as well as by Hussmann and Edler [HE10] (different tap A/B behavior, later described in detail) is handled by a dynamic linear calibration step. All methods mentioned so far have in common that motion artifacts are reduced (affected regions by a factor of three) but not eliminated.

In order to really remove the artifacts, it is possible to replace the invalid raw data with valid ones from the past as proposed by Schmidt [Sch11]. In his *standard* approach, the last recorded frame is considered as source for valid old data whereas the *burst internal detection* (BID) *approach* combines this with the *framerate increase* or pseudo-four-phase-shift method to consider only data from one single recorded frame. This correction fails, if there is more than one depth or brightness change (called *event*) within two consecutive frames (standard) or within a single frame (BID variant). Lee et al. [Lee+12; Han+12] propose artifact detection and correction working on tap difference images and the recent paper of Jimenez et al. [JPM14] gives a calibration of the discontinuity detection threshold, a combination of pseudo-four-phase-shift method extended by event detection with spatial neighborhood information as well as a real-time C++ CPU implementation.

Another important subset of methods working on raw data uses optical flow to compensate the motion occurring between the exposures. The specialized variant proposed by Hussmann et al. [HHE11] assumes one-directional motion of objects on a conveyor belt which may be detected by measuring the artifact stripe width and so may be implemented efficiently in hardware. Lindner and Kolb [LK09] apply the TV-L^1 dense optical flow method (GPU implementation by Zach et al. [ZPB07]) to image pairs of the sub-frames ((I_1, I_2), (I_1, I_3), (I_1, I_4)) and warp the following sub-frames to the first one to correct for lateral motion and then additionally apply a axial motion correction step by extrapolation the depth differences of the preceding frames. An extension of this method was proposed by Lefloch et al. [LHK13] reducing the number of image pairs for optical flow computation ((I_1, I_3) and (I_2, I_4)) to increase stability and real-time performance. Based on this work, Hoegg et al. [HLK13] propose a block matching (BM) variant of flow estimation restricted to artifact influenced regions to speed up computation to reach real-time performance with high frame rates or additional post-processing steps.

1.2. Own Contributions

Regarding the *Kinect* data, the main contribution is twofold: First, a novel channel alignment algorithm which largely reduces image areas without valid measurements compared to previous methods. Secondly, an extension of existing range flow approaches to cope with invalid and unstable depth estimates. The proposed methods are intended to be applied to data captured with the *Kinect* device, but should work in any multi-modal (RGBD) setting where different cameras are used to capture depth and color images. As stated above, this work was first dealing with this topic, triggering numerous extensions and improvements in the meanwhile.

Considering motion artifacts on *Time-of-Flight* (ToF) data, this work gives a first comparison of this class of methods and the algorithm implementations may act as a baseline for future research.

During the investigations, an improved non-linear cross-tap-calibration model has been developed. The usage of this new model leads to significantly improved motion compensation results in *all* presented methods. The proposed combination of the *framerate increase* with flow-based motion compensation gives improved results with less computational effort. This work includes a rigorous qualitative and quantitative evaluation on multiple real and synthetic ToF sequences.

By reverse-engineering of the computations in the black-box PMD SDK, it is possible to fix the computed depth values after application of any kind of post-processing the same way as the manufacturer, including correction of a systematic shift and depth wiggling. An automated rotor position detection pipeline revealed insight into technical details of the used recording hardware.

Publications Contributing to this Work

[GFG11] Jens-Malte Gottfried, Janis Fehr, and Christoph S. Garbe. "Computing Range Flow from Multi-modal Kinect Data." In: *ISVC (1)*. Ed. by George Bebis et al. Vol. 6938. Lecture Notes in Computer Science. Springer, 2011, pp. 758–767. ISBN: 978-3-642-24027-0. DOI: 10.1007/978-3-642-24028-7_70.

[GK12] Jens-Malte Gottfried and Daniel Kondermann. "Charon Suite Software Framework." In: *IPOL 2012 Meeting on Image Processing Libraries*. 2012. URL: http://www.ipol.im/event/2012_imlib/.

[Got+14] Jens-Malte Gottfried et al. "Time of Flight Motion Compensation Revisited." In: *IEEE International Conference on Image Processing 2014 (ICIP 2014)*. Paris, France, Oct. 2014.

Chapter 2.

Technical Background

In this chapter, an overview of the device principles considered in this work is presented. First, the structured light approach used by the *Kinect* is described in Section 2.1, then a background on *Time-of-Flight* (ToF) imaging including a description of the sources of motion artifacts is given in Section 2.2.

2.1. Structured Light Imaging

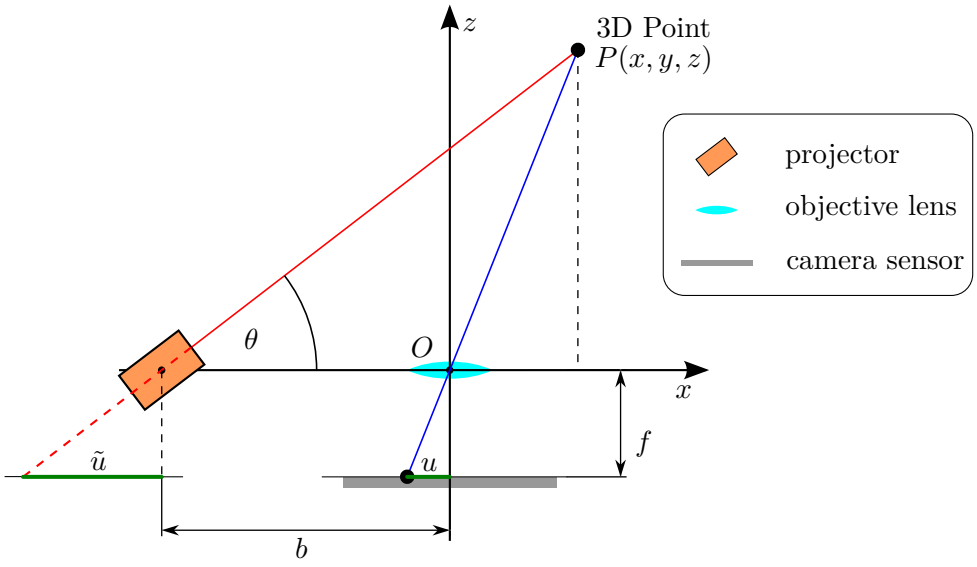


Figure 2.1.: Base principle of a structured light imaging device. The position of a projected point on the camera sensor depends on its z distance. Sketch based on [Bes88, Fig. 7], extended to visualize derivation of the inverse-disparity formula (2.1d).

Measuring the depth of a 3D point by triangulation is a well-known vision technique and may be implemented as a passive variant using multiple cameras (stereo vision, human eye) or by actively illuminating the recorded scene with a projector and recording it with a single camera. The latter is called *structured light imaging*.

A survey of the different variants developed in the 1970s and 1980s including projection of a single points or lines using a laser to multi stripe projection, binary patterns and random textures is given in [Bes88, Sec. 4].

Regarding a single projected point under a known projection angle θ, using the sensor coordinates (u, v) and a known focal length f of the camera (pin hole model) it is possible to reconstruct the 3D coordinates (x, y, z) of the point P as visualized in Figure 2.1:

$$\frac{x}{z} = \frac{u}{f} \Rightarrow x = z \cdot \frac{u}{f} \qquad \cot\theta = \frac{b+x}{z} \Rightarrow z\cot\theta = b + z \cdot \frac{u}{f} \tag{2.1a}$$

$$\Rightarrow z = \frac{b}{\cot\theta - \frac{u}{f}} = \frac{b \cdot f}{f \cdot \cot\theta - u} \qquad \Rightarrow x = \frac{b \cdot u}{f \cdot \cot\theta - u} \tag{2.1b}$$

Combining this with analogous terms in the orthogonal direction ($y = z \cdot \frac{v}{f}$) yields the triangulation formula given by [Bes88, Eq. 8]:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{b}{f \cdot \cot\theta - u} \cdot \begin{pmatrix} u \\ v \\ f \end{pmatrix} \tag{2.1c}$$

Introducing a virtual quantity $\tilde{u} = f \cdot \cot\theta$, i.e. the sensor position of the 3D point P of a virtual similar camera located at the projector position, the *disparity* d of the projected point may be computed as $d = \tilde{u} - u$ which yields using (2.1b)

$$z = \frac{b \cdot f}{f \cdot \cot\theta - u} = \frac{b \cdot f}{\tilde{u} - u} \qquad \Longrightarrow \qquad z = \frac{b \cdot f}{d} \tag{2.1d}$$

which is known as *inverse-disparity* formula used in *stereo-triangulation* (cf. [OK93, eqn. (1)] [Jäh05, eqn. (8.4)]).

The used Microsoft *Kinect* device uses a static pattern of projected points which seem to be distributed randomly (but repeated 3x3 by similar looking tiles). This pattern is visible in Figure 3.1 (bottom right). By using such a pattern covering the whole imaged area, it is possible to estimate the distance of all visible points with only one exposure. The (pseudo-) random structure is needed to identify the projected points and get their corresponding projection angles θ or rather the virtual value $\tilde{u}$ for disparity computation. This way an almost dense depth map of the recorded scene may be generated. But note that obviously, this methods fails in regions where foreground objects cast shadows since there are no points of the projected pattern there.

One of the reasons to choose this device was the availability of multiple software libraries to control the capturing process (adjust resolution and camera mode like infrared (IR) or color (RGB) exposure) and provide access to the captured data. The library actually used in this work is called *OpenKinect* [Mar10] and was the first library developed by reverse engineering of the transferred USB data. It was

published a few days after launch of the *Kinect* and one of the main reasons of the hype to use this device for a large variety of applications, especially in the computer vision, robotics and human-computer-interaction. Another promising project called *OpenNI* by the manufacturer *PrimeSense* of the recording unit used within the *Kinect* device provided a rich set of plug-ins including automated, real-time skeleton tracking of persons in front of the camera. This project was discontinued 2014 after an acquisition by another company. Some forks of the open source components are still present but the proprietary plug-ins (including the tracking module) are no longer available. Notably there is a software development kit (SDK) of the vendor *Microsoft* [MKS13] which was not used because it was not yet available at the beginning of this work and because of its proprietary license and limited target platforms (Windows only).

2.2. Time-of-Flight Imaging

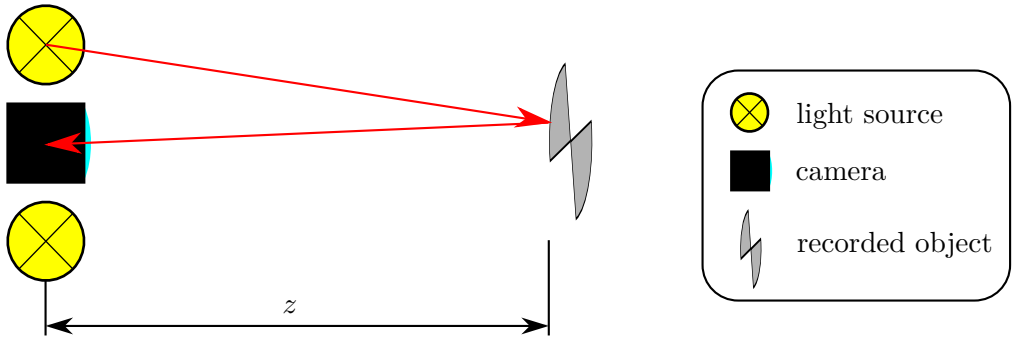
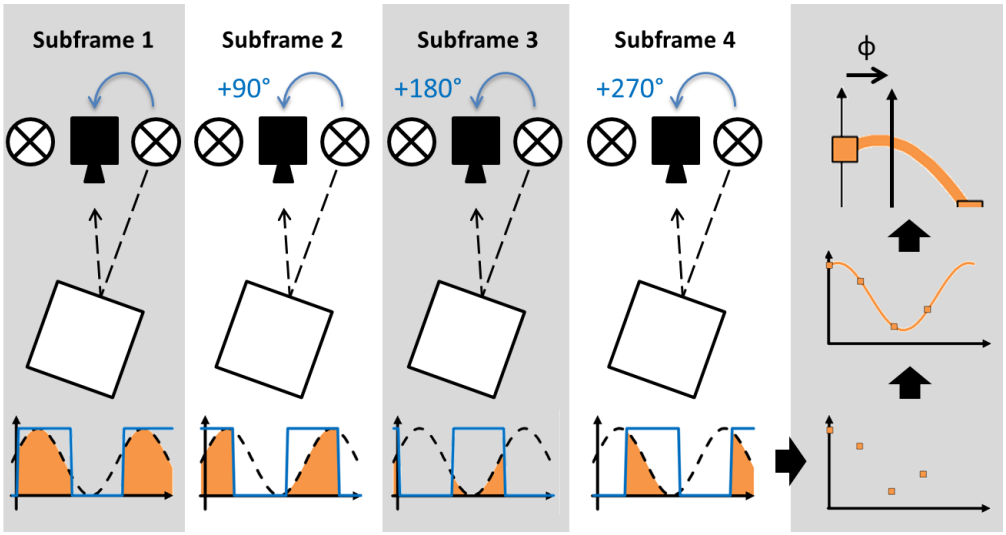


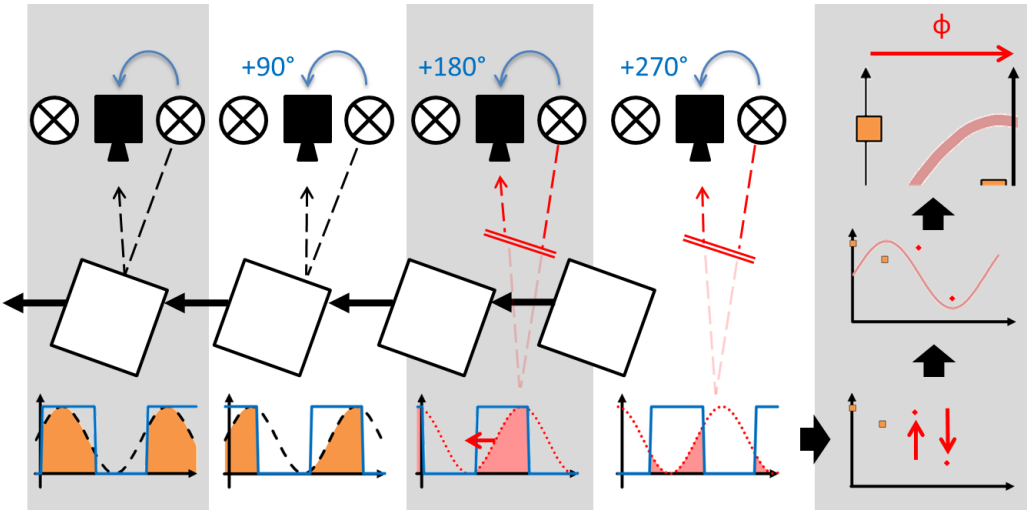
Figure 2.2.: Base principle of all Time-of-Flight imaging devices. Light is emitted from an IR light source illuminating the object surface. Scattered light is traveling back from the object surface to the camera. Surface distance is large compared to the distance between sensor and emitter(s), so in reality, the rays are almost parallel. Distance is computed from the measured Time-of-Flight spent travelling between emitter and sensor.

In contrast to the triangulation approach used by the *Kinect* device, another technique to estimate the distance to recorded objects is to measure the runtime of a light signal travelling from a light source to the object surface and back to the camera. Usually, the light sources are located nearby the camera, sometimes attached or even mounted in the same housing. Multiple light emitters arranged on all sides of the camera reduce casting shadows. Measuring this round-trip-time and using the known speed of light to retrieve the actual distance is called *Time-of-Flight* (ToF) imaging, visualized in Figure 2.2.

In this work, *continuous wave intensity modulation* (CWIM) sensors are considered, as virtually all current ToF devices use this technique. The basic working principle of this kind is depicted in Figure 2.3. It shows the phase shift estimation



(a) Standard Case



(b) Error Case

Figure 2.3.: Working principle of a ToF camera. Each recorded frame consists of four sub-frames (first four columns). For simplicity only one single tap is shown here. The light source emits an intensity-modulated signal which is reflected back to the sensor pixel (black and red dashed lines) with a phase shift ϕ. This phase shift is determined by measuring the correlation (bottom row colored areas) between the incident and the reference signal shifted by a fixed phase angle (blue line). The correlation function (a sine-wave) is fitted to the measured values (right column). The sub-frames are acquired sequentially. Hence, if the depth changes between two sub-frames (lower figure), wrong correlation values are obtained (last two columns). The standard reconstruction formulas using these measurements lead to an erroneous estimation of ϕ.

by correlation together with an example how moving objects cause wrong results. Other sensors e.g. use a pule-based principle with flash-like light sources or even more sophisticated patterns of the emitted light.

CWIM ToF devices measure the phase shift between a reference sinusoid signal and the incident reflection of light emitted by an intensity modulated light source. The brightness of this light source is modulated with the same reference signal. This is done by sampling the correlation function between incident and reference signal on at least four different phase shifts by recording four so called *raw-frames* containing the correlation results.

Modern two-tap sensors measure two of these raw-frames during one single exposure (sub-frame) in the following way [Sch+95; Spi+95]: Each pixel of such a sensor collects the photo-electrons not only in one single potential well like usual CCD sensors, but in two separate ones, called *taps* here. By applying an electric field the photo-electrons may be deflected into one specific tap. Modulating this electric field with a rectangular version of the sinusoidal reference signal, each pixel samples the correlation of the incoming light signal ejecting the photo-electrons and the reference with one tap (and its negative in the second tap) during the full integration time and thus for a large numbers of periods ot the modulation frequency (modulation period e.g. 50 ns vs. integration times in a range from 100 µs up to 4 ms). Since the electrons do not vanish, summing up the numbers of both taps yields an intensity image as given by usual CCD sensors. Yet, as the taps usually have different response curves [Erz11] (cf. Figure 4.2), usually two more (redundant) sub-frames are recorded (Table 2.1). Individual sub-frames are acquired in a sequential way. Motion artifacts occur when the static scene assumption between sub-frame exposures does not hold. Examples for such artifacts are depicted in Figure 4.1.

2.2.1. ToF Depth Estimation

With the four correlated samples of the returning light $\{I_0, I_{90}, I_{180}, I_{270}\}$, it is possible to reconstruct the *Brightness Image* I_B (intensity), the *Amplitude Image* I_A and the *Phase Shift* φ of the incoming modulated light (cf. [Spi+95; LK09; Sch11]):

$$I_A = \sqrt{(I_{270} - I_{90})^2 + (I_{180} - I_0)^2} \tag{2.2}$$

$$I_B = \frac{1}{4}(I_0 + I_{90} + I_{180} + I_{270}) \tag{2.3}$$

$$\varphi = \operatorname{atan2}(I_{270} - I_{90},\ I_{180} - I_0) + \pi \tag{2.4}$$

Using the phase shift φ, the radial distance r between device and a recorded object surface is computed by

$$r = \varphi \cdot \frac{c}{4\pi\nu_M} \tag{2.5}$$

Since there are two taps (A/B) on the recording chip, there are eight correlated raw images and hence several options what to use as I_k in the formulas above:

single-tap $I_k = A_k$ (tap A only) or $I_k = B_k$ (tap B only)

average $I_k = (A_k + B_k)/2$

subset use S1 $= \{A_0, A_{90}, B_{180}, B_{270}\}$ or S2 $= \{B_0, B_{90}, A_{180}, A_{270}\}$

Note that the manufacturer's SDKs usually use the average approach of depth estimation as shown later. In order to gain an impression how this choice influences the estimated depth images, see Figure 4.8 and Figure 4.10.

time	t_0	t_1	t_2	t_3
tap A	A_0	A_{90}	A_{180}	A_{270}
tap B	B_{180}	B_{270}	B_0	B_{90}

Table 2.1.: Overview of the eight captured raw-frames. Index denotes the phase shift during correlation.

frame	Capturing process of all images needed for ToF depth estimation, usually this contains four exposures.
raw-frame	Recorded data of one tap during one exposure.
sub-frame	All data recorded during one exposure, with a two-tap device this contains two raw-frames.
subset	Certain selection of at least four raw-frames (out of the usual eight available) used for depth estimation. The different variants are depicted in Subsection 2.2.1.

Table 2.2.: Dictionary of used terms to describe ToF data

Chapter 3.

Structured Light Sensor: Multi-Modal *Kinect* Data

Figure 3.1.: *Kinect* device and its raw data output. **Top:** Kinect device with projector (A), color cam (B) and IR cam (C); color and depth channel overlayed. **Bottom** (left to right): raw color image; pseudo-color coding of the 11bit depth image provided by the on chip depth estimation; raw IR image showing the projected point pattern.

3.1. Introduction

This chapter, which has been published as conference paper [GFG11], contains a framework for the estimation of range flow fields from multi-modal (color and depth, RGBD) video sequences captured by *Kinect*. The computation of optical flow [Bak+07] from 2D image sequences or range flow [SJB02] from depth image sequences plays an important role in the middle layer of a wide range of computer vision algorithms, such as object tracking, camera motion estimation or gesture recognition. Flow estimation has been investigated for a long time and many sophisticated algorithms (especially for optical flow) have been introduced so far [Bak+07].

However, due to the *Kinect* technology, standard range flow algorithms cannot simply be used "out of the box" – which makes the computation of range flow fields from this device a non-trivial task. The main difficulties result from the following two facts: First, *Kinect* provides only uncalibrated data where color and depth channels

(which are recorded by different cameras) are not aligned. Second, the depth channel may contain large areas of invalid values and that edges in the depth-map are not always stable as visible in Figure 3.1.

The remainder of this chapter is organized as follows: Section 3.2 provides a discussion of *Kinect's* depth imaging technology and points out the main problems that have to be solved in order to use the raw data for range flow estimation. Section 3.3 introduces the calibration process and a novel algorithm for the alignment of depth and color channels, before the actual flow algorithm is discussed in Section 3.4. Finally, results and experimental evaluations are presented in Section 3.5.

3.2. A Brief Introduction to the *Kinect* Imaging Hardware

Kinect's depth imaging device is based on a *structured-light* illumination approach [Bes88]. The range information is estimated from the distortion of a projected point pattern which is captured with a camera placed at a certain baseline distance from the projector.

Figure 3.1 shows the hardware setup: *Kinect* uses an IR laser diode to project a fixed point pattern which is invisible to the human eye. In combination with an IR camera, the estimation of the depth-map is computed directly in hardware at approximately 30 frames per second at VGA resolution[1]. The depth resolution is approximately 1cm at 2m optimal operation range [Mar10].

The main advantage of the *Kinect* concept is, that it allows a more or less dense depth estimation of a scene even if it contains objects with little or no texture. Additionally, a second camera captures VGA color images from a third position.

3.2.1. Limitations of *Kinect* Data

The main problem, at least from a computer vision perspective, is the computation in hardware which is neither user accessible nor can it be circumvented altogether. Hence, the entire process is a "black-box", with very little publicly known details on the obviously extensive post-processing. It most likely includes some sort of edge preserving smoothing and up-scaling of the depth-map. This has significant impact on the noise characteristics and correlations.

The provided depth estimation is more or less dense, but there is a systematic problem common to all structured-light approaches which use a camera offset: there are regions where the projected pattern is shadowed by foreground objects – making it impossible to estimate the depth at these positions (see Figure 3.1, depth and **!** image). Another problem is that depth values tend to be unstable and inaccurate at object boundaries. This is caused by the fact that the dense depth-map is interpolated from discrete values measured at the positions of projected point patterns.

[1] Note that VGA resolution is probably a result of the extensive on-chip post-processing. The true hardware resolution is bounded by the number of projected points, which is much lower than VGA.

Finally, since there are two different cameras capturing color and depth images, these images are not necessarily aligned to each other (Figure 3.1 top right). In addition, the two cameras have slightly different focal length and the optical axes are not perfectly parallel.

3.3. *Kinect* Calibration and Data Alignment

Proposing a multi-modal flow algorithm, it is essential that the color and depth image information at a certain image location belong to the same object point. For this, a novel alignment algorithm which is especially suitable for *Kinect* data is introduced here.

It is based on previous methods by [Bur], but performs a more complex inverse mapping from the depth image onto the color image – whereas [Bur] uses the straight forward mapping of the color image onto the depth image. The advantage of this proposed method is, that it is still possible to compute at least the xy-components of the optical flow for areas with invalid depth values, whereas all information is lost in such areas if one applies the original alignment approach by [Bur].

3.3.1. Camera Calibration

In a first step, a stereo-calibration of the cameras is needed. For this task a standard checkerboard target with good infrared (IR) reflection properties and extra illumination in the IR spectrum is used followed by a standard stereo-calibration procedure as provided by [Bra+00] (process similar to [Zha99]). To avoid influences of the projected points, the projector has to be closed e.g. by covering it with black tape. With the parameters estimated during this procedure it is possible to compute epipolar images from the IR and RGB cameras, i.e. images corrected in a way that objects visible in both images are located at the same y-position and show only a shift in x-direction (disparity) depending on the z-distance of the sensors. Especially different focal length and misalignment of the optical axes are eliminated as well as some distortions caused by deviations of the ideal projection properties of the used lenses.

3.3.2. Data Alignment

As discussed at the *OpenKinect* driver's documentation[2] the *Kinect* does not compute real z-depth values on the device but provides a map of raw disparity values d_{raw} which may be computed from the shift of the projected point pattern. Several different methods are proposed for computing z-values from this raw disparity map. To avoid this issue, for the rest of this chapter, "depth image" $Z(x, y)$ is used synonymously for these raw disparity images. If necessary, computing real z-values could be implemented as a post-processing step of the output generated by the methods presented here.

[2] `https://openkinect.org/wiki/Imaging_Information`

The actual data alignment algorithm is based on the assumption that these raw disparity values d_{raw} are linearly correlated with the point-wise disparity d between pixels in the color image and their corresponding pixels in the raw disparity image. A principal component analysis (PCA) over the positions of the checkerboard corners from the calibration process is used to obtain a linear approximation $d = a \cdot d_{\text{raw}} + b$ of this dependency (cf. Figure 3.2).

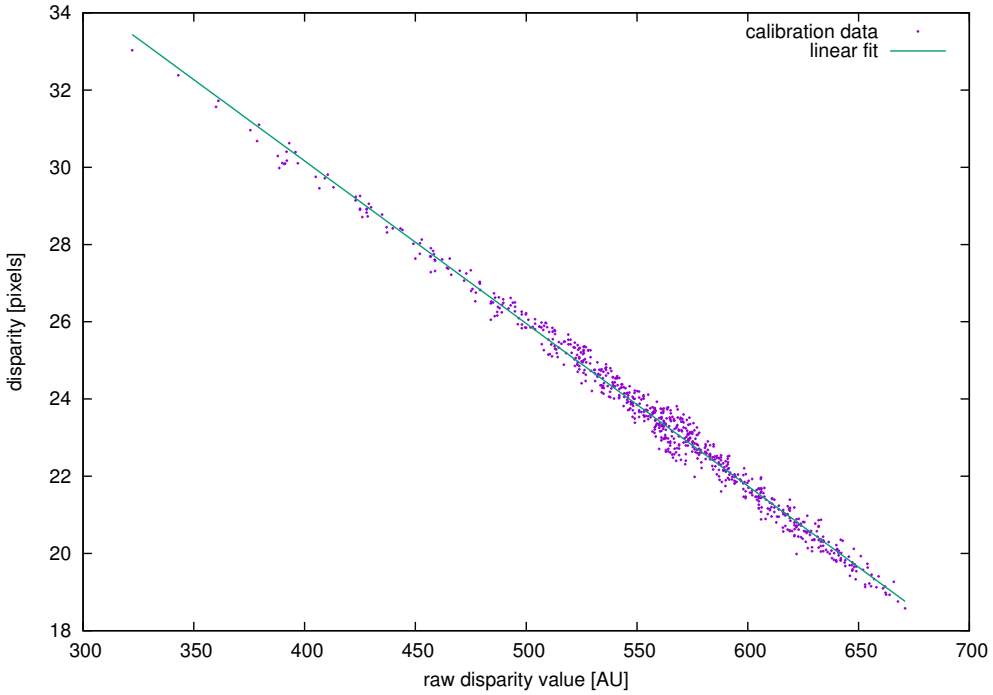


Figure 3.2.: Relation between raw disparity values given by the *Kinect* device and the measured disparity of the checkerboard corners in epipolar images after the stereo-calibration step. Each data point represents one checkerboard corner. As visible, the assumption of a linear dependency holds.

The original approach in [Bur] shows, that it is easy to warp the color image $I(x, y)$ such that it is aligned with the disparity image $Z(x, y)$ by use of the disparity field $D(x, y) = d(Z(x, y))$:

$$\tilde{I}(x, y) = I(x + D(x, y), y) \tag{3.1}$$

where $\tilde{I}(x, y)$ is the warped color image. Since $x+d$ may be fractional pixels, one has to interpolate the integer pixel values of I. For regions, where $Z(x, y)$ has no valid depth value, computation of $\tilde{I}$ is impossible. Such regions are clearly visible e.g. at the shadow of the hand shown in Figure 3.1. They have to be marked, e.g. setting the color to black. As a consequence, the color image information in these regions is completely lost. To avoid this, it is necessary not to modify the color image, but to invert the mapping and align the depth image to the color image. This is done

by computing the *inverse disparity map* $D^*(x, y)$ such that

$$D(x, y) = -D^*(x + D(x, y), y) \tag{3.2}$$

This problem is similar to the computation of the *inverse optical flow* $\vec{h}^*$ as proposed in [SS06, appendix A]. Because the disparity is known to be limited to the x-direction only (i.e. the y-component of the optical flow field $\vec{h}$ vanishes) a simplification of this approach may be used to reformulate the ideas of [SS06] (for the i-th pixel in x-direction, i.e. at position (x_i, y)) as

$$D^*(x_i, y) = -\frac{\sum_j D(x_j, y)\ p(x_i, x_j + D(x_j, y))}{\sum_j p(x_i, x_j + D(x_j, y))} \tag{3.3}$$

where the weighting function p is computed using

$$p(x, x') = \max\{0, r - |x - x'|\}. \tag{3.4}$$

The radius r specifies the region of influence of each pixel. Invalid depth values $D(x_j, y)$ have to be excluded from summation. For some target positions, the denominator can become very small (i.e. if no depth values would be warped to this position). In this case $D^*(x_i, y)$ is marked as invalid as well. Using D^* the alignment of the depth map $\tilde{Z}(x, y)$ to $I(x, y)$ is computed in a similar way as before in eqn. (3.1):

$$\tilde{Z}(x, y) = Z(x + D^*(x, y), y) \tag{3.5}$$

Figure 3.3 shows a qualitative comparison of the results of the proposed method and the pure stereo calibration approach.

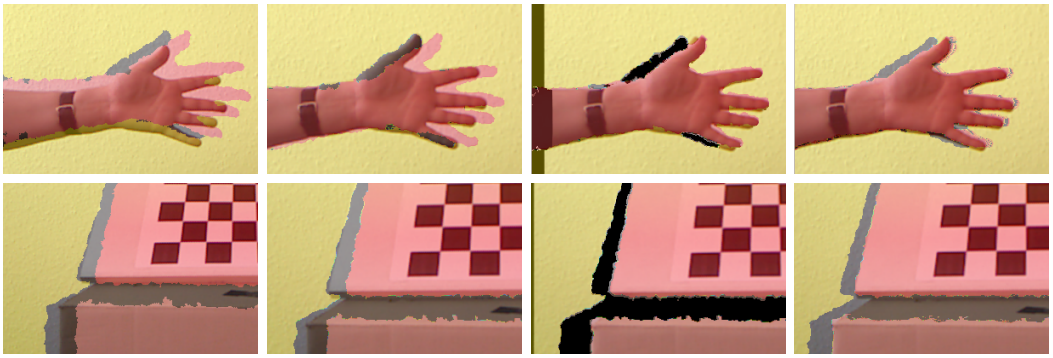


Figure 3.3.: Comparison of calibration results. In gray overlay regions, the depth values are invalid. In black regions, there are neither valid color nor depth values.
Columns (left to right): uncalibrated; pure stereo calibration (epipolar); aligned warping color image with D; aligned warping depth image with D^*

Another approach to align the data could be a reprojection of the 3D data points given from the IR camera using the projection matrix known from the calibration step. As mentioned above, this would require a proper method to compute real z values out of the given raw disparities which has been avoided here completely.

3.4. Range Flow

Range flow (syn.: *Scene flow* [Ved+99]) is the established term for the 2.5D extension of *optical flow*, describing the local 3D motion in an image sequence. Mathematically, the range flow field is a 3D vector field $\vec{h}_R$ which is defined on a 2D image plane, i.e.

$$\vec{h}_R : \mathbb{R}^2 \mapsto \mathbb{R}^3 \qquad \vec{h}_R(x, y) = (u, v, w)^T \qquad \vec{h}(x, y) = (u, v)^T, \tag{3.6}$$

where the first two components (u, v) are identical to the motion description in standard 2D optical flow $\vec{h}$ and w encodes the motion in the depth direction.

Numerous algorithms have been proposed in literature to solve the standard *optical flow* (u, v) (e.g. see [Bak+07]). For the sake of simplicity, the proposed method is based on a standard method for global *optical flow* [SRB10] (which it self is a reinterpretation of the classical flow paper by Horn and Schunk [HS81]). It should be noted that the proposed algorithm (see section 3.4.1) should also work with most other global methods (like [Pap+06] or [Sun+08]). As most of the "more advanced" techniques focus on increased sub-pixel accuracy, which is not very likely to be a useful property given the accuracy of real world *Kinect* data a survey of the different methods was out of scope of this chapter.

The method by [SRB10] applies a pixel-wise brightness constancy assumption called *optical flow constraint* (OFC) (or BCCE) that may be formulated as

$$\nabla I \cdot (u, v)^T + I_t = 0 \Leftrightarrow (I_x, I_y, 0, I_t) \cdot (u, v, w, 1)^T = 0 \tag{3.7}$$

where I is the 2D image data (here: color image converted to gray-scale) and the indices denote derivation with respect to the specified variable.

As proposed by [SJB02], a similar term may be formulated for the depth data Z, adding the motion in depth direction w:

$$\nabla Z \cdot (u, v)^T + w + Z_t = 0 \Leftrightarrow (Z_x, Z_y, 1, Z_t) \cdot (u, v, w, 1)^T = 0 \tag{3.8}$$

This equation is called *range flow motion constraint* (RFMC). As one can see, the 2D terms in eqns. (3.7) and the depth term in (3.8) are based on the same principle. Using this fact, range flow estimation using color (converted to gray-scale) images and depth data may be performed using any optical flow estimation algorithm with an additional data term incorporating eqn. (3.8).

3.4.1. Robust Flow Estimation

The multi-modal data which is computed using the proposed data alignment algorithm (see Subsection 3.3.2) has a dense color channel. Still, there can be invalid values in the depth channel and artifacts at object borders. Both artifact types do not remain constant in time, resulting in estimated depth changes even if there is no motion.

Therefore it is essential to exclude these regions for a robust flow computation. Regions with invalid depth values may be recognized by simply thresholding the

depth channel (in the raw depth output, these pixels are marked by the hardware with a depth integer value of 2047 = `0x7FF`, i.e. the largest 11 bit integer value). Object borders may be recognized by thresholding the edge strength $(Z_x^2 + Z_y^2)$. If at least one of these threshold conditions is met, pixels are excluded from the RFMC data term (3.8), i.e. only the color image data is used for computation at this position.

Since the linear filters used to compute the derivatives of the depth image usually have a width of 3 pixels (Sobel or Scharr filters [Sch00]), this exclusion region has to be extended, e.g. using the morphologic dilation operator. A radius of 2 pixels showed to be sufficient. In the excluded regions, the value of w is interpolated from the valid neighbors by regularization of the range flow field.

Strong regularization leads to smooth flow fields but also causes blur effects at motion edges. Since motion edges often correspond to edges in the depth image, estimation results can be improved further by using the exclusion mask. Using strong regularization in valid and weak regularization in excluded regions yields much sharper motion edges and separation between fore- and background motion. This adaptive regularization is another benefit of using the depth image information.

3.4.2. Algorithm Summary

The final implementation of the proposed method is realized in a standard pyramid scheme with two levels of iterations. The outer iteration implements the multi-scale image pyramid. The inner iteration (line 10 of Algorithm 1) recomputes the flow increments on the given input image pairs (I_1, I_2, Z_1, Z_2), where the second has been warped with the flow $\vec{h}_R^0$ computed during the previous iteration.
This is done by minimizing the following energy functional:

$$\iint \Big[(I_x u + I_y v + I_t)^2 + \lambda_Z(x,y)(Z_x u + Z_y v + w + Z_t)^2 + \lambda_R(x,y)(|\nabla u|^2 + |\nabla v|^2 + |\nabla w|^2)\Big] \mathrm{d}x\ \mathrm{d}y \quad (3.9)$$

where

$$\lambda_Z(x,y) = \begin{cases} c_z > 0 & \text{where } M = 0 \\ 0 & \text{else} \end{cases} \qquad \lambda_R(x,y) = \begin{cases} c_{R1} > 0 & \text{where } M = 0 \\ c_{R2} > 0 & \text{else} \end{cases} \quad (3.10)$$

with $c_{R1} \gg c_{R2}$.
Using weak regularization in invalid regions (where $M = 1$) may be counter-intuitive. As stated above, the mask M is also used as edge detection so weak regularization in invalid regions leads to sharper motion borders on edges.

3.5. Evaluation

Unfortunately, there is no publicly available data base with ground truth range flow fields for *Kinect* data, as this is the case for 2D optical flow (e.g. as provided by

Algorithm 1 Final *Kinect* Range Flow Algorithm

for all multi-modal image pairs I_1, I_2, Z_1, Z_2 **do**
  SCALE I_1, I_2, Z_1, Z_2 and $\vec{h}_R^0$ accordingly
  for all flow iterations **do**
    WARP I_2, Z_2 with $\vec{h}_R^0 \rightarrow \tilde{I}_2, \tilde{Z}_2$
    if Z_1 or $\tilde{Z}_2$ invalid **then**
      SET exclusion mask $M = 0$
    else
      SET exclusion mask $M = 1$
    end if
    COMPUTE FLOW $\vec{h}_R$ using OFC (3.7) and RFMC (3.8) data terms with strong regularization where $M = 1$, but OFC only with weak regularization where $M = 0$.
    APPLY MEDIAN FILTER on $\vec{h}_R$ to suppress outliers
    SET $\vec{h}_R \rightarrow \vec{h}_R^0$
  end for
end for

[Bak+07]). Hence, no generally agreed and sufficiently accurate methodology for quantitatively analyzing such algorithm is available. Therefore only a qualitative discussion of the results is given.

3.5.1. Results

Qualitative results

Range flow estimation results are shown in Figure 3.4. A moving hand sequence has been recorded for quantitative evaluation. The rows in this figure represent different algorithm configurations, i.e. combinations of optional usage of the region validity masks and usage of the forward or inverse disparity (D or D^*) for warping.

The first column shows the first frame of the image pair as used for range flow estimation. In the second column, the first two components of the range flow result $\vec{h}_R$, i.e. the standard optical flow field $\vec{h}$ is visualized. The distance between the arrows is 16 pixels, an arrow length of 5 pixels corresponds to a flow magnitude of 1. For better visualization of the flow contours, a HSV representation using flow angle as hue and flow magnitude as saturation has been drawn in the background. The third column shows the depth dimension of $\vec{h}_R$, i.e. w. The last column shows the raw depth data with exclusion mask. Since the depth value of invalid depth pixels is `0x7FF`, these pixels appear bright. Note that the exclusion mask also consists of edges in the depth image, the contours of the hand are reproduced well.

Since strong regularization is applied within connected valid depth regions, the flow result in the first two rows is smooth over the hand area as well as in the background. The hand contours are reproduced well, only low blurring effects at the borders are visible. Without usage of the masks, outliers as well in the (u, v) as

in the depth channel w show up as visible in the last two rows.

The algorithms have not been tuned to be highly parallelized or using GPU computing, so the runtime is currently about half a minute per frame pair. Significant speed improvements are expected from such an optimized implementation.

For testing, a very simple sequence that simulates the targeting application area of gesture recognition has been used. Future research should focus on generating more realistic test sequences with given ground truth such that quantitative analysis becomes possible.

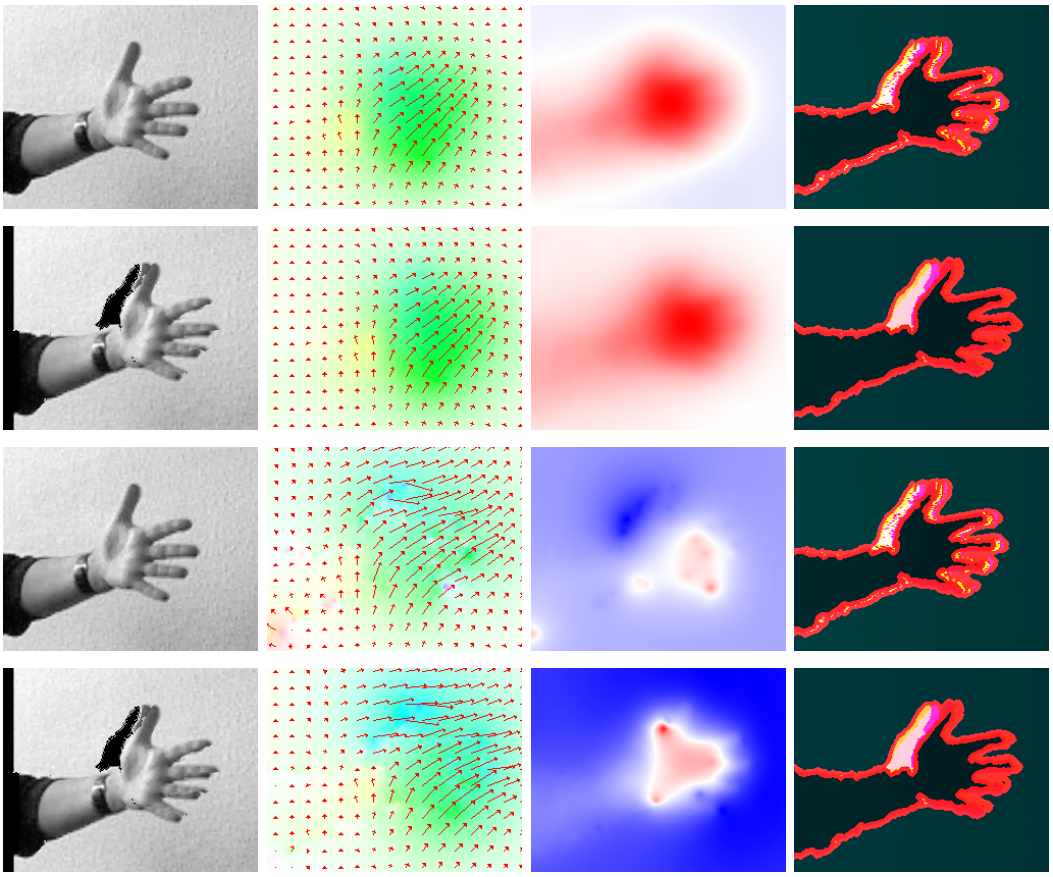


Figure 3.4.: Range flow estimation result of an image pair of a moving hand sequence.
Rows (top to bottom): warp depth with D^*, using masks; warping color with D, using masks; warp depth with D^*, no masks; warping color with D, no masks; cf. Algorithm 1
Columns (left to right): gray image; optical flow $\vec{h}$ (HSV visualization with quiver); range flow depth component w (blue: positive, red: negative values); depth image with exclusion mask (red/bright=excluded)
The exclusion mask is shown even if it has not been used to compute the flow.

Chapter 4.

Time-of-Flight Data

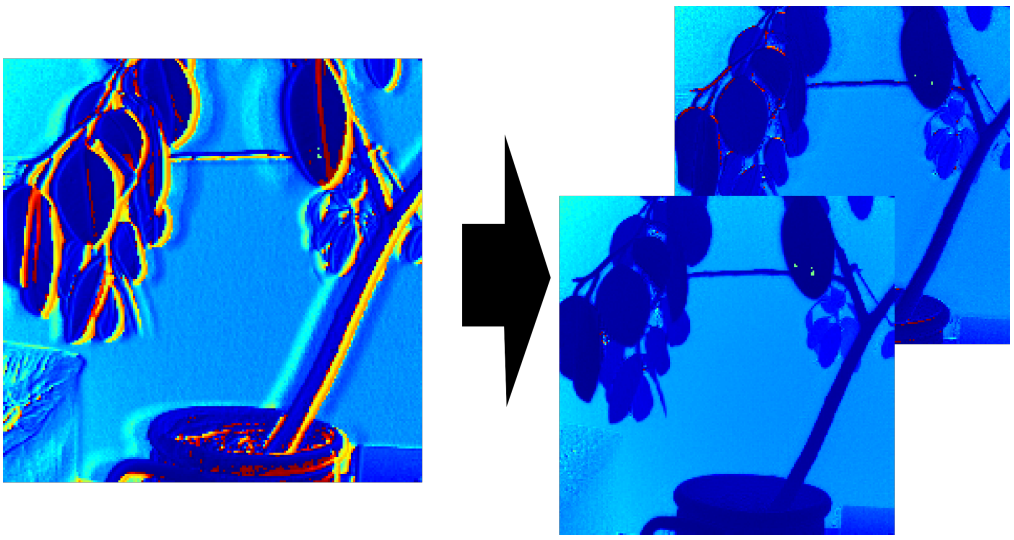


Figure 4.1.: Example depth map with motion artifacts (left) and results of presented methods (Schmidt's BID method (right, upper), Lindner&Kolb (right, lower)).

4.1. Introduction

Today, Time-of-Flight (ToF) imaging is a mature technology, being frequently deployed in industrial applications such as optical inspection, robot control and surveillance. With the new generation of low-cost sensors such as *Microsoft's Kinect 2* it has also hit the mass consumer market and therefore impact human computer interaction, artistic expression and citizen sciences. Yet, like any other depth imaging modality, ToF data does have its own set of issues such as flying pixels, depth wiggling, multi-path and motion artifacts. In this chapter motion artifacts and existing methods designed to deal with them are of specific concern. These artifacts occur in dynamic scenes due to the sequential nature of the measurement process and are a problem inherent to all current ToF cameras.

This is the first comparison of this class of methods and the implementations given here may act as a baseline for future research.

By identifying common building blocks used in many of the methods, it is possible to implement them in a modular way. Thus, this framework is not only a benchmark of the proposed methods but may also be used for evaluation of the sensitivity of the proposed pipeline towards the choice of different subcomponents (e.g. the optical flow algorithm some systems use).

Finally, during the investigations, an improved cross-tap-calibration model has been developed. The usage of this new model leads to significantly improved motion compensation results in *all* presented methods.

Parts of this work have been published in a conference paper [Got+14]. Compared to the paper, this chapter is enhanced by much more extensive experiments including real and synthetic test sequences, a quantitative evaluation and finally a decision guide for users on how to select the algorithm appropriate for the application in Figure 5.1. Additional details e.g. comparison of the optical flow result using a variety of optical flow methods are given in Appendix A.

The rest of this chapter is organized as follows: Since the technical background of ToF imaging was described above in Section 2.2, this chapter continues with the needed preprocessing steps including a novel inter-tap calibration in Section 4.2. Since this is common to all subsequent methods it needs special attention. Detailed descriptions of the considered motion compensation methods are given in Section 4.3. During the experiments, some issues of wrong depth values computed using the standard formulas have been discovered which is described and solved in Section 4.5. As a prerequisite for the quantitative evaluation, an automated rotor position detection is presented in Section 4.6 which is also used to discover some of the technical properties of the used *CamCube3* device in Section 4.7. Finally an evaluation and comparison of the motion compensation methods is given qualitatively in Section 4.4 and quantitatively in Section 4.8.

4.2. Preprocessing and Calibration

Before discussing the performance of the *framerate increase*, *detect and repair* and *flow based* motion compensation methods on real and synthetic datasets in the next section, it is important to first revisit the preprocessing steps needed by most of the algorithms, i.e. tap calibration and image homogenization. As the linear fitting approach proposed in literature did not work well on the captured datasets with the used device, a non-linear fitting approach which shows improved results on all compensation methods has been developed. Tap calibration is also a prerequisite for *framerate increase*. During the experiments, a *CamCube3* device by PMDTechnologies has been used. The emitted infrared light is modulated with an adjustable frequency around $\nu_M = 20\,\mathrm{MHz}$ resulting in a non-ambiguity range

$$R_N = \frac{c}{2\nu_M} \approx \frac{3 \cdot 10^8 \,\frac{\mathrm{m}}{\mathrm{s}}}{2 \cdot 2 \cdot 10^7 \,\frac{1}{\mathrm{s}}} = 7.5\,\mathrm{m} \tag{4.1}$$

Objects further than R_N cause the measured phase to wrap around 2π causing ambiguities in the measured distance. Exposure times may be adjusted between $t_{\min} = 10\,\mu s$ and $t_{\max} = 4\,ms$, the modulation frequency ν_m may be selected from the values $\{18\,MHz, 19\,MHz, 20\,MHz, 21\,MHz\}$ resulting in slightly smaller or larger values of R_N. Using different frequencies it is possible to work with multiple *CamCube* devices on the same scene simultaneously or to disambiguate the measured depth of objects with a distance larger than R_N.

4.2.1. Tap Calibration

In the subsequent methods, it is essential that the two record units (*taps*) of each pixel behave similarly. Especially the *framerate increase* does not work at all without this property since it mixes values from both taps to estimate the phase shift of the returning light. Without any calibration one would take samples out of two different sine functions (with different offsets and amplitudes) leading to wrong phase estimation results. So as a first step the photo response of the taps of each pixel is investigated.

As stated by [Sch11], one should expect a linear dependency between the responses of the two taps of an individual pixel. To check this, an *exposure ramp* of a flat white wall has been recorded. The camera was mounted fixed to assure a static scene. Integration times were chosen in a range from the lowest possible value to saturation of the center pixels (0.1 ms to 3 ms in steps of 0.1 ms). To avoid random noise, the average of 128 exposures with the same integration time has been computed. As visible in Figure 4.2, the assumption of linearity does not hold for low intensities. Pixel values are given in digital units (DU), i.e. the integer values as provided by the device. The pixel shown at the left side shows a strong curvature in the raw pixel values below $1 \cdot 10^4$ DU. For medium and high intensities, the behavior is quite linear. At some other pixels, the curved region extends over the whole value range as visible at the right plot. One may argue that the regions of non-linearity are small compared with the intensity value range. As visible in Figure 4.3 (left), estimated phases in regions of low intensity cover almost the whole value range after linear correction using r_1. This shows that under the condition of low intensities, this linear method is not able to provide stable results.

To overcome this issue, a new reconstruction formula is proposed to cope with nonlinear behavior at low intensities but linear extrapolation at high intensities. A polynomial fit of higher degree is able to adapt the curved data ($d = 5$ showed to be a good choice) whereas a linear fit has good extrapolation properties. The final reconstruction formula is constructed by using a combination:

$$r_c(b) = \Theta(b) \cdot r_1(b) + (1 - \Theta(b)) \cdot r_5(b) \tag{4.2}$$

with the polynomials r_d of degree d and a sigmoid switching function Θ:

$$r_d(b) = \sum_{k=0}^{d} \alpha_k \cdot b^k \qquad \Theta(b) = \frac{1}{2}\left(\operatorname{erf}\left(\frac{b-\mu}{2\sigma}\right) + 1\right) \tag{4.3}$$

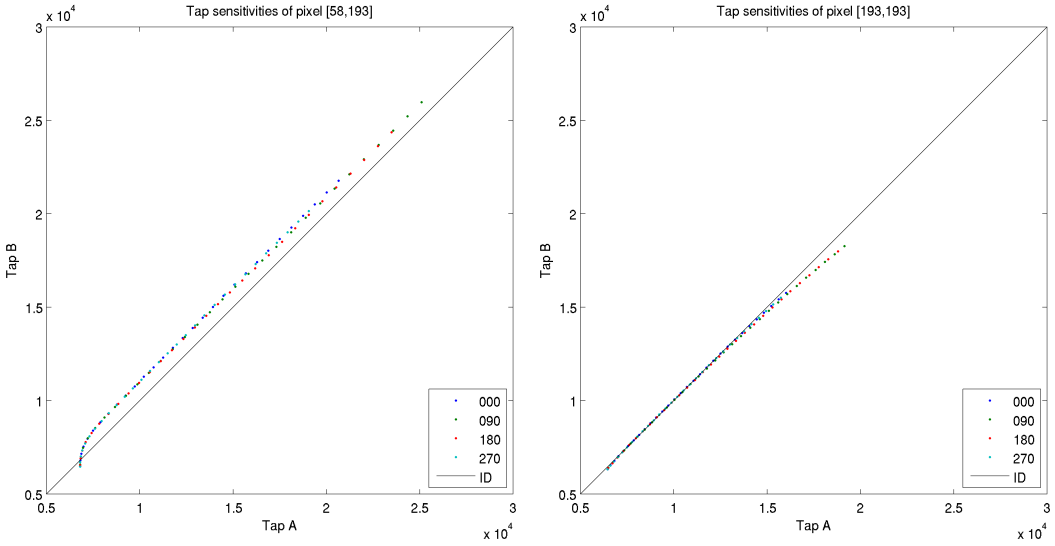


Figure 4.2.: Comparison of photo response of the different taps of two individual pixels. Each dot represents a single exposure, the colors encode the sub frames with different phase shifts. Values in *digital units* (DU) as given by the device. The left plot shows strong non-linear behavior at low intensities, the right plot shows curvature over the whole value range. So linear calibration is not sufficient.

Θ generates a smooth transition between the fitted linear and polynomial functions with the following properties:

$$\Theta(x \ll \mu) = 0 \qquad \Theta(x \gg \mu) = 1 \tag{4.4}$$

The transition region where values of r_1 and r_5 are mixed may be adjusted via the parameter σ. Since the central pixels get saturated at large integration times, the value range below $1.8 \cdot 10^4\,\mathrm{DU}$ has been used for fitting. Higher values cannot be trusted for all pixels. The parameters of the transition function Θ were chosen such that r_5 is used for interpolation within the fit range and r_1 for extrapolation:

$$\mu = 1.5 \cdot 10^4\,\mathrm{DU} \qquad \sigma = 100\,\mathrm{DU} \tag{4.5}$$

The effect of applying the reconstruction r_c to the pixel values of tap B is shown in Figure 4.4. Since the phase is proportional to the radial distance, the flat wall generates a pattern with circles of equal radial distance from the camera. The estimated phase of tap B after application of r_c looks the same as the reference tap A. This shows that reconstructing the behavior of tap A in the other taps works as expected. At the backside, this also causes artifacts of tap A to appear also in tap B like the horizontal line structures which are slightly shifted and more weak in the original tap B picture.

Regarding the averaged phase using the raw pixel values of tap A and B together, application of the reconstruction r_c to the tap B values decreases the quality of the

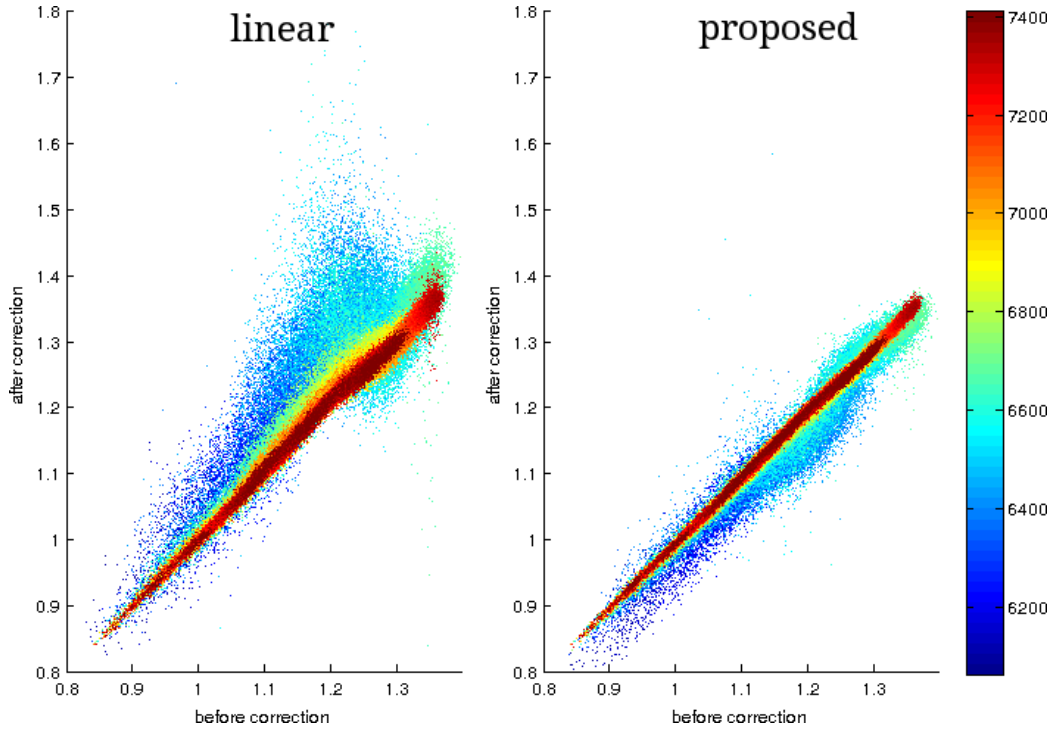


Figure 4.3.: This scatter plot shows the estimated phases using all eight available raw frames (averaging approach) with tap B values without correction (x-axis) and with r_1 correction (y-axis). The color encodes pixel intensity. Only the 10% with lowest intensities have been shown, points for brighter pixels are lying almost on a straight line (identity). **left:** linear reconstruction r_1 **right:** proposed reconstruction r_c In the left plot, dark pixels cover a large value range after reconstruction causing bad phase estimation results in dark regions. The proposed method works more robust for low intensities.

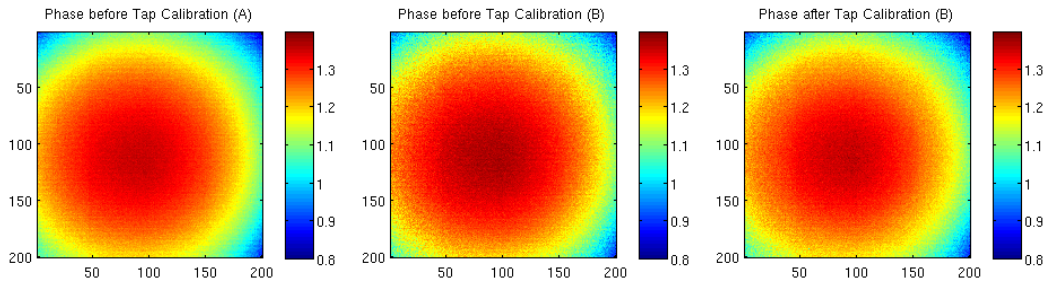


Figure 4.4.: Influence of the tap calibration on the estimated phase. Reference Tap A (**left**), Tap B before (**center**) and after application of the reconstruction r_c (**right**). Right picture looks almost like the left one (shape and noise pattern). Independent behavior of the two taps is lost after correction.

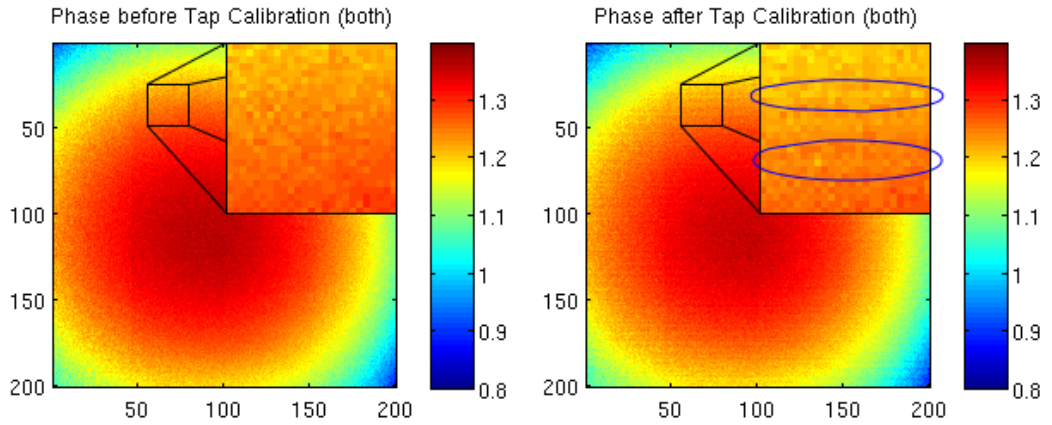


Figure 4.5.: Influence of the tap calibration on combined phase (averaging A and B). Shown using Tap A and B before (**left**) and after (**right**) application of r_c to B. Zoomed region is magnified by four. Right picture shows line artifacts (marked by blue ellipses).

results as visible in Figure 4.5. Averaging the raw values of the two taps reduces the effects of artifacts of the individual taps whereas the reconstruction of tap B introduces the artifacts of tap A also to the tap B values. So the reconstruction cancels the advantages of the averaging approach.

So tap calibration has advantages and disadvantages. Both taps of each pixel show different behavior which is helpful to average out pixel artifacts using all eight available uncalibrated raw data for phase estimation. So on static scenes, this averaging approach should be used without any reconstruction applied. At the other hand, decreasing the number of consecutive sub-frames to be captured to get all needed four phase shifts would decrease motion artifacts a lot. So assimilating the behavior of the taps is an important goal. An ideal (yet hypothetic) ToF device capable to record depth images without motion artifacts would use (at least) four taps per pixel and capture correlations with all four phase shifts simultaneously. But dividing the size of the surface on chip available to each tap even further, non-uniform behavior of these taps would even be stronger. Instead of picking one physical tap as a reference and assimilating the others one should use a ground truth photo response which would also tackle the problem of fixed-pattern-noise (DSNU/PRNU). But this ground truth is hard to estimate on the fly. Temperature and integration time dependency of the dark signal make this problem even harder and additionally impossible to calibrate this ground truth once and use it when record real sequences later. So the calibration approach presented here is the best one can do under the conditions of present ToF devices.

4.2.2. Image Homogenization

To be able to estimate the optical flow between the captured intensity images (sum of tap A and B of the individual sub-frames), one has to use tap-calibrated data (after application of r_c to the tap B records) to avoid effects of different behavior of the taps. Second, there is a strong fixed pattern noise in the captured pictures as well as inhomogeneous lighting. Most optical flow methods do not work well under such conditions. Lindner and Kolb [LK09] propose a homogenization approach which is an *inter-pixel* calibration to avoid these issues. Experiments showed that using the proposed form f

$$f(R_i) = a\sqrt{R_i + b} + cR_i + d \qquad g(R_i) = cR_i + d \tag{4.6}$$

performs equally well as a simple linear formulation g (setting a, b to zero). R_i are the captured raw frames. The latter has the additional advantage that it does not change the value of the calculated phase since the parameters cancel out in the phase calculation formula and thus may be considered as being orthogonal to the *inter-tap* calibration presented before.

4.3. Motion Compensation Methods

This section contains a detailed description of the considered motion compensation methods. Starting with Schmidt's *framerate increase* able to reduce the artifacts by a fixed factor, this leads to the detect-and-repair methods and the optical-flow based methods working on raw intensity images.

4.3.1. Framerate Increase

As presented in Subsection 2.2.1, there are several options to choose the data used for depth estimation.

Whereas usually (i.e. querying the depth from the manufacturer's software development kit (SDK), possibly computed on device depending on the used camera model), all available data is used for depth estimation by selecting the averaging approach. To reduce random noise and the influence of the individual taps, this is the best method when recording static scenes.

Under the conditions of a non-static scene, e.g. a scene with moving objects and/or a scene recorded with a moving camera, Schmidt proposes a method he called *framerate increase* [Sch11, section 5.3.3.2] which aims to keep the number of used exposures as small as possible, i.e. in the case of the present two-tap sensor two instead of four. This is done by selecting the subsets S1 or S2 and hence discarding the first or last two exposures. At the other hand, this allows to compute two independent depth maps from ToF data of a single frame, so this may be considered as an effective doubling of the number of depth maps available per time interval which leads to the name framerate *increase*. But note that depending on how the ToF device is configured, the time delay between the exposures within a single frame

may be different from what could be expected from the rate of multiple frames and so this method's name may be misleading. In case of the used *CamCube* device in *triggered* mode (adjustable frame rate), the exposures of a single frame are recorded as fast as possible (burst), then the device waits for the next trigger event. In a so-called *freerun* mode (high frame rate dictated by the device, depends on integration time), the delay between exposures stays constant even across frames. These details are analyzed in Section 4.7.

4.3.2. Detect-and-Repair Methods by Schmidt

Additionally to the *framerate increase* method reducing but not eliminating the motion artifacts, Schmidt proposes two variants of a method capable to detect such artifacts under certain circumstances and to repair them by using data recorded earlier if needed [Sch11, section 6.2]

First, during the *detection* step, differences of the recorded raw data over time are determined (may be considered as a temporal derivative). The *standard* method works on an inter-frame basis, i.e. comparing the values of all eight raw frames with corresponding ones from the preceding frame and hence may be used with all variants of ToF depth computation presented in Subsection 2.2.1. The burst internal detection (BID) variant works within a single frame and compares the raw values from the second subset (S2) with the corresponding ones from the preceding subset (S1). Thus, this variant restricts the depth computation to the S2 variant, using only the (possibly corrected) values from the second subset.

A raw value change above a certain (fixed) threshold is considered as an *event* which may cause motion artifacts. Motion artifacts arise if the event happens between the exposures needed for depth computation, i.e. in the open interval (t_0, t_3) for the standard and (t_2, t_3) for the BID variant (cf. Table 2.1 for definition of the t_i).

In case such a condition is detected, a second step is neccessary to repair this artifact affected data. All raw data of the particular channel recorded after that event is discarded and replaced by old data from the preceding time period, i.e. using data from the preceding frame (standard) or the preceding subset (BID).

The method is capable to handle at most one single event within two consecutive time periods, i.e. at most one event per two frames (standard variant) or at most one event per frame (BID variant). If multiple events fall into the mentioned time windows, detection still works, but correction will fail because the data used to repair the discarded values is wrong as well.

This method has been implemented as a CPU-based C++ application first constructing a discontinuity map (thresholding the raw data differences with a configurable fixed threshold) followed by a correction step if necessary as described above. Since computations are identical for all pixels, a GPU parallelization should be feasible but the implementation used here is already quite fast (and possibly I/O bound).

4.3.3. Optical Flow Based Motion Compensation

In contrast to the methods of Schmidt which strictly work on a per-pixel basis with a discontinuity threshold, Lindner and Kolb [LK09] try to revert the motion causing the artifacts. Here, the detection step consists of application of a 2D optical flow algorithm to the preprocessed intensity images (sum of tap data collected during one exposure). The needed image homogenization is described in Subsection 4.2.2. Pairs of the first intensity image and all three consecutive ones are used as input for the flow algorithm.

As repair step, all raw frames despite the ones from the first exposure are warped with the computed flow field effectively reverting the detected motion process. The warped raw frames are then used as input for the usual depth computation variants (cf. Subsection 2.2.1). Note that this does not only mix raw values from different taps of a single pixel but also from different pixels which may require additional calibration steps.

4.3.4. Combinations

Additionally to the provided implementations and comparison of the methods presented so far, it is possible to further enhance the results by combining the ideas of individual methods to create new variants. Especially the *framerate increase* approach may be used to enhance computational and algorithmic performance of the flow based motion compensation method by Lindner and Kolb. The former alone does not eliminate artifacts but reduce them by a factor of three. Since only data from two exposures is used for depth computation (subset S2), this is only one image pair which needs optical flow estimation (instead of three as proposed by Lindner and Kolb). As optical flow estimation works better when displacements are small, it is an additional benefit that this combination avoids flow estimation on raw image pairs spanning more than two consecutive exposures.

The *detect-and-repair methods* by Schmidt and the flow based method are incompatible with each other since one has to decide if the detected artifact affected values should be replaced by values from the past or from the neighborhood. But at least, the detection step of Schmidt's methods may be used to check the quality of the results from the flow based method. Since this pixel-based detection is quite fast, in a post-processing step this could be used to mark regions where discontinuities are a hint to failed correction based on warping with the flow results.

4.4. Qualitative Evaluation

All methods have been evaluated on multiple real and synthetic test sequences. As a real-life example, a short *office sequence* of 256 frames has been recorded heavily moving and rotating the camera to obtain strong motion artifacts. The sequence contains diffuse as well as transparent and specular reflecting objects which are all

within the non-ambiguity range. The camera position itself was almost kept constant so motion is only in the lateral and not in the depth direction.

To avoid influence of the sensor characteristics such as *dark signal non-uniformity* (DSNU) and *photo response non-uniformity* (PRNU) a synthetic *moving plane sequence* consisting of two frames with four sub-frames each has been created. A plane centered in the camera's field of view was moved from a distance of 1.6 m in frame 1 to 4 m in frame 2. It moves with a speed of 5 cm per sub-frame in the first and 10 cm per sub-frame in the second frame. The dark background has a distance of 4.2 m. The used point light source is located exactly at the origin at the camera position. Hence, complementary to the rotating scene, this sequence is ideal to analyze the influence of motion along depth direction. The setup generated with the *Blender* software is visualized in Figure 4.6. To generate the ToF raw frame output the simulator by Meister et al. [MNK13] has been used.

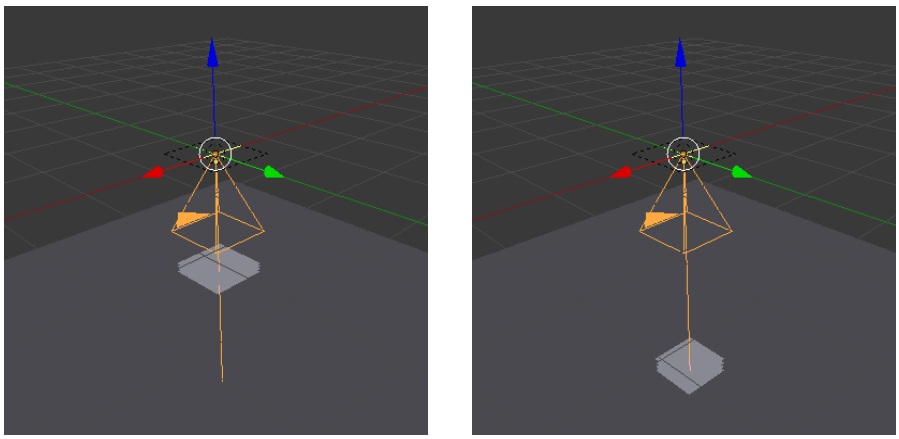


Figure 4.6.: Setup of the rendered synthetic *moving plane sequence*. Sub-frames are printed simultaneously. **left:** frame1 **right:** frame2

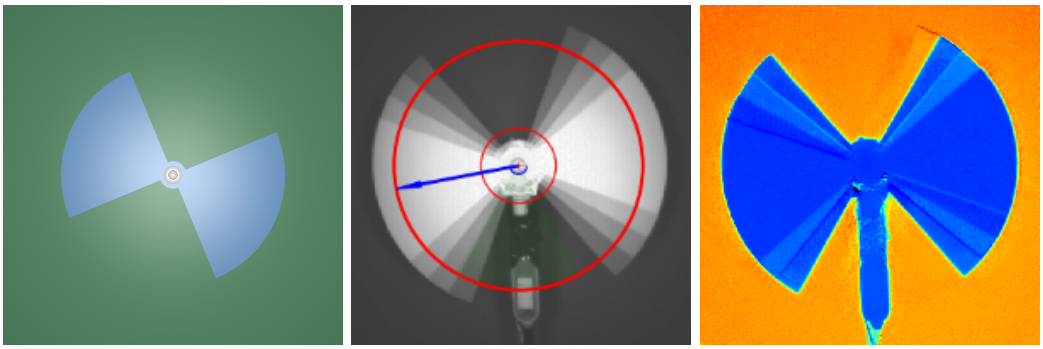


Figure 4.7.: Rotor test target: **left:** sketch of the target shape, **center:** CamCube3 intensity image with overlay of radii R_1, R_2 and estimated position φ, **right:** corresponding depth image with motion artifacts

Additionally, for quantitative evaluation, the rotating target proposed in the work of Schmidt [Sch11] shown in Figure 4.7 has been used. This sequence has been recorded as real ToF sequence as well as simulated with the ToF simulator. Since optical flow estimation on a non-textured surface only yields interpolated results (due to regularization), the target has been recorded plain white as well as with a more or less randomly textured pattern (structure with low and high frequencies).

4.4.1. Framerate Increase

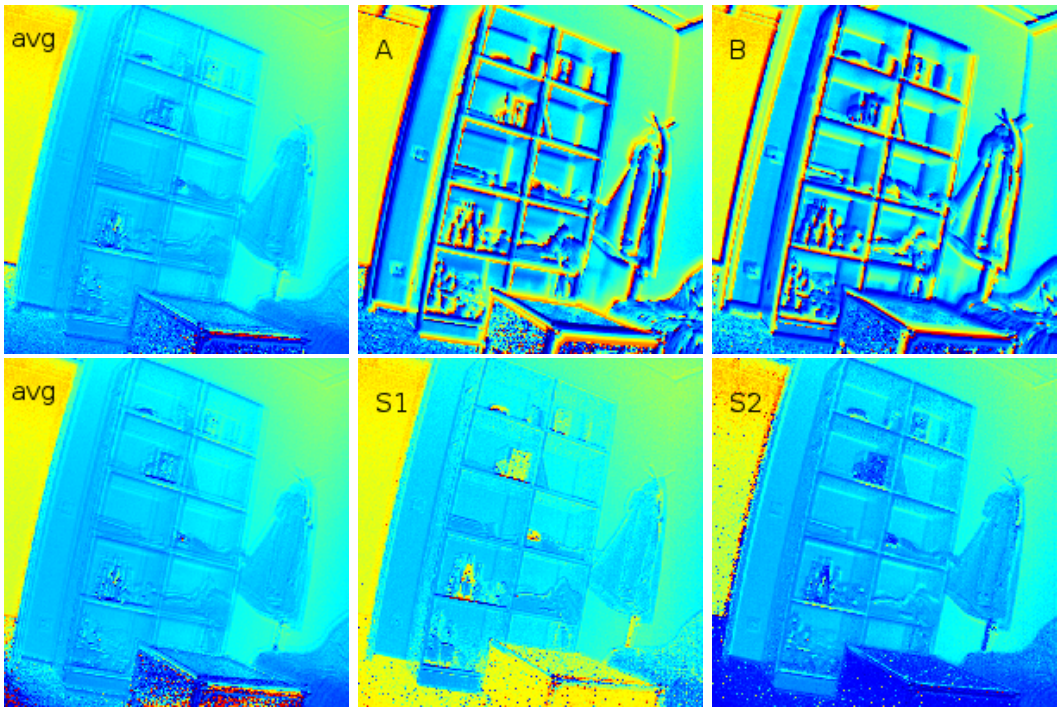


Figure 4.8.: Phase estimation of a dynamic scene using different subsets of the captured raw data. **Top row:** All data as given by the device, without tap calibration. Depth of all eight raw images using averaging approach (**tl**), all four tap A frames (**tc**), all four tap B frames (**tr**). **Bottom row:** All data with application of r_c to the tap B records. Averaged depth of all eight raw frames (**bl**), first subset of A and B raw frames (**bc**), second subset of A and B frames (**br**). S2 results (bottom right) look best.

Using the tap calibration presented before, it is possible to compute the phase of the returning light with less than all four acquired exposures. As visible in Figure 4.8, the estimated phase results differ significantly depending on which data have been used to compute it. Using uncalibrated data, there are three options that are shown in the top row. All four exposures taken consecutively are needed. Only taking one tap for phase estimation shows strong artifacts (top center and right image), averaging them decreases the effects but fine structures have still up to four halo-like artifacts. The top left and bottom left image both use all eight raw frames but

at the bottom row, all tap B data are corrected using r_c from the tap calibration. Using corrected tap B data and averaging it with tap A decreases the quality benefit provided by the averaging process. So the bottom left image looks slightly worse than the top left one. But using calibrated Tap B data, it is possible to compute the phase using the raw images $A_0, A_{90}, B_{180}, B_{270}$ (subset S1) which are taken during the first two acquisitions. Also a second subset consisting of $B_0, B_{90}, A_{180}, A_{270}$ (subset S2) which are captured during the last two exposures is available. Phase computed using these subsets are shown in the last two bottom pictures. Since there is only one delay between the exposures (instead of three delays using the standard approaches), this reduces the motion artifacts by a factor of three. Results clearly outperform the phase estimation methods using all four sub-frames, only a doubling effect at edges and fine structures remains.

4.4.2. Detect and Repair Methods by Schmidt

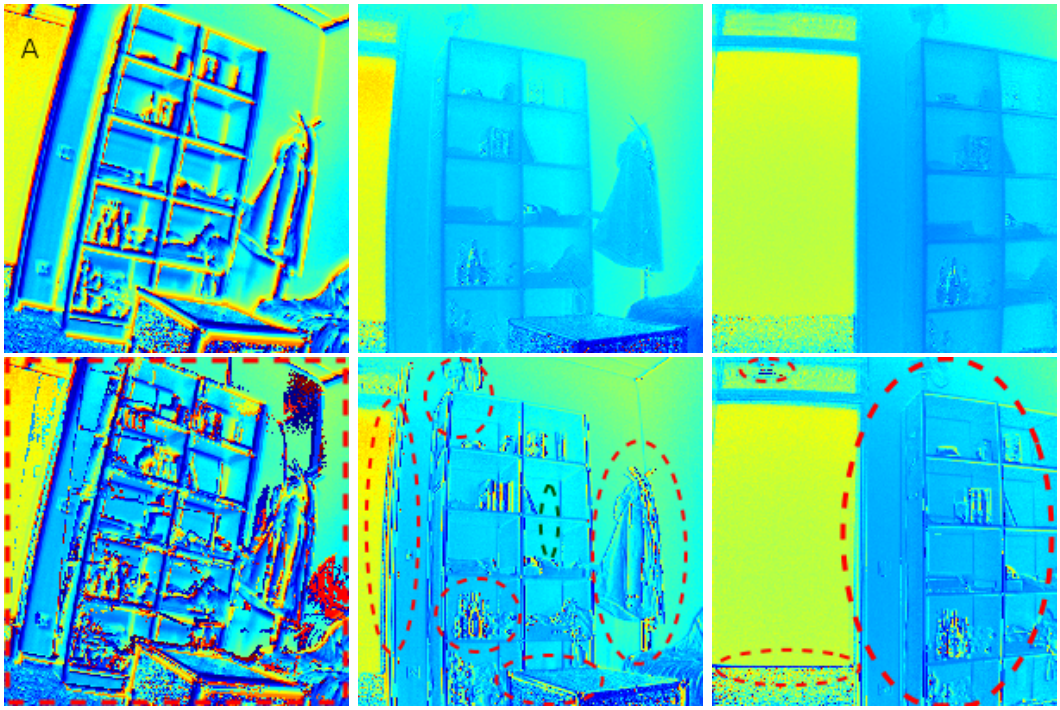


Figure 4.9.: Motion compensation method by Schmidt (Standard approach). All pictures show computed phases using the four tap A raw frames. Value range is the full non-ambiguity range R_N(4.1). **Top row:** before correction. **Bottom row:** after correction with marks of good (green) and bad (red) results. The columns show three different frames from the captured dynamic sequence with large motion speeds (**left**), first moved frame (**center**) and only small displacements (**right**). In all cases this method fails and causes strong artifacts. It is not suitable for moving cameras.

Figure 4.9 shows the results of the standard motion compensation approach by

Schmidt [Sch11]. The top row (uncorrected) corresponds to the top center picture of Figure 4.8 as the input to the standard method are the raw frames from all four exposures. For simplicity, only the tap A phase images have been printed. Applied to the dynamic office scene, the results look really poor, the method fails completely on this sequence. The algorithm assumes that at most one discontinuity (called *event* in [Sch11]) within two consecutive frames. This assumption is heavily violated in this real-world sequence. It may be fulfilled using a mounted camera in front of a almost static scene with only a few objects moving in lateral direction e.g. object inspection at a conveyor belt.

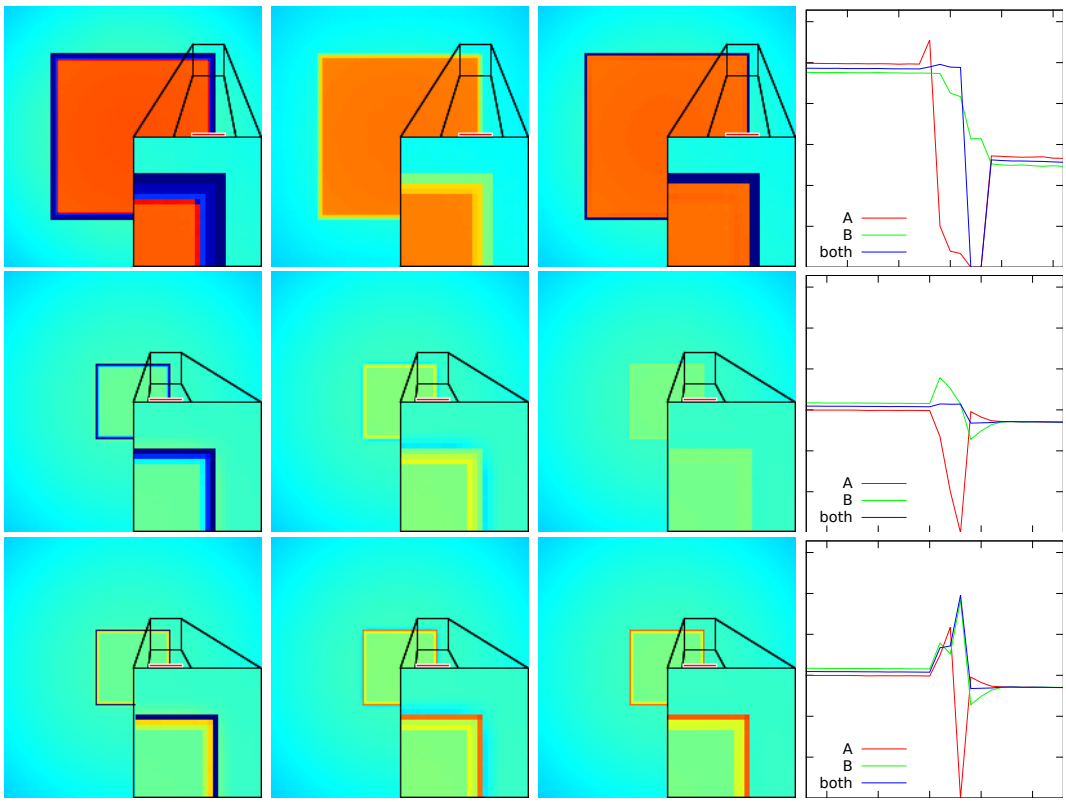


Figure 4.10.: Motion compensation method by Schmidt (Standard approach) on a synthetic sequence. Pictures show the estimated phase. Value range is the full non-ambiguity range R_N (4.1) (color-map and plots on the right). The top right edge was enlarged by four. **row1:** frame 1 (no correction possible) **row2:** frame 2 (uncorrected) **row3:** frame 2 (standard correction) **col1:** only using tap A **col2:** only using tap B **col3:** averaging tap A and B **col4:** graph of row 99 values of the preceding pictures (see red mark). Correction does not work well since replacing values of frame2 by some from frame1 causes new artifacts.

To test the performance in cases where the assumption of at most one event per two frames holds, the *moving plane test sequence* has been designed. In first experiments, Schmidt's standard approach did not perform any corrections at all. The used point

light source causes the plane moved far away from the camera to appear darker than closer to the light source. So the whole plane was detected to be discontinuous and so be kept uncorrected. To avoid influence of brightness change between the frames, one should use a light source without intensity decrease depending on the object distance (sun light source). The used simulator only supports point light sources with a light decrease proportional to $\frac{1}{r^2}$ (radial distance r). The object brightness had to be fixed by multiplying all intensities of the moving plane in the second frame by the average of z_2^2/z_1^2 (with the z-distance of the plane from the camera, in the considered case this fraction yields a factor of 4.908). After this preprocessing step, detection works as expected, only the object borders are marked as discontinuities. Results are shown in Figure 4.10. The first two rows show the phases before correction, last row the corrected frame 2. The first frame cannot be fixed since there is no preceding frame to check for discontinuities. Correction takes place at pixels where at least the first sub-frame (A_0 and B_{180}) is still continuous but values one of the subsequent phase shifts shows change between the two frames. So regions where the object moved away in all four channels are not touched and correctly estimated as background. Corrections are only necessary Discontinuous regions are fixed by replacing their raw values by old ones of the first frame (for the discontinuous raw channels only, the first raw channel is never changed). So after correction, the phase is estimated using mixed values from both frames. Since the plane moved a lot in depth direction between the frames, arbitrary depth values may occur. This explains why the phase results in the third row look worse than the uncorrected values. This shows that Schmidt's standard method is also not suitable for objects moving in depth direction.

Burst Internal Detection (BID) The main problem using Schmidt's standard approach is that there are large displacements between two frames of the dynamic sequence violating the assumption of a most one discontinuity within two frames. The improved variant called *burst internal detection* (BID) does not use two frames but two subsets of one frame to detect and fix the discontinuities. This allows one event per frame without violating the assumptions. It also decreases the considered time window since the delay between all consecutive exposures to capture the sub-frames is much shorter than the delay between frames (in this case, all four sub-frames are captured in about $\frac{1}{4}$ of the time between two frames).

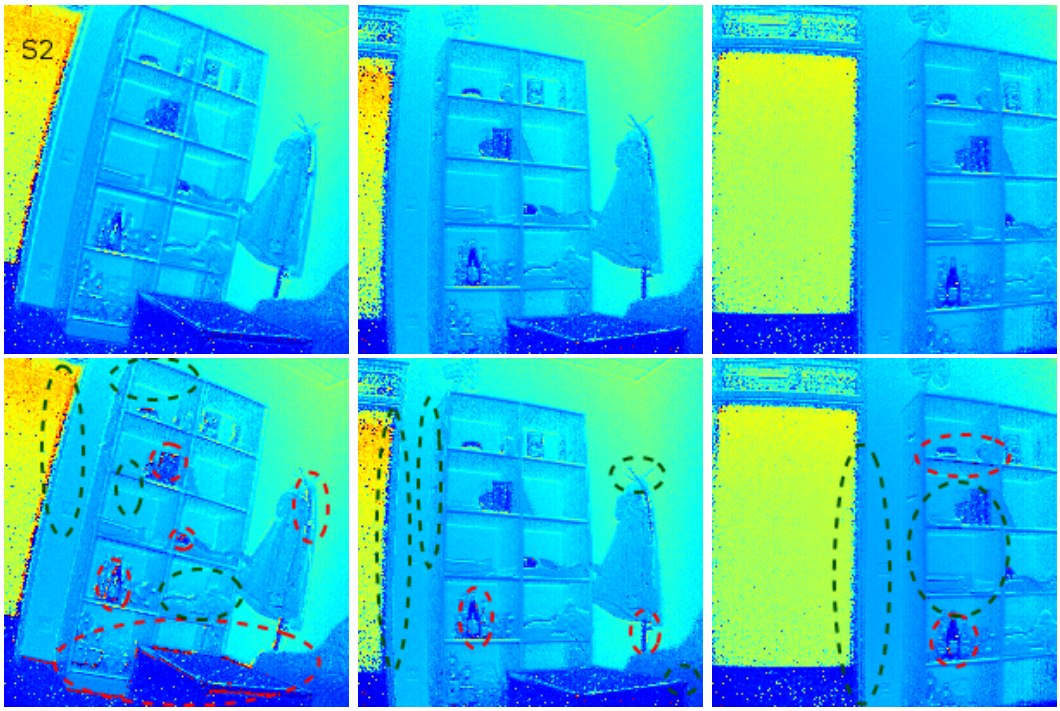


Figure 4.11.: Motion compensation method by Schmidt (BID approach). All pictures show phases computed using second subset. Value range is the full non-ambiguity range R_N. **row1:** before correction. **row2:** with BID correction and marks of good/bad (green/red) results. The **columns** show the same selected frames as in Figure 4.9. In the left pictures, the horizontal boards are sharper and with less halo-like artifacts. Same applies to the vertical boards in the center pictures. Very few correction in the right ones. In all cases new artifacts appear, at least at the bottles. But at least it works much better than standard approach (Figure 4.9).

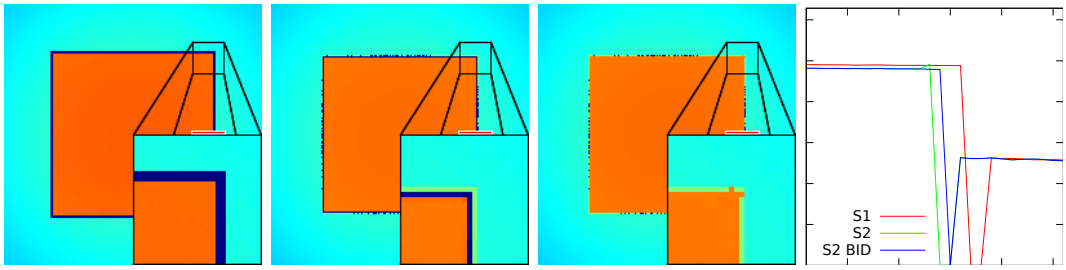


Figure 4.12.: Motion compensation method by Schmidt (BID approach) on a synthetic sequence. Pictures show the estimated phase. Value range is the full non-ambiguity range R_N (4.1) (color-map and plots on the right). The top right edge was enlarged by four. **left:** first subset (no correction possible) **second:** second subset (uncorrected) **third:** second subset (BID corrected) **right:** graph of row 99 values of the preceding pictures (see red mark). Correction worked well, only boarder of one pixel shows flying-pixel artifacts (blue/green) that are also present on static scenes.

Applied to the dynamic *office sequence*, the BID method gives the results shown in Figure 4.11. The top row (uncorrected) corresponds to the bottom right picture of Figure 4.8 as the input to the BID algorithm are the two subsets of one single frame of the sequence and only the second subset is corrected (by values from the first one). This is clearly an improvement compared to the standard method. There are still regions with artifacts, especially on translucent or reflecting objects but the doubling artifact occurring at depth edges in the uncorrected images is removed in most cases.

Applied to the *moving plane* test sequence, Schmidt's BID method gives very accurate results. Brightness changes between the two considered subsets, caused by the point light source are small enough and did not need for any correction like above. Only a border of one pixel with wrong depth values remains which can be considered as a flying pixel artifact which is also present in ToF records of static scenes at depth edges. So the BID approach solves the motion-artifact problem at this simple synthetic test sequence.

4.4.3. Optical Flow to Warp the Sub-frames

Since the presented system is intended to work in real-time, only reference implementations of optical flow methods working on a graphics processing unit (GPU) have been investigated. Recently, many state-of-the-art methods have been implemented using CUDA and OpenCL in the OpenCV [Bra+00] library. Especially the following algorithms are available: The algorithm by Brox et al. [Bro+04], the duality-based total variation (TV-L^1) method by Zach et al. [ZPB07], the method by Farneback [Far00], a pyramid variant of the method by Lucas and Kanade [LK81] (PyrLK) [LK81], as well as block matching (BM) algorithms. To simplify testing the different algorithms, the OpenCV methods have been wrapped into charon-suite modules [GK12; GMK]. This way, the common pre- and postprocessing parts could be performed using existing code.

As presented in the Appendix A, optical flow result vary heavily depending on the used method. All local methods give unusable flow fields since they do not produce dense results (i.e. zero flow e.g. where structure was too low). TV-L^1 and Brox Algorithm perform nearly equally well but regarding the residual image, TV-L^1 results are slightly better (even if there are two artifacts at the top of the image).

Figure 4.13 shows the computed phase images using the raw frames warped with the TV-L^1 flow results. As an interesting observation one should note that the quality strongly depends on the choice of phase computation method. Selecting all data from one single tap gives bad results (top row). Especially the object edges show misalignment effects that look like aliasing. Combining images from both taps removes this kind of artifacts.

Averaging all tap A and B exposures gives sharp edges and low noise in homogeneous regions. Only parts where the intensity was low (floor and box in the front) show high noise as well as the reflecting/translucent objects in the shelf. For some

reason the noise in the low-intensity parts seems to be much higher in the first two exposures causing the depth maps computed using the first subset (bottom center) to be nearly useless there. Using the second subset only, results look drastically better. Edges are much sharper, even the bottles and dark objects in the shelf have been estimated correctly. Only at the horizontal wooden boards there are still some small halo-like artifacts that are not visible in the averaged image.

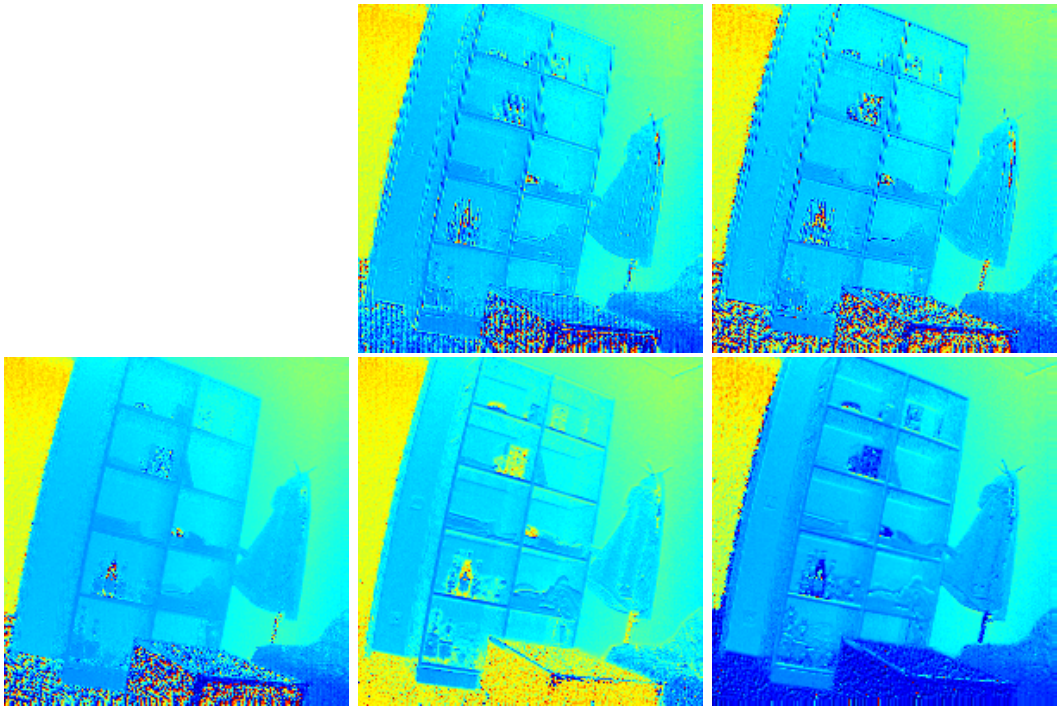


Figure 4.13.: Same Phase Visualizations as in Figure 4.8 but after warping the raw frames (r_c corrected) with the calculated optical flow computed using the TV-L^1 method. **Top**: phase using tap A (**center**) and B (**right**) records. **Bottom:** phase using all raw frames (averaging, **left**), first subset (**center**) and second subset (**right**). A combination of warping with computed flow and using framerate increase (S2, bottom right) gives the best results.

4.5. PMD Depth estimation

Working with the raw data provided by the PMD *CamCube* devices, one should note that the depth values computed using the usual formulas found in literature and the depth values computed by the SDK provided by the manufacturer differ heavily.

4.5.1. PMD SDK

During experiments with the *CamCube3* device SDK provided by the manufacturer (`PMDSDK 2.2.1`) has been used to connect to the device and as reference for raw data processing. This software library provides options to configure the device (e.g. set/get integration time, modulation frequency) and the capability of raw data capturing. Additionally, it is possible to compute

- aplitudes (cf. (2.2))
- intensities (cf. (2.3))
- radial distances (cf. (2.4) and (2.5))
- flags (e.g. saturated pixels)
- 3D coordinates

This post processing works on-line (while capturing data with the device) and off-line (given the captured raw data stored on disk) but requires data be exactly in the format as captured from the device. This means that it is not easily possible to apply this post processing pipeline to processed raw data, i.e. after application of the algorithms described in this work.

4.5.2. Comparison of the Computed Values

As a first step, a comparison the computed phase shift of the returned light by using (2.4) and the distance values computed by the SDK converted to a phase shift via (2.5) is given. To sample the whole range of phases ($[0, 2\pi)$), a out of the window scene where an inclined wall covers most of the image has been recorded. It also includes regions with high amplitude of the returned light as well as low amplitudes in regions far away. The computed values and their difference are shown in Figure 4.14.

The best match of the computed values was computed using the average approach of phase reconstruction. This is reasonable because it uses all available captured data and reduces tap artifacts. But nevertheless there is a systematic offset, some sinusoidal structure and the values do not lie on a straight line but cover a range of about 0.1 rad (Figure 4.14, right). Zooming in to one of the extrema of the sine structure reveals that the values are not scattered randomly but show equidistant sine waves. By only selecting the values of one of those parallel structures, it is

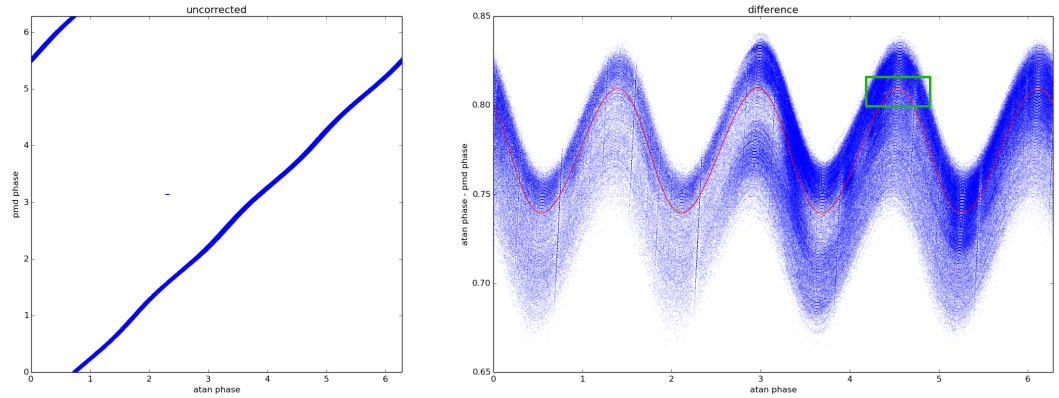


Figure 4.14.: Comparison of Phase values computed on raw data with usual formula and provided by the PMD SDK. Each pixel data is represented by a single small point. The line width in the left picture is caused by the variations better visible in the difference view on the right. Fitted correction function shown in red.

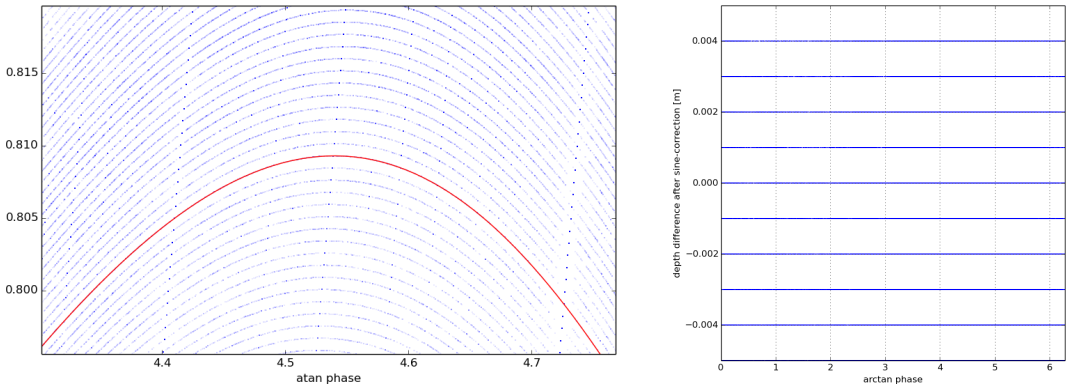


Figure 4.15.: **left:** zoom to an extremum of the sine shape of phase difference (marked rectangle in Figure 4.14), values in [rad]. Best fit of a single stripe plotted in red. **right:** remaining radial depth difference (in [m]) after correction of shift and sine wave structure (by applying the correction function plotted in red in left plot).

ν_M	φ_{off}	a	φ_{shift}
18 MHz	1.220 706 482 046 9 rad	0.034 906 6 rad	$-\frac{\pi}{4}$
19 MHz	0.978 259 187 887 76 rad	0.034 906 6 rad	$-\frac{\pi}{4}$
20 MHz	0.774 375 510 153 79 rad	0.034 906 6 rad	$-\frac{\pi}{4}$
21 MHz	0.511 849 839 279 5 rad	0.034 906 6 rad	$-\frac{\pi}{4}$

Table 4.1.: Fitted parameters of the correction function in (4.7).

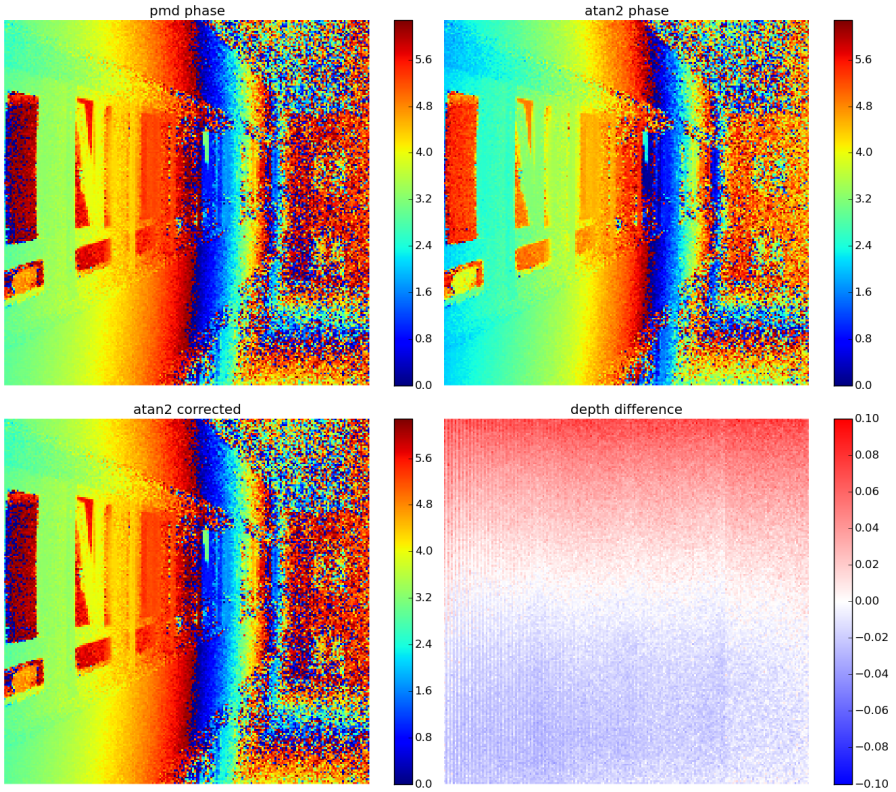


Figure 4.16.: Phase images computed with PMD SDK (**top left**), using *atan2*-formula (**top right**), after correction using (4.7) (**bottom left**) and the remaining difference pattern in units of meter after correction (**bottom right**)

possible to fit a sine wave to these filtered data perfectly. After application of the correction function

$$\varphi_{corr} = \varphi_{atan2} + \varphi_{off} + a \cdot \sin\left(4 \cdot \varphi_{atan2} + \varphi_{shift}\right) \tag{4.7}$$

the remaining distances after correction are in range of float/double precision.

There are exactly four periods of the sine wave within the value range, which is the most dominant mode of the *wiggling artifact* occurring on CWIM ToF devices with non-perfect sinusoidal modulation of the emitted light [Sch11]. The fitted parameter φ_{off} depends on the used modulation frequency ν_M, amplitude a and shift φ_{shift} are constant (cf. Table 4.1).

After correction of offset and sine wave, the remaining difference are horizontal parallel lines. After computing the distance (in [m] using (2.5)), these lines have distances of exactly 1 mm. This fact occurs independent of the captured scene, the used integration time and modulation frequency (regarding (2.5), the phase distance (in [rad]) changes with modulation frequency, but the radial distance (in [m]) keeps constant).

Additionally, this remaining correction pattern has not only mm values but also really stays the same and so must be hard-coded into the processing pipeline. Also note, that this could not be the result of a camera calibration process since this processing pipeline was applied offline to recorded raw data without communication to the device. The only possible calibration would be a live-calibration using the recorded data which would yield different correction patterns when applied to different recorded files, which is not the case. Regarding the data format of the stored raw data, it does only contain the recorded raw data, some meta data like used modulation frequency, integration time, device type but there is not enough space to pass a whole calibration image. So this hard-coded pattern may be specific to the used device type but not to the actually used camera. The structure of this pattern is shown in Figure 4.16. Also note that this correction is really relevant since the depth difference is up to 10 cm.

Using this described processing pipeline, it is possible to compute exactly the same depth values from the raw data as given by the black-box function provided by the PMD SDK up to floating point number precision.

4.5.3. Provided 3D Point Coordinates

With computed values of the radial distance, it is possible to compute the 3D point coordinates (x, y, z) for each pixel using the intrinsic camera parameters:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \frac{r}{\sqrt{u^2 + v^2 + f^2}} \cdot \begin{pmatrix} u \\ v \\ f \end{pmatrix} \qquad \text{with} \qquad \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u_p - u_c \\ v_p - v_c \end{pmatrix} \tag{4.8}$$

This formula may directly be derived from the sketch in Figure 2.1 regarding the used pinhole camera model. The radial distance r should be equal to the norm (lenght) of the vector $(x, y, z)^T$. So by rescaling the vector $(u, v, f)^T$ pointing in the same direction, the 3D coordinates may be retrieved. The coordinates in the camera system $(u, v)^T$ have to be computed by calculation of the difference of the pixel position $(u_p, v_p)^T$ (e.g. values in [0,...,199] for a 200 pixel sensor) and the sensor center $(u_c, v_c)^T$ on the optical axis. By shortening f, u and v may be replaced by ratios $u/f_x, v/f_y$ and thus f_x, f_y may be specified in multiples of the pixel lattice periods in x, y directions instead of real length values. For non-quadratic pixels, the values of f_x and f_y differ.

If there are pre-computed values of the 3D coordinates as well as radial distances, as given by the *black-box* PMD SDK, this may be inverted to determine the built-in camera parameters. This has been implemented in Listing A.3 and yields the following results when applied to some recorded *CamCube3* dataset:

```
$ getCamParams.py fast-1700.h5
fx = 278.876505 fy = 278.876396 cx = 101.500000 cy = 101.499999
```

This shows that the built-in parameters are set to the ideal values of a camera with a field of view (FOV) of 40° and a sensor size of 204 × 204 pixels, which are

the properties of the *CamCube2* device:

$$u_c = v_c = \frac{204-1}{2} = 101.5 \qquad f = \frac{101.5}{\tan(40°/2)} \approx 278.868\,958\,074 \tag{4.9}$$

Although the used *CamCube3* device has other sensor properties (200×200 pixels, but same FOV), the SDK applies these fixed values nevertheless (even despite the fact that the used camera model is stored in the meta-data recorded by the SDK and present during the 3D point coordinate computations by the same SDK).

4.6. Automated Rotor Position and Velocity Detection

For the further experiments and quantitative analysis, the orientation and speed of the used test target are important quantities. To detect them on large number of frames in various sequences and using different ToF devices, it is useful to be able to determine them in an automated way. Hence a processing pipeline for automatic rotor position detection had to be developed.

Figure 4.17 shows a visualization of the used approach. A virtual rotor mask may be generated for arbitrary angles $\varphi_r \in [0, \pi)$. This boolean mask is described in 2D polar coordinates (r, φ) with origin located at the rotor center (rotation axis of the fan) and constants R_1, R_2 describing the inner and outer radius of the rotor:

$$M_{\varphi_r} = \left\{ (r,\varphi) \middle| r \in [R_1, R_2],\, \varphi \in \left(\bigcup_{k=-1}^{2} \left[\varphi_r - \frac{\pi}{4} + k\pi,\ \varphi_r + \frac{\pi}{4} + k\pi \right] \right) \cap [0, 2\pi) \right\} \tag{4.10}$$

The value sum of the pixel-wise product of this mask with the intensity image is used as an objective function to be maximized with respect to the rotor angle (`argmax`). In presence of rotor-artifacts, there may be steps or plateaus in the objective function which yields to unstable optimization results. This may be avoided by pre-blurring the intensity image by convolution with a Gaussian filter.

By computing the differences between rotor positions of consecutive frames, it is possible to estimate the angular velocity of the rotor for each frame. This is shown in Figure 4.18. The plots show data points with the computed position differences between consecutive frames. Since all positions are within the interval $[0, \pi)$ (note the point symmetry of the rotor target), there may be negative differences when the rotor crosses the $\varphi = 0$ angle. This may be fixed by adding a fixed value of π to all negative difference values:

$$\Delta\varphi_n = \varphi_n - \varphi_{n-1} + \begin{cases} \pi & \text{if} \quad \varphi_n < \varphi_{n-1} \\ 0 & \text{else} \end{cases} \tag{4.11}$$

Also note that with a rotor speed of about 6 revolutions per second and a recording at 30 and 60 frames per second (and a four times larger rate of sub-frames), there are no ambiguities (like rotor turned by 180 degrees), even in the case of single frame

drops. The plots in Figure 4.18 show the data points $\Delta\varphi_n$, their arithmetic mean (red line in left plots), their standard deviation σ (green range in left plots) and the box-whisker-plots (more robust in presence of outliers) in the right column. These box plots show the median value (red line), the quartiles (blue box spans the 25 to 75 percentiles, almost invisibly small in the first two rows), the whiskers span the range of the 5 to 95 percentiles. Points outside of this range are considered as outliers and marked by dots.

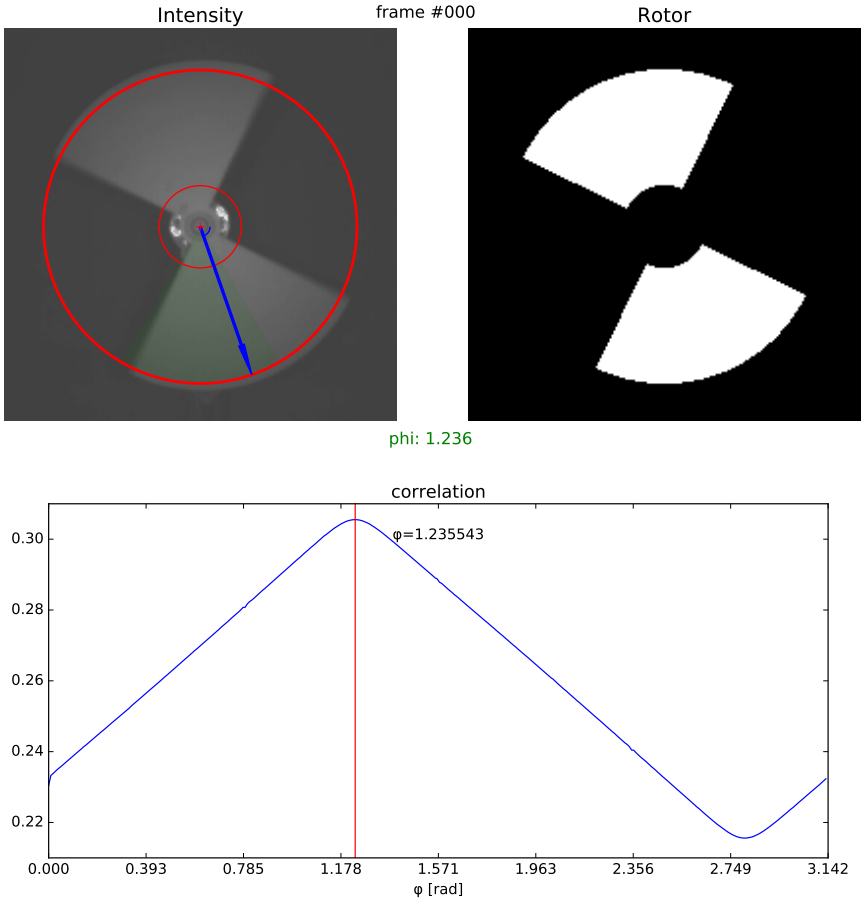


Figure 4.17.: Automated rotor position detection by correlating the intensity image (**left**) with a virtual rotor mask (**right**). Using a scalar optimizer (from Python SciPy package) the angle of maximal correlation (**bottom**) is determined.

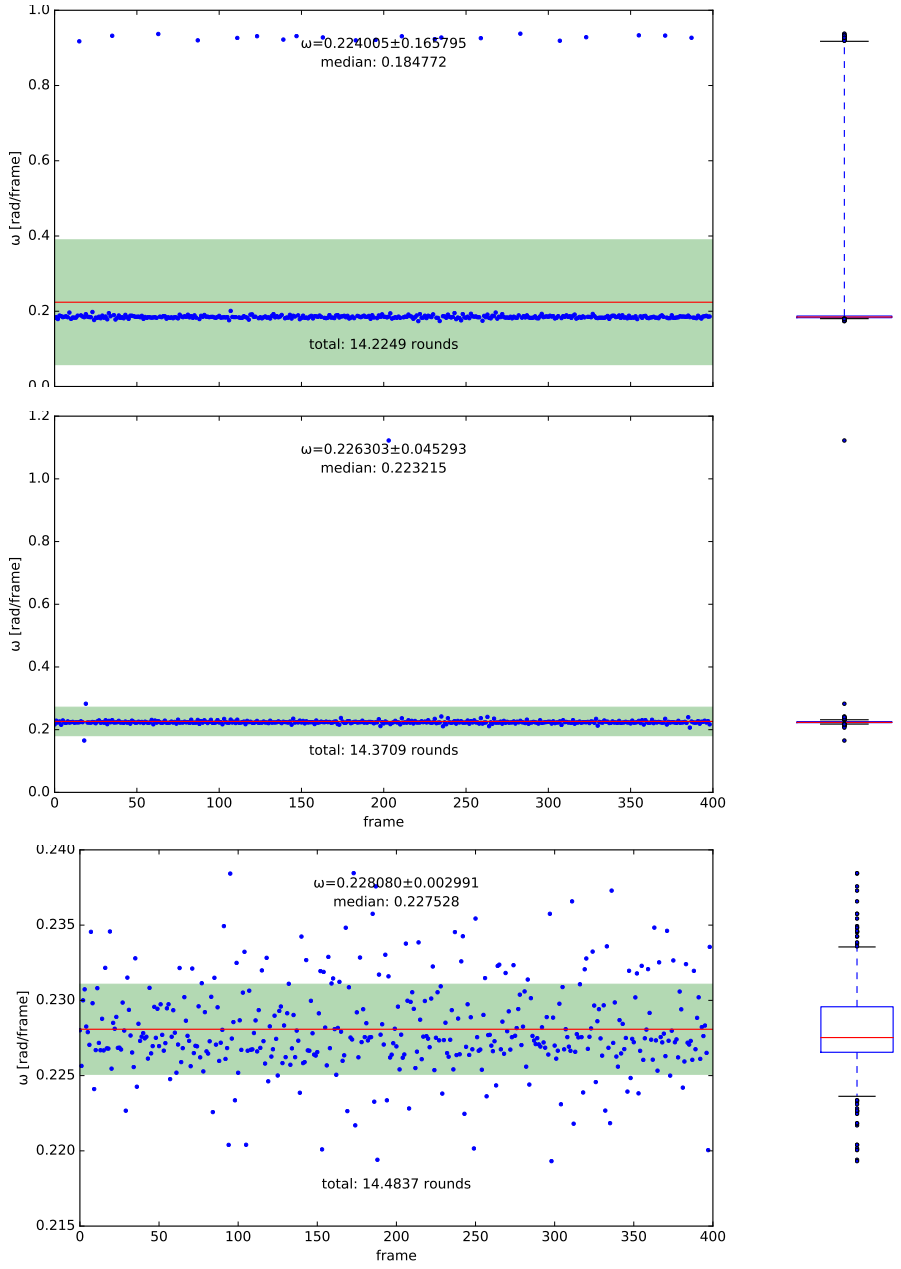


Figure 4.18.: Examples of estimated rotor velocity estimation on a fast rotor sequence with different integration times. **Rows:** $t_i = 500\,\mu s$ (**top**), $t_i = 1500\,\mu s$ (**middle**), $t_i = 1600\,\mu s$ (**bottom**). **Columns:** Individual angle difference between sub-frames (**left**), box-whisker-plots to summarize the distribution of the values by quantiles (**right**). When frame-drops occur, the estimated angle difference is five time as large as usual, because the dropped frame contains four sub-frames which are skipped in this case. At low integration times (top), there are many frame-drops, one single drop at the center plot and no drops at all on the last plot (note the different y value range!).

4.7. Technical Details on Recording Data with the *CamCube* Device

The PMD *CamCube* software interface provided by the manufacturer's SDK provides three different modes to control the capturing process of the device:

Freerun Mode The device runs its exposures as fast as possible without additional delays. The internal frame rate may not be adjusted. The internal frame rate only depends on the configurable integration time. The recording software is responsible to check for new available data and transfer it as fast as it is recorded, otherwise data of some frames is dropped. Especially for short integration times, data is generated too fast to be transferred using the USB connection between computer and device. This may be avoided by selecting a specific region of interest (ROI) (which will further shrink the already small image size of 200x200 pixels) and thus reducing the data to be transferred per frame.

Software Triggered Mode Each acquisition of a frame has to be started explicitly by a function call in the recording software. All exposures for that frame run then at the same internal sub-frame-rate as in freerun mode but after all exposures for that frame, the device waits until the next acquisition is requested by the software. Since the trigger command is also responsible to initiate the data transfer, there will be no frame drops using this mode.

Hardware Triggered Mode Each acquisition of a frame has to be started using an external hardware trigger signal. There will be frame drops if the data transfer is slower than generation on the device.

Experiments with the *CamCube3* device show that the delay between individual exposures only depends on the selected integration time. If using the freerun mode, this delay even stays constant across acquisition of multiple frames. Using the rotor target designed for quantitative evaluation, it is possible to retrieve some of the technical properties of the device. With a fast, fixed rotation speed, the target was recorded at different integration times t_i. To avoid pixel saturation at high integration times, a gray filter had to be placed in front of the camera objective. Then, the rotor orientation in the recorded intensity images was estimated automatically by maximizing the correlation of the image with a virtual rotor at arbitrary orientations. By computing the differences of the rotor angles of consecutive sub-frames (fixed when wrapping by 2π occurs), the rotor velocity may be estimated in units of [radian/subframe]. Each of the recorded sequences contains 100 frames and hence 400 sub-frames. Using the median value of all those differences, this velocity may be estimated accurately even in presence of frame-drops. The results of this experiment with different integration times are given in Figure 4.19. Each plotted data point represents such a median value. The individual computed angle differences of sample sequences are shown in Figure 4.18. Because of the high number of values used

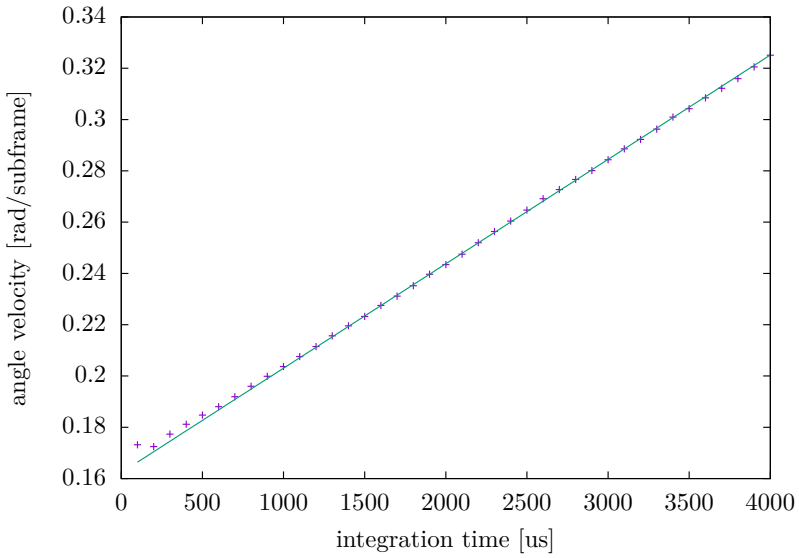


Figure 4.19.: Apparent angular speed of the rotor as visible in recorded sequences using different integration times. Each data point is the median value of the angle difference per sub-frame of sequences having 400 sub-frames each. Slope and offset of the linear fit may be used to estimate the real rotor speed and the delay between exposures.

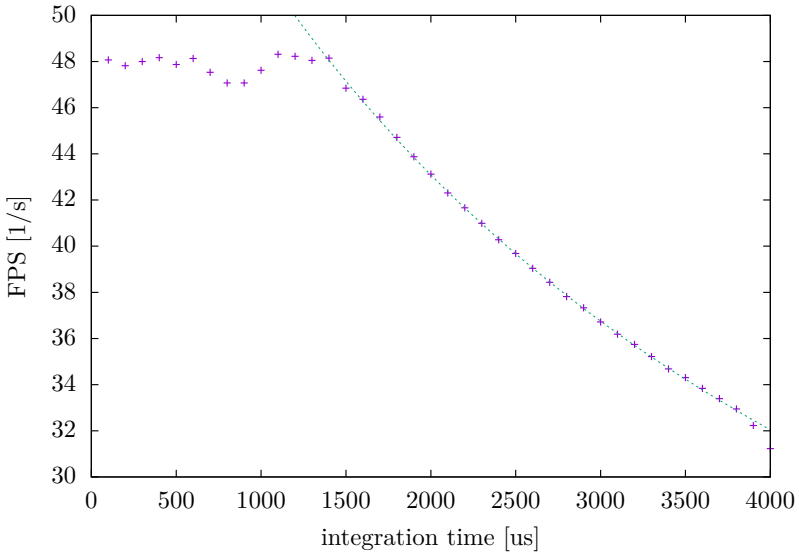


Figure 4.20.: Integration-time dependency of the record frame rate reached by the rawCaptureFreerun camera application. Max frame rate is about 48 FPS. Framedrops occur up to integration times of 1500 µs. The expected hyperbolic behavior is present for higher integration times.

for median computation and the small standard deviation of the individual samples (without the frame-drop outliers), the error values of the data points are negligible compared to the fitting error given later. E.g. the error of the value for $t_i = 1600\,\mu s$ is about

$$\text{average}(\Delta\varphi_r) = 0.228\,080\,\text{rad} \tag{4.12a}$$

$$\text{median}(\Delta\varphi_r) = 0.227\,528\,\text{rad} \tag{4.12b}$$

$$\sigma_m = \frac{\sigma}{\sqrt{N}} = \frac{0.002\,991\,\text{rad}}{\sqrt{400}} = 0.000\,149\,55\,\text{rad} \tag{4.12c}$$

which is about 0.066 % of the value.

Obviously, there is a linear dependency of the apparent rotor speed ω_{SF} per subframe and the integration time. This may be explained by an exposure process of two phases: integration time (with configured duration t_i) and a fixed time delay t_d between exposures, e.g. caused by the readout time of the chip and potential preparation/post processing (like adjusting the frequency shift of the correlation signal). Fitting a linear function

$$\omega_{SF} = \omega \cdot (t_i + t_d) \tag{4.13}$$

yields the values

$$\omega = (40.72 \pm 0.15)\,\frac{\text{rad}}{\text{s}} \qquad\qquad t_d = (3986.8 \pm 23.8)\,\mu s \tag{4.14}$$

As an independent measurement, the size of the delay t_d may also be estimated from the recorded FPS rate: The number of frames recorded per second is reciprocal to the duration to capture one frame. Since one frame consists of four exposures, this yields

$$\nu_{FPS} = \frac{1}{4 \cdot (t_i + t_d)} \tag{4.15}$$

This hyperbolic behavior is clearly visible in Figure 4.20. The shift t_d (in x-direction) of this hyperbola gives:

$$t_d = (3802.04 \pm 2.89)\,\mu s \tag{4.16}$$

The error values are the fitting errors and do not contain any systematic errors that may have occurred. Especially the FPS rate has been computed by dividing the total record time spent by the application by the number of recorded frames which may not be very accurate and influenced by other processes running on the computer. The accuracy of the first variant may be affected by the precision of the device clock used to set up the proper integration times. Regarding those unknown additional sources of errors, the different values computed for t_d should be considered as compatible.

4.8. Quantitative Evaluation

Additionally to the qualitative evaluation before, it is also important to compare the algorithms quantitatively.

4.8.1. Used Measure

For a quantitative evaluation of the algorithms, in literature there is only one proposed quantitative measure of artifact strength, the so called *relative distorted area* by Schmidt [Sch11]:

$$\rho = \frac{A_{art}}{A_{max}} \in [0, 1] \tag{4.17}$$

where A_{art} is the detected number of artifact pixels and A_{max} is the theoretical maximal possible number of artifact pixels for the considered scene. In principle A_{art} could be any kind of artifact pixels, not only motion artifacts, but to be able to quantify a maximum number of possible artifacts, the scene geometry and it's motion have to be known. To handle this, one needs a simple scene, e.g. with a foreground and background layer and simple known geometry and motion. Foreground and background are assumed to have a constant distance from the camera in z-direction. Pixels differing from these distances are considered as artifacts. Additionally, the brightness of the foreground is assumed to be light, background to be dark. Pixels with brightness values between foreground and background level are also considered as artifacts.

4.8.2. Test Scene

Figure 4.7 shows the used test setup. The circular shape with quarters of foreground/background and rotation with constant angular velocity yields (cf. [Sch11, (6.3)], here with fixed angular velocity and revolutions per frame):

$$A_{\max} = \pi(R_2^2 - R_1^2) \cdot 4n = (R_2^2 - R_1^2) \cdot 2\omega \tag{4.18}$$

where n is the number of revolutions per frame, and $\omega = 2\pi n$ is the angular velocity (in $[\frac{\text{rad}}{\text{frame}}]$). R_1, R_2 are the inner and outer rotor radius as in Section 4.6. The maximal reasonable speed of the test target is $n = \frac{1}{4}\frac{\text{round}}{\text{frame}}$ or $\omega = \frac{\pi}{2}\frac{\text{rad}}{\text{frame}}$. In this case, the maximal distorted area is the full rotor area $A_{max} = A_{rotor}$.

In principle, even higher rotor speeds could be considered, but $A_{\max}$ as worst-case scenario keeps constant in this case. The image regions which are hit by more than one foreground/background change event in this case count only once in $A_{\max}$ which may cause misleading results.

4.8.3. Discussion of the Measure

Regarding the design of the *relative distorted area*, there are some points to consider. First, all pixels, that are not in foreground/background plane are detected as

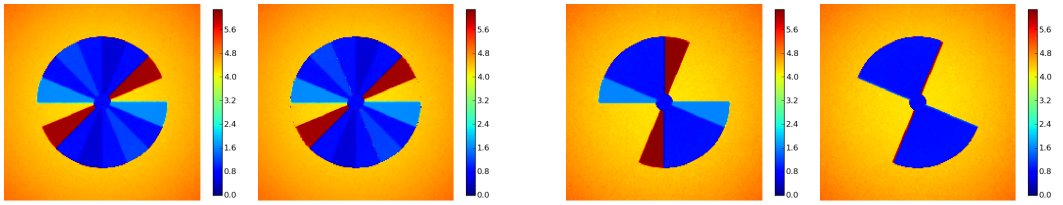
artifact pixels by thresholding as well the pixel intensity as well as the computed z-distance. Of course, this detects pixels influenced by motion artifacts but also e.g. flying pixels at object boundaries caused by mixing foreground/background signal at border pixels. Therefore it is useful to set the rotor speed of the test target to high values so that the area of motion artifacts becomes large compared to the border pixels.

A theoretical ToF device with sub-frames captured in equidistant time steps Δt and no but motion artifacts (i.e. without flying pixels) yields a *relative distorted area* $\varrho = \frac{3}{4}$ because time distance between frames is $4\Delta t$ and there are three time steps between the four used sub-frames for depth computing. Only using a subset of two sub-frames (*framerate increase*) will reduce this to a value $\varrho = \frac{1}{4}$ because within such a sub-frame, there is only one time step Δt.

The real ToF device (*CamCube3*) introduces some more parameters that influence the measured values of ϱ. They vary heavily depending on the frame rate when using the so called *soft trigger mode*, i.e. triggering the device using a software timer. In this setup, the time steps between sub-frames keeps constant and depends only on the selected exposure time. After acquiring the four sub-frames, the device waits for the next software trigger signal. So especially if using low frame rates (e.g. 10-30 FPS), this looks like sub-frames being acquired in *bursts* and yields low ϱ values because this waiting for next trigger artificially enlarges the rotor speed ω from frame to frame and so the A_{max} (cf. eqns. (4.18) and (4.17)). The higher the selected frame rate (up to 50 FPS), the closer the ϱ values approach to the theoretical value of $\frac{3}{4}$. The assumption of sub-frames captured equidistant in time does never hold exactly in soft trigger mode because there is always at least a short delay waiting for the trigger signal. If used in so called *freerun mode*, the device avoids this waiting step and so really reaches equidistant time steps between sub-frames. But at least for low exposure times, this results in more captured data than can be transferred via the USB2 connection and so causes frame drops. Selecting an exposure time of 1500 µs or higher, the internal frame rate of the device gets below 50 FPS and frame drops almost disappear. In this setup, using the maximal exposure time of 4000 µs (and a gray filter to avoid too bright images), an average value of $\varrho = 0.785 \pm 0.011$ has been determined which is close to the theoretical value of 0.75 for equidistant subframes. The value higher than $\frac{3}{4}$ is caused by border pixels and motion-blur during exposure.

4.8.4. Algorithm Comparison

Using the rotor test target presented above, it is possible to reveal a large difference between the two methods by Schmidt [Sch11] which demonstrate the importance of the prerequisite of at most one *events* per two considered time periods which makes a distinction between his standard and BID variant. Regarding the optical flow based methods, it is interesting how the estimation is able to cope with non-textured and textured targets in presence of large displacements and rotation, both making this a tough task.



$\varrho = 77.80\,\%$ $\varrho = 77.80\,\%$ $\varrho = 28.12\,\%$ $\varrho = 3.08\,\%$

Figure 4.21.: Fast Synthetic Rotor Sequence with Schmidt-Correction. $\omega = \frac{90°}{\text{frame}}$. Left pair: Phase computed using all eight raw images (average) before and after Schmidt's standard method. Correction does not improve the computed depth map. Right pair: Phase computed using second subset (S2) before and after Schmidt's BID method. Regions affected by artifacts are fixed, only some artifacts at the borders remain.

Synthetic Sequences

Using an artificial rotor scene, rendered using the ToF simulator by Meister et al. [MNK13], the influences of non-linear response curves and different pixel behavior are not considered to focus on the scene prerequisites alone and neglect the non-ideal sensor properties. Hence this synthetic case does not need any kind of tap calibration or image homogenization.

With a rotor speed set to the maximum possible value of $\frac{1}{4}$ rounds per frame, the application of Schmidt's detect-and-repair methods is shown in Figure 4.21. Images show the computed phase using the maximal possible number of sub-frames for each method, so average approach using all four sub-frames for the standard method (left set of two images) and using the frame subset S2 for the BID method (right set of two images). For each method, an image pair is given, with the uncorrected image (left) and the corrected one (right). Color encoded, the computed phase (range $[0, 2\pi)$) is visualized. Below each image, *relative distorted area* ϱ is printed. This number has been computed as defined above setting ω to the known value used to set up the simulated scene and choosing the radii $R_1 = 15, R_2 = 62$ pixels. The rotor center is located at $\vec{r} = (99.5, 99.5)$ pixels.

As visible, Schmidt's standard methods (left) fails completely. No artifact-affected regions are corrected at all. The "corrected" image looks almost like the uncorrected one, there are some single pixels at the edges of the rotor target where the values have changed, but the new values are worse than before. This visible behavior is not reflected by the ϱ number, since only the absolute number of artifact pixels matters and this number keeps unchanged, even if some artifact pixels changed their value but are still wrong. The value of ϱ is slightly larger than the theoretical value of 75 % for equidistant exposures because of the border pixels where foreground and background values are mixed and thus intensity and depth threshold yield a classification as artifact affected. Also note, that there are three segments on each wing of the rotor target, where the computed phase is in the correct foreground range but only the central one is classified as not affected by artifacts because in the intensity image there is an obvious mixture of foreground and background in

the other two segments. The correct phase is caused by a canceling effect of the raw-frame averaging shown here:

$$\varphi_0 = \text{atan2}(A_{270} - B_{90} + B_{270} - A_{90},\ A_{180} - B_0 + B_{180} - A_0) + \pi \tag{4.19a}$$

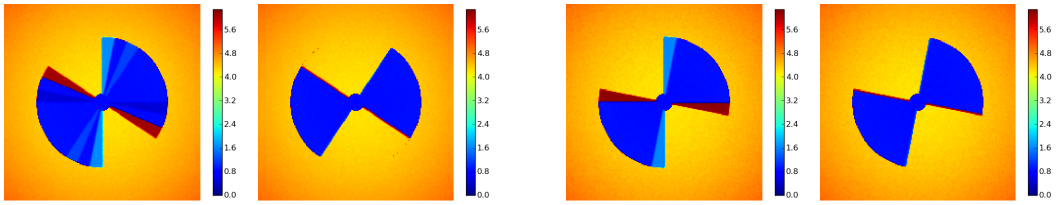
$$\varphi_1 = \text{atan2}(\cancel{A_{270} - B_{90}} + B_{270} - A_{90},\ A_{180} - B_0 + B_{180} - A_0) + \pi \tag{4.19b}$$

$$\varphi_2 = \text{atan2}(\cancel{A_{270} - B_{90}} + B_{270} - A_{90},\ \cancel{A_{180} - B_0} + B_{180} - A_0) + \pi \tag{4.19c}$$

$$\varphi_3 = \text{atan2}(\cancel{A_{270} - B_{90}} + \cancel{B_{270} - A_{90}},\ \cancel{A_{180} - B_0} + B_{180} - A_0) + \pi \tag{4.19d}$$

The rotor turns counter-clockwise. The sequence itself is shown in Figure A.8. Note that the first frame cannot be used for this detect-and-repair method since it needs values of a preceding frame for correction. Regarding Figure 4.21, there are two segments with correctly computed background depth (yellow) and two segments with correct foreground depth (blue, perpendicular to the yellow ones). These segments are not affected at all by any discontinuity events. They stay foreground/background over the full four recorded sub-frames. To understand the behavior of the artifact affected segments, the eqns. (4.19) show which values are used for depth computation using the averaging approach. The sub-frames colored blue contain foreground values, the canceled ones colored red contain background values. Note, intensity values in the foreground are about $I_{\text{FG}} \approx 6\,\text{AU}$, whereas the background pixels have intensity values of $I_{\text{BG}} \approx 0.4\,\text{AU}$ and hence $I_{\text{FG}} \ll I_{\text{BG}}$ (values given in *arbitrary units* (AU)). (4.19a) denotes the central foreground segment, where all sub-frames contain foreground values and hence computation works as expected. The next segment in clockwise direction (dark blue), denoted by (4.19b), has foreground values for the first three sub-frames and background values at the last exposure. Since background values may be neglected, this leads to a numerator value of the `atan2` arguments divided by a factor of two whereas the denominator stays the same. Hence the result of this computation gives no correct results. The next segment (blue), denoted by (4.19c), contains two foreground and two background sub-frames. The small background intensity values have almost no influence when averaged with foreground ones which effectively turns this depth computation to use the subset S1 only which yields correct foreground depth results. Regarding the last segment (red), like in (4.19b), the numerator and denominator of the atan2 are again unbalanced in (4.19d) which leads to wrong results. Similar arguments apply to the three segments at the opposite side of the central foreground segment.

The next method presented in Figure 4.21 (right image pair) is Schmidt's BID approach. This image shows the result of two methods at once. Since this method is based on the *framerate enhancement*, even the "uncorrected" image shows a much less value of ϱ (slightly larger than the theoretical value of 25 %) than the depth computed using the averaging approach as before. This shows the expected behavior of reduction of motion artifacts by a factor of three by selecting the subset S2 for depth computation. On this data, application of the BID method recovers almost all pixels still affected by motion artifacts. Only border pixels (again about 3 %) with invalid depth values remain, but this effect is caused by foreground-background mixing and not by motion.



$\varrho = 80.62\,\%$ $\varrho = 5.85\,\%$ $\varrho = 31.09\,\%$ $\varrho = 6.01\,\%$

Figure 4.22.: Slower Synthetic Rotor Sequence with Schmidt-Correction. $\omega = \frac{45°}{\text{frame}}$. Left pair: Phase computed using all eight raw images (average) before and after Schmidt's standard method. Right pair: Phase computed using second subset (S2) before and after Schmidt's BID method. With this slower rotor speed, both methods work well.

Reducing the rotor speed to half of this value, the precondition of at most one discontinuity event per two frames also holds and so the standard approach works too. With fast speed, there is one discontinuity per frame, so the precondition for the standard approach is violated, but for the BID method, which is able to handle up to one discontinuity within the four sub-exposures, the precondition holds for fast and slow rotor speed. This is shown in Figure 4.22. The visualization is the same as before in Figure 4.21, but the angular velocity is now $\frac{1}{8}$ rounds per frame. All other parameters are unchanged. Note that the total number of border pixels is also the same as before, but since the rotor speed is reduced to half the value of before, also the maximal possible area $A_{\max}$ is also halved, doubling the percentage of border pixels. Despite the fact, that now, both variant of Schmidt's detect-and-repair methods visibly work as well as the BID variant with the fast rotor speed, the values of $\varrho \approx 6\,\%$ are twice as large.

After Schmidt's methods, the next class of detect-and-repair methods is based on optical flow estimation between the sub-frame intensity images applied to the same rotor sequence used before in this section (cf. Figure A.8). Lindner and Kolb [LK09] estimate the flow on three image pairs consisting of the first sub-frame intensity image and all others. The results of this estimation are given in Figure 4.23. The leftmost column shows the ground truth optical flow of the first image pair which is a pure rotation field on the rotor area and zero flow in the background. This field has been exported directly from the *Blender* software used to design the synthetic test scene. The visualization shows the magnitude and orientation of the flow vectors encoded in HSV colors as well as with small vector arrows (quiver). Ideally, all results of real flow estimations should look like this ground truth frame, identically for the first image pair and with larger flow values but same shape for the other pairs because of the larger angle difference. Note that e.g. doubling the angle difference does not simply double the flow values but changes as well length and orientation of the flow vectors. This field visualizes the effect of applying a rotation matrix to the pixel coordinates.

The central part of the same Figure 4.23 shows the actual flow estimation results in the three flow visualization images, one for each sub-frame intensity image pair.

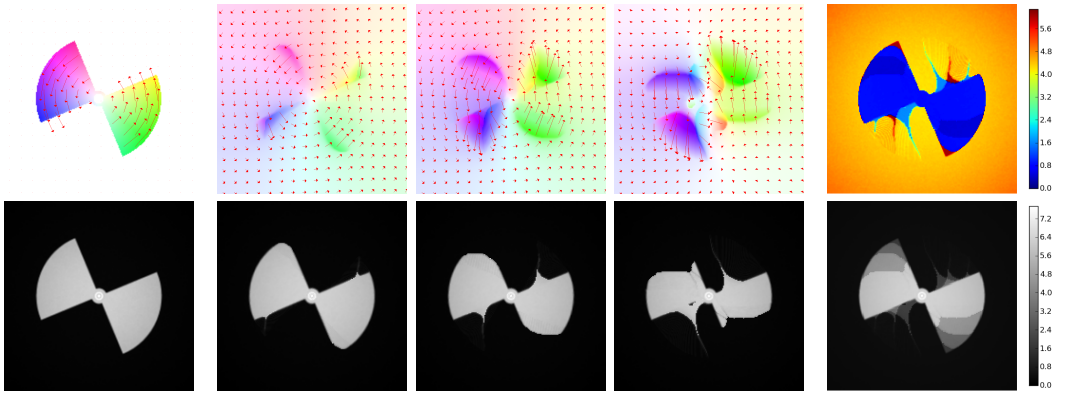


Figure 4.23.: Fast Synthetic Rotor Sequence with correction using Optical Flow based method by Lindner & Kolb. **top**: flow ground truth of first pair (**left**), flow fields between first and the other three intensity images (**center**). Phase computed after correction (average approach, **right**). **bottom:** first intensity image used as reference (**left**), following three intensity images warped with estimated flow shown above (**center**). Sum of all intensity images after correction (**right**). Ideally all warped images in this row should look like the image on the left. After Correction: $\varrho = 26.90\,\%$. Using a non-textured target, this method fails.

Obviously, the used non-textured fore- and background lead to non-zero flow also in the background area because of regularization. Note that a pure rotational flow field for the whole image (not only the rotor area) would perfectly solve the optical flow constraint (cf. [Bro+04, eqn. (1)], reconstruction with this flow field would work) because of this lack of texture. Visibly, the results look quite bad, only at the rotor borders, there are areas with remarkable flow magnitudes. All other values are only caused by regularization since all image derivatives usually needed in optical flow data terms are vanishing inside the non-textured areas. Essential parts are the segments affected by motion artifacts, but even the shape of these segments is not covered well, especially at the corners. This fact may not be explained by the aperture problem that may influence the results at straight borders. On the second and third image pair, this behavior becomes even worse, the flow magnitude at some of the corners are way too low and there are strange transitions within the artifact affected segments introducing spikes and deforming the rotor shape when applying these flow fields to correct the sub-frames. This is shown in the bottom images, where the flow fields are used to warp the intensity images of sub-frame 1,2 and 3. Ideally, this should look exactly like the reference sub-frame 0 shown in the leftmost image of the second row (the input sequence with the sub-frame intensity images is shown in Figure A.8, here, frame 0 is used). Using these results to compute ToF depth and intensity images, the images in the rightmost column are generated, depth image at the top and intensity image at the bottom. Although the flow results are visibly bad, the computed quantity $\varrho \approx 27\,\%$ is in the same range as the *framerate increase* method.

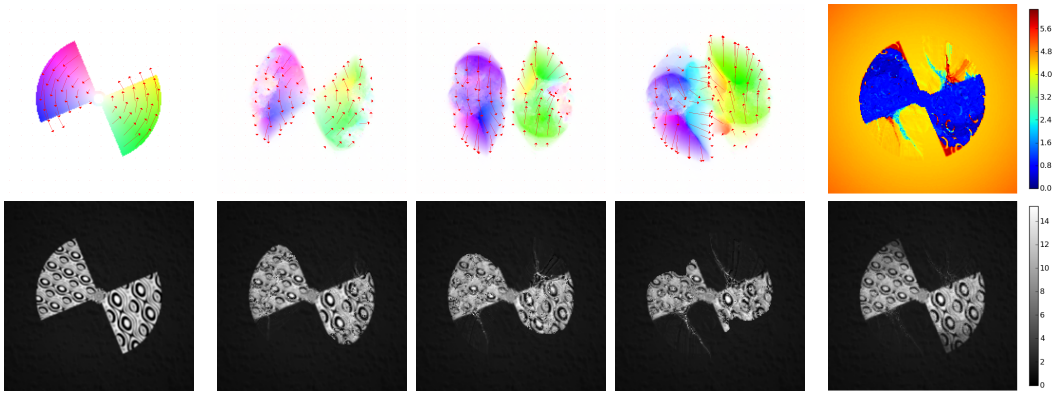
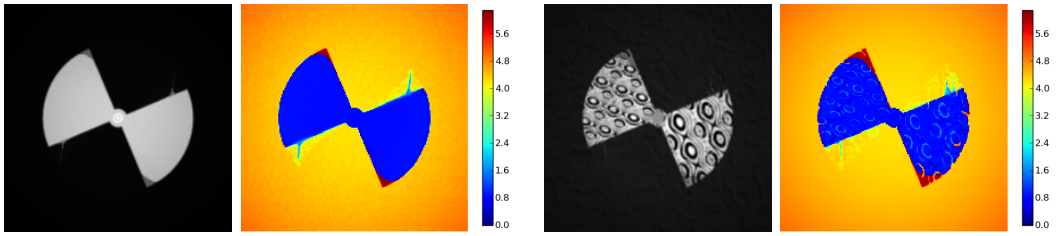


Figure 4.24.: Using a textured rotor target and background, zero flow in the background is estimated correctly, but the foreground flow result (rotation) is still wrong. It is also visible that the warped results of the last two subframes are especially bad as well as with as without texture. Here, $\varrho = 42.28\,\%$.

To add more hints to the optical flow estimation algorithms, the same analysis has been conducted with additional texture applied to the fore- and background in Figure 4.24. The ingrain wall paper texture of the background is hardly visible because of the low light intensity but causes correct estimation of the zero optical flow there. With the textured rotor, the area of perceptible flow magnitudes is larger than before, especially the result of the first image pair looks well. But the problems at the rotor corners and the strange spikes and structures within the rotor area still persist. Whereas the reconstructed intensity image (bottom right) looks acceptable, the depth image (top right) also shows the spike shaped artifacts and additional wrong results at dark texture regions. Since the amplitude of the returning ToF signal is low there, averaging with values of the surrounding pixels happening during the warping (which needs interpolation of float valued pixel coordinates and thus positions between the regular pixel grid) may be considered as an additional source of noise which has a strong impact there. The value $\varrho \approx 42\,\%$ reflects these effects and is in the range between the uncorrected case ($75\,\%$) and the non-textured or *framerate increase* case ($25\,\%$).

Since the flow estimation results get worse with larger displacements and larger rotations as shown above, it seems to be promising to only use the first sub-frame image pair for reconstruction. This is the idea of combining the *framerate increase* method by Schmidt and the flow based method by Lindner and Kolb. Avoiding the last two sub-frames restricts the available raw-frame data to the subset S1 for depth computation. So starting with a value $\varrho \approx 25\,\%$ for the uncorrected case, warping the second sub-frame with the optical flow as estimated before should at least improve the results. By only applying the optical flow estimation on one image pair, this also dramatically speeds up this method by a factor of three. Additionally, this is a really simple approach not needing any advanced techniques like proposed by [LHK13] to get this speed benefit.



$\varrho = 7.06\,\%$ $\varrho = 14.83\,\%$

Figure 4.25.: Combination of Lindner&Kolb method with Schmidt's framerate enhancement on rendered sequence with non-textured (**left**) and with textured surfaces (**right**). Here, only the first image pair (subset S1) is used for depth computation, so only the flow field between this pair is needed. This heavily reduces the artifact strength compared to usage of all subframes above. Note: This does not include any of Schmidt's correction methods (standard/BID) mentioned above which are not applicable here.

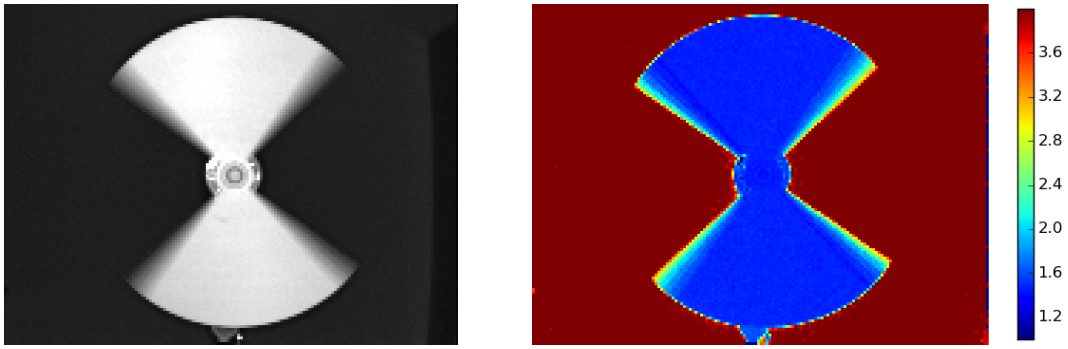
The resulting intensity and depth images are shown in Figure 4.25. The used flow fields are the same as in Figure 4.23 resp. Figure 4.24. As the values of $\varrho \approx 7\,\%$ for the non-textured case and $\varrho \approx 15\,\%$ for the textured case show that this combined method performs almost as well on this test target as the detect-and-repair methods by Schmidt. But note that the flow based methods are not limited to the assumption of at most one discontinuity event per two time periods. So potentially they are applicable even on highly textured scenes with large displacements but at the other hand, they introduce all the difficulties related to optical flow estimation like aperture problem, occlusions etc. where wrong flow results are very likely.

Real Sequences

Simulated or rendered data avoid a lot of the disturbing effects present on real ToF devices like sensor inhomogeneity and non-linear responses. Therefore it is also necessary to compare the algorithms with sequences recorded by real devices.

To point out that motion artifacts are not a problem of particular hardware, it is important to check if such artifacts are present in different ToF devices, ideally by different manufacturers. As a reference camera serves a *Fotonic E40P* device with a resolution of 160×120 pixels. This device does not provide access to the recorded raw data, so the algorithm presented above cannot be applied here. So this used to demonstrate the artifact presence and strength in the data as provided by the manufacturer's SDK. As shown in Figure 4.26, there are similar staircases as visible in the simulated data, but there seem to be more than four sub-exposures per frame (and so steps at the borders). The average value $\varrho_m = (29.87 \pm 0.79)\,\%$ may be caused by a burst-like exposure scheme, as discussed above for the PMD device.

For analysis of the discussed methods, raw data access is needed. Therefore the rotor test target has been recorded with the PMD *CamCube3* device. To avoid arbitrary ϱ values influenced by the requested FPS rate, the sequence has been recorded in the *freerun* mode of the device. Using this mode, the recorded sequence should



$$\varrho = 29.01\,\%$$

Figure 4.26.: Artifacts occuring using *Fotonic E40P* device with Panasonic chip. Median artifact strength of the whole sequence is $\varrho_m = 29.75\,\%$. Note that staircase effect at the borders is also present here but looks different than on PMD data.

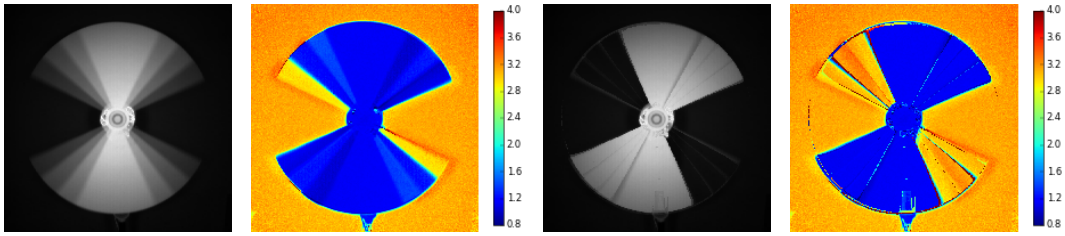
have similar properties as the synthetic ones presented before, i.e. sub-exposures captured in equidistant time steps, even across different frames. An integration time of 1700 µs has been chosen such that no frame-drops occurred during the records. The recorded sequence used for the following evaluations consists of 100 frames with four sub-frames each. To get reasonable ϱ values, only frames where the mount is covered by the rotor have been used since this region is always classified as artifact affected since it does not fall into the foreground or background thresholds. With these criteria, about 33 of these 100 frames are considered as "good" frames. ϱ_m values are the median of the ϱ values of all these "good" frames.

Results of Schmidt's standard approach are shown in Figure 4.27. Left image pair shows an uncorrected frame of this recorded sequence. As already noticed in the synthetic sequence, the artifact affected regions are slightly larger than expected theoretically. As before, this is caused by mixing of foreground and background at the border pixels but also by motion blur present in each sub-frame. Note that this blur is caused by the movement during the integration time and is fundamentally different from the motion artifacts caused by motion between exposures.

The right image pair shows the corrected frame. Even if the prerequisites for this method are not strictly fulfilled by this sequence (determined angular velocity $\omega_F \approx 0.92$ is a bit larger than $\frac{\pi}{4} \approx 0.785$), this method visibly does correctly detect the affected segments and the corrected values are closer to the correct foreground/background than before. The borders seem to be problematic, artifacts are still visible there as well in the intensity as in the depth image. This is also reflected in the value $\varrho_m \approx 22.4\,\%$. Regarding the fact that the last time step of the eight has no effect within the two considered frames, this leads to a slightly higher angular velocity this method is able to handle:

$$\omega_{F,\max} = \frac{\pi}{4} \cdot \frac{8}{7} \approx 0.898 \frac{\text{rad}}{\text{frame}} \approx \frac{51.43°}{\text{frame}} \tag{4.20}$$

Since the determined velocity is only slightly higher, this could explain the still



$\varrho_m = 80.30\,\%$ $\varrho_m = 22.42\,\%$

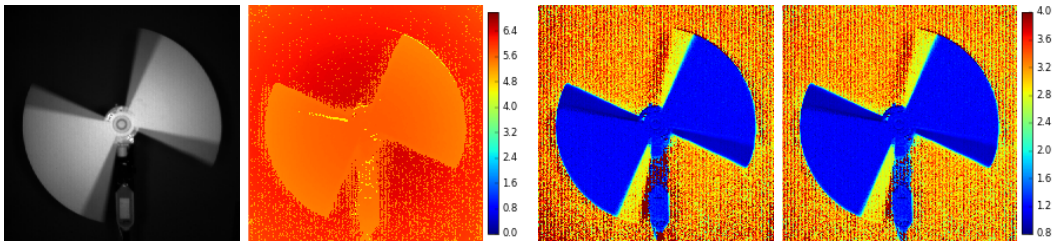
Figure 4.27.: Schmidt's standard correction on real rotor sequence recorded with the PMD *CamCube3* device. Before correction (**left pair**) and corrected (**right pair**). ϱ_m values are the median of the ϱ values of 33 good frames of the sequence. The depth images show the z-distance in meters. This method works here, but correction of the artifacts in the background area is not optimal.

present artifacts at the center of the foreground and background regions.

As a base of the subsequent methods, results of Schmidt's *framerate increase* are given in Figure 4.28. To show the importance of the preprocessing (tap calibration), this method has been applied to the raw data as-is (second image). The computed depth values are way off from expected results. Virtually all pixels have been classified as artifacts, even the ones not covered by any depth discontinuity event. This is the reason of the high value $\varrho_m \approx 165.6\,\%$. Also note the different color-bar ranges. Mixture of values of the different taps, as it is done here by selecting the raw data subset S2 does not work at all without calibration.

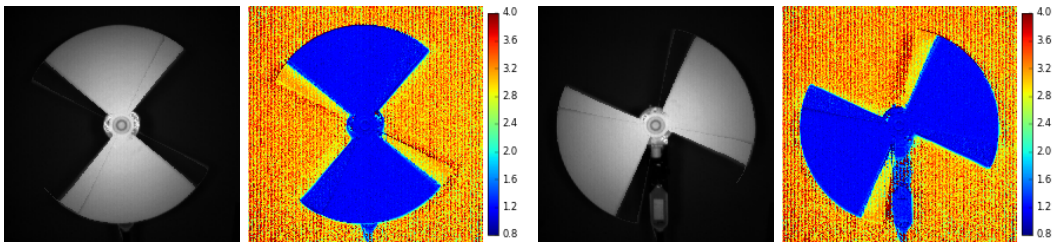
With tap-calibration, there are the two methods presented in Subsection 4.3.1, one using a linear approximation and the more sophisticated non-linear variant for lower intensities. Linear is shown in the third image, the proposed non-linear variant in the last image. The median ϱ values of the variants ($\varrho_m = (33.64 \pm 0.25)\,\%$ vs. $\varrho_m = (32.86 \pm 0.15)\,\%$) differ remarkably, but visually, the effect may be seen best in the region of the mount, below the rotor center. In this region with low intensities (which has been excluded for ϱ computation), the estimated depth values are closer to reality using the non-linear variant. Using the linear method, the upper part of the mount is mis-estimated in a distance of about 4 m. To show this difference, frame 4 of this sequence has been used for visualization, even if this is not considered as "good" frame for ϱ computation. The values itself are higher than theoretically expected ($\varrho = 0.25\,\%$), caused by the effects already discussed (borders and motion-blur). Also note that there is much more noise, visible with a striped pattern and much stronger in the background area where intensities are low compared to depth computation with the average approach using all captured raw data.

Results of Schmidt's BID variant are given in Figure 4.29. Visibly, this method works almost perfect for intensity image reconstruction, regarding the depth images, some artifacts stay at the corrected background segment but values are better than in the uncorrected case (tap calibration only, shown in Figure 4.28). Also mixing values at borders and motion blur may not be eliminated by this method, explaining



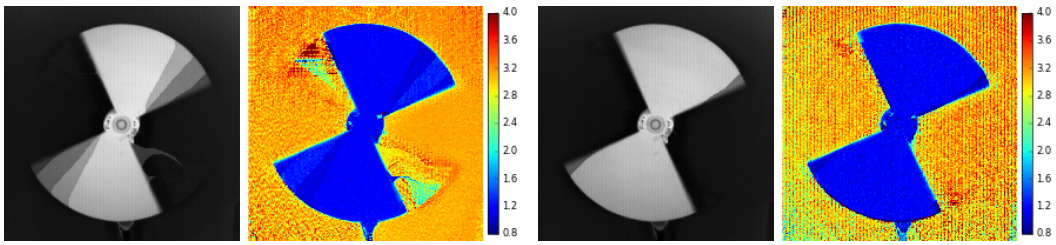
$\varrho_m = 165.56\,\%$ $\varrho_m = 33.64\,\%$ $\varrho_m = 32.86\,\%$

Figure 4.28.: Real rotor sequence (*CamCube3*), framerate increase, all images computed using the subset S2. Intensity image (**left**), depth in meters computed using no calibration (**second**), with linear calibration (**third**) and proposed combined calibration (**right**). Proposed calibration works best at darker regions.



$\varrho_m = 12.26\,\%$

Figure 4.29.: Real rotor sequence (*CamCube3*), with Schmidt BID correction, all images computed using the subset S2. Image pairs representing frame 2 (**left**) and frame 4 (**right**, to compare with Figure 4.28). For each pair: Intensity image (**left**), depth in meters (**right**). Correction works better than using the standard approach.



$\varrho = 36.17\,\%$ $\varrho = 11.80\,\%$

Figure 4.30.: Real rotor sequence (*CamCube3*), with applied flow based corrections. Depth computation by averaging as proposed by Lindner and Kolb (**left pair**). Combined with *framerate increase* i.e. depth computed using subset S1 (**right pair**). For each pair: Intensity image (**left**), depth in meters (**right**). Proposed combination performs best.

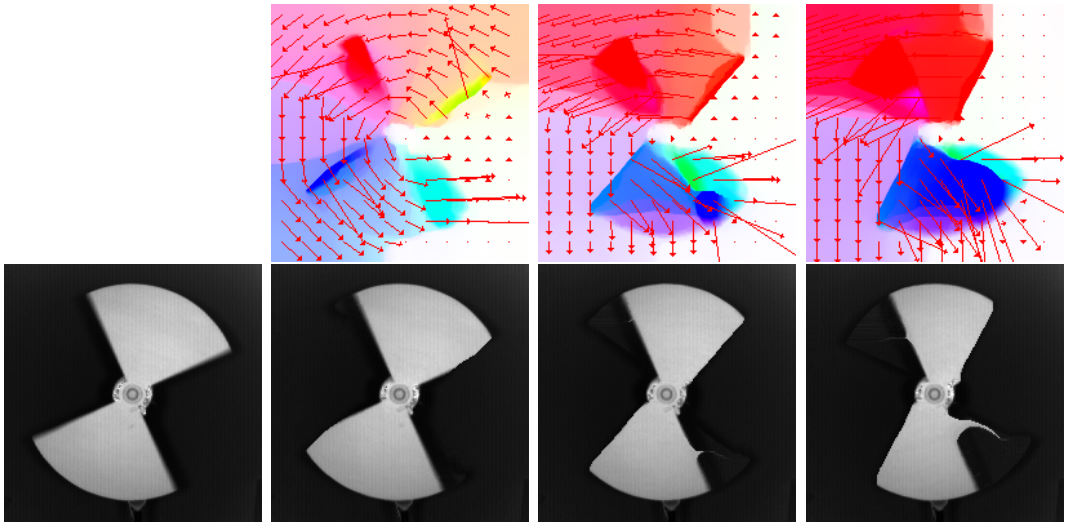


Figure 4.31.: Estimated optical flow results used in the artifact compensation method by Lindner and Kolb and its combination with the *framerate increase*. First row gives visualizations of the three computed flow fields with corresponding warped intensity images below. Bottom left image is the reference sub-frame 0. Ideally, the subsequent images should look like this reference which is obviously not the case. First sub-frame pair works best.

the value $\varrho \approx 12.3\,\%$ (compared with $3\,\% \ldots 6\,\%$ in the simulated case and expected theoretical value of zero). The figure shows two different frames out of the same sequence, frame 2 (left) is a "good" frame for ϱ computation, frame 4 (right) is shown to compare with the previous method.

Flow based method by Lindner and Kolb applied to the real rotor sequence is shown in Figure 4.30. The left image pair visualizes the unmodified implementation, i.e. computation using all eight raw images and three optical flow estimations. The actual visualizations of the flow images are given in Figure 4.31. Regarding these flow results, with larger the time difference and so the rotation angle $\Delta\varphi$, results get worse. Interestingly, the brightness change from background to foreground performs better than the opposite site of the rotor, but even there, spike-like artifacts occur. After correction, the staircase-like artifacts present in uncorrected images are clearly visible in intensity and depth images. This method fails there completely, but also at the better side of the rotor, the depth results are not correct. Combined with the *framerate increase*, i.e. dropping the worst two of the three estimated flow fields, this method performs remarkably well. The value $\varrho \approx 11.8\,\%$ is the best result of this comparison working on the real sequence. Only results of Schmidt's BID method are in the same range. This is no surprise since they share the same tap calibration and subset selection for depth computation. The also have in common that because of not averaging all raw frames, the strong artifacts in the background area persists, but seems to be a bit reduced using the flow method.

Chapter 5.

Conclusion and Outlook

In this thesis, dynamic sequences recorded with depth imaging devices of different kinds have been considered. Depending on the device principle, several kinds of artifacts may be observed. On structured light data, the sequences are used to estimate the 3D motion of recorded scenes, called *range flow*. The present artifacts require special adaption of the proposed algorithms. On Time-of-Flight (ToF) data, motion in the recorded scene is the source of errors in the computed depth maps that may be reduced or even eliminated by the methods considered in the second part of this work. This survey includes a qualitative and quantitative evaluation of existing methods. Additionally a novel tap-calibration approach is introduced, improving the results of existing methods and proposes a novel motion compensation approach combining the different ideas of the methods compared before.

Regarding *Kinect* data in Chapter 3, a novel framework for robust range flow estimation on sequences from multi-modal RGBD data has been presented. Artifacts within this data are inherent to the device principle and consist of unstable depth edges and holes with invalid depth information. Calibration with a novel alignment algorithm with a back-ward mapping scheme provide a general and robust solution to register the color and depth images provided by the hardware with means of image processing (in contrast to dealing with 3D point clouds), which can also be applied to a wide range of other applications. The presented range flow estimation provides stable flow fields and is able to cope with the systematic errors induced by the hardware setting. Hence, this framework provides a useful middle-layer for the further development of high level algorithms for the *Kinect*, which has happened in the meanwhile. The corresponding paper [GFG11] has drawn attention in the community (at least 40 citations at the time of writing) since it was one of the first dealing with this topic and especially the novel alignment approach made estimation of motion on *Kinect* data feasible at all.

Future tasks could be generation of a ground truth dataset for *Kinect* data similar to e.g. the Middlebury database [Bak+07] for a quantitative analysis of the presented methods and development of more sophisticated range flow algorithms incorporating state-of-the-art optical flow methods combined with data terms based on the presented range flow motion constraint (RFMC). Especially the latter has been addressed by more recent publications but there still lacks a ground truth database.

The good of Chapter 4 is a survey of all purpose methods to cope with motion artifacts in Time-of-Flight images. This includes implementations of all methods and an analysis on real and synthetic ToF sequences as well visually as quantitatively. By combination of the methods a novel accurate and fast method is proposed.

Schmidt [Sch11] made an outstanding contribution to this field of research since he proposed at least three different methods and the only existing measure for quantitative analysis. This explains why those cover large parts of the survey.

A central part is the *tap calibration* which has been extended by a novel variant to work with the non-linear behavior of the present capturing device. An *exposure ramp* analysis, sweeping the range of adjustable integration times revealed these details of this inter-tap behavior invisible in the dynamic calibration proposal by Schmidt. The novel calibration variant yields visibly better results, especially in the dark regions since the deviations from linear behavior mostly occur at low intensities.

This calibration allows to use only two out of four exposures per frame reducing motion artifacts by a factor of three (*framerate increase*). This method leads to quite well results even in presence of strong motion and should be sufficient for a large class of applications. Compared to the usual averaging approach of depth computation, this method does cancel the benefit of averaging the values of both taps and hence introduces additional noise in the depth maps, especially at dark regions. The proposed non-linear calibration performs visibly better in these regions but the heavy noise persists. If the remaining artifacts are still too strong, it should be used at least as a base for further processing.

Schmidt's detect-and-repair methods (proposed in two flavors) work well when there are static background regions and low textured foreground object moving in lateral direction. These conditions virtually enforce a statically mounted camera (or a really non-textured background at constant distance from the camera). If this is fulfilled, these methods yields almost perfect results using very cheap computations. The standard variant requires image sequences of multiple frames and uses information of the preceding frame for correction (hence the first recorded frame will always stay uncorrected). The BID variant works more robust and is even applicable on single frames since it only works on the individual sub-frames. It uses the tap corrected data and replaces discontinuities in the second subset by values from the first one. It is able to handle depth discontinuities occurring at a two times higher rate than the standard variant can.

More expensive but better results are given by the flow method by Lindner and Kolb applied to the sub-frames. Such flow based methods work better with more textured scenes and are also able to cope with sequences recorded by a moving camera. Combining this with the *framerate increase*, thus only considering the first subset of raw-frames, causes small displacements which simplify optical flow computation and yield good results. Additionally only one flow field has to be computed instead of three (as in the traditional approach using all four exposures).

Pixel-based correction and the flow based methods both have pros and cons. Flow based methods are able to cope with a moving camera but introduce expensive computations (need for GPU to get real-time) and artifacts present in flow estimation.

But combined with the *framerate increase*, this method outperformed Schmidt's BID variant on the real sequence and yields slightly worse results in the synthetic sequence.

Regarding the design of the quantitative measure ϱ reflecting the *relative distorted area*, the segmentation of a foreground and background by thresholding intensity and depth values enforces a test target with perfect preconditions for the pixel based detect-and-repair methods. The benefit of the flow based methods like the ability to handle textured scenes with heavy motion may not be reflected by this measure but is clearly visible in the qualitative analysis. But even despite the fact that the rotating test target infers additional difficulties to the optical flow estimation the proposed combined framerate-increased flow method performs almost as well as Schmidt's BID variant on the test sequences.

During the experiments with real data provided by the PMD *CamCube3* device systematic differences between computed depth values by the manufacturer's SDK and using the formulas from literature have been discovered. By reverse-engineering, the black-box SDK behavior could be recovered, enabling the possibility to apply the same computations to processed raw data. This way it is possible to compare the output of the presented motion compensation methods with values computed by the SDK. Additionally, this gives inside into the corrections e.g. of the depth-wiggling-effect taking place there but also reveals some questionable design decisions of this pipeline that may be considered as bugs (e.g. application of the *CamCube2* intrinsics to *CamCube3* data).

To automate the quantitative analysis using a rotor target as proposed by Schmidt, an automated position detection and evaluation pipeline has been developed. As an application, the exposure pattern of the *CamCube3* device has been investigated revealing different properties like a burst-like behavior in triggered mode and equidistant time-steps in freerun mode. By developing a tuned variant of the recording software, the high frame-rates present using this freerun mode could be handled (up to 50 FPS), avoiding frame-drops for a large range of integration times. Using this software, additional properties like the delay t_d between exposures could be determined.

These results provide additional insight in the used device and especially the modeled PMD processing pipeline may act as a base for future research since it is applicable to raw data which does not to reflect the manufacturers data format with its specific and partially undocumented meta-data.

To summarize the results of various motion compensation methods discussed in this work, a decision guide given in Figure 5.1 may help to select the right algorithm for a specific application.

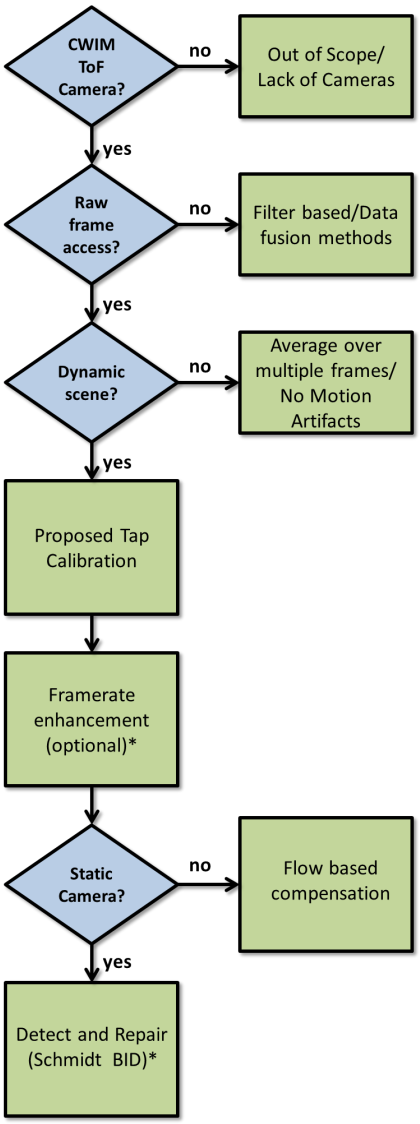


Figure 5.1.: Selection guide on ToF motion compensation methods. Methods marked with an asterisk (*) are real-time capable (even without sophisticated GPU processing).

Appendix A.

Time-of-Flight Data – Extended Details

In this chapter, further plots and results provide additional insight in the discussed methods. The Section/Subsection numbers have been adjusted to match the corresponding ones of Chapter 4, e.g. Subsection A.2.1 matches Subsection 4.2.1.

Additionally, the rendered rotor sequence used in Subsection 4.8.4 is shown in Section A.4 and some of the used source code files are presented in Section A.5.

A.2. Preprocessing and Calibration

A.2.1. Tap Calibration

To show the difference between linear and proposed tap B reconstruction formula (r_1 vs. r_c), Figure A.1 is an extended version of Figure 4.8. The second row shows the same phase estimations as the third but with linear reconstruction. In the averaged phase estimation, the regions with low intensity (floor, box at bottom center) are reconstructed better using the combined approach than the linear one. It also shows that least of all artifacts at dark regions occur *without* any of the presented reconstruction formulas (top row). But in this case, the subsets may not be used for phase computation (S1, S2). So one has to choose between motion artifacts (averaging all raw frames) or reconstruction artifacts at low intensities (using subset methods).

A.2.2. Image Homogenization

Application of the homogenization to the calibration frames from the *exposure ramp* is shown in Figure A.2. How this homogenization affects the intensity frames $I_i = A_i + B_{i+180}$ (used e.g. to compute the optical flow) is shown in Figure A.3. Before homogenization, the intensity images show line artifacts (best visible in Figure A.2) which are removed during homogenization. Also, the brightness change towards the borders is decreased and almost removed applied to the calibration frames (as expected). Since the brightness also depends on the object distance, this correction does not work that well applied to the real-world *office sequence*. The remaining brightness change (after homogenization, the borders look brighter) is still less than before and does not influence the flow computation since it is of a much larger scale than the object structures. The main benefit of the homogenization is the removal of the fixed-pattern-noise.

A.3. Evaluation of Motion Compensation

A.3.2. Detect and Repair Methods by Schmidt

To provide a deeper insight of the BID algorithm by Schmidt[Sch11], Figure A.4 shows the pixels, where pixel values from the second subset have been corrected. In this variant of the algorithm, correction is a binary process, only values from the last sub-frame (i.e. A_{270}, B_{90}) are replaced (by B_{270}, A_{90} from the same frame), others are not touched at all. Using the standard approach, arbitrary combinations of the sub-frames (except A_0, B_{180}) may be replaced by values from the preceding frame. So this Figure shows only the BID variant.

A.3.3. Optical Flow to Warp the Sub-frames

Results of the different optical flow methods are given in Figure A.5 and Figure A.6. Regarding the residual values and the smoothness of the computed flow fields, the TV-L^1 method turns out to be the algorithm of choice.

Implementation Modules for GPU optical flow computing have been wrapped as Charon-Suite modules[GK12]. Figure A.7 gives an impression how the flow method by Lindner and Kolb [LK09] has been implemented in a modular way. At the left, `crop1`...`crop4` select the intensity frames $I_1 \dots I_4$ (using given regions ROI). The optical flow between these frames is then computed by `estimate`...`estimate3`. The three flow results are then combined to a warping map by `images2sequence4`. The raw frames (tapA/B at all four phase shifts) are then warped with the flow result by `simplewarp4`. `pmd_tapselect1` chooses the phase estimation scheme (the ones presented e.g. in Figure A.1) and `pmd_phasefromraw1` performs the phase computation. The rest of the modules serves for displaying, data I/O or converting between different image formats.

Runtime Information Since the images are of size 200×200, all tested algorithms have no noticeable execution time using their Cuda/OpenCL implementations. The full workflow used during the experiments takes about 0.86 s on a PC with an Intel I5-2500 CPU and a NVIDIA GeForce GTX 560 Ti GPU, including the flow calculation for all 3 time steps, warping the raw frames with the flow results, data rearrangement and phase calculation. Note that only the optical flow computation itself has been computed using the GPU. According to Lindner and Kolb [LK09], more than 10 FPS should be possible, but this has not been of scope for this work.

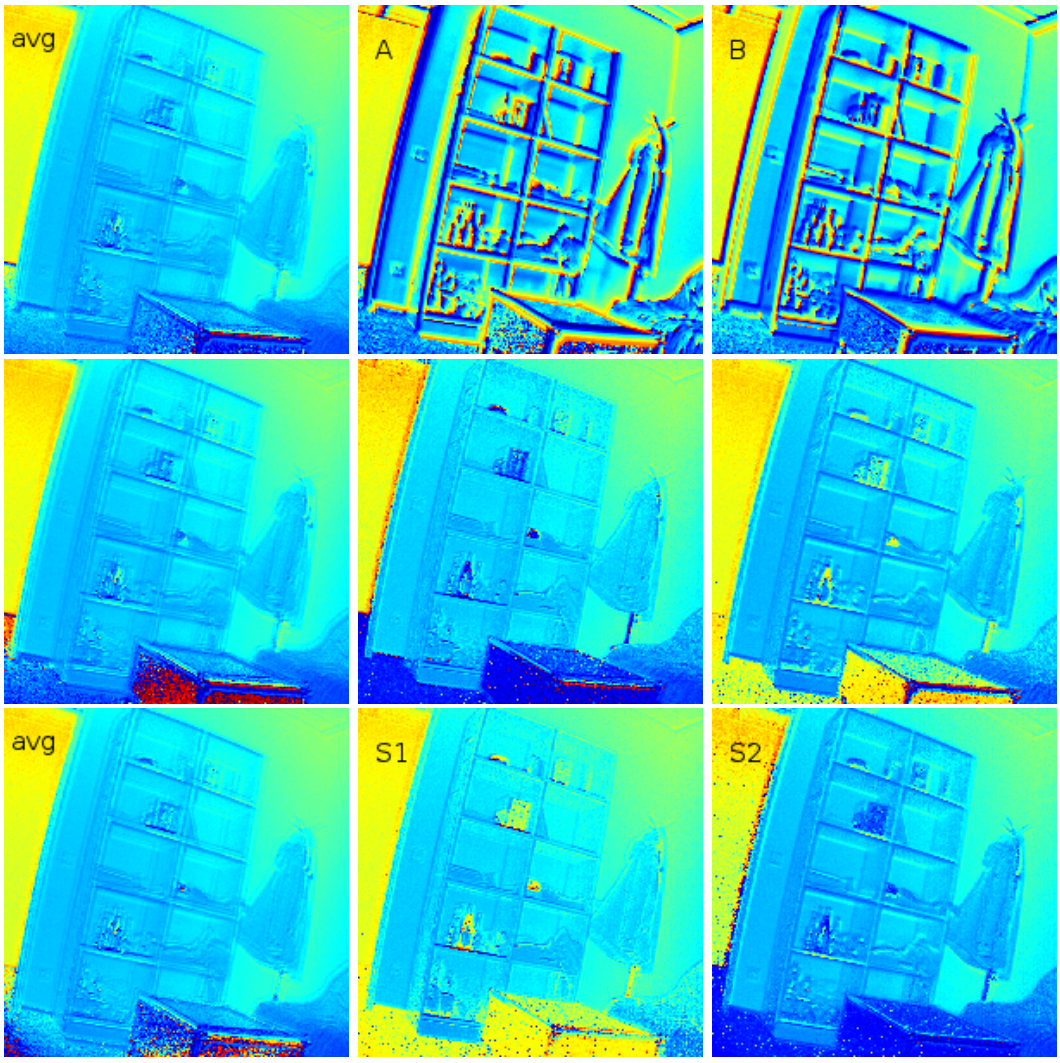


Figure A.1.: Phase estimation of a dynamic scene using different subsets of the captured raw data. **Row1:** All data as given by the device, without tap calibration. Depth of all eight raw images using averaging approach (**left**), all four tap A frames (**center**), all four tap B frames (**right**). **Row2:** All data with application of r_1 (linear) to the tap B records. Averaged depth of all eight raw frames (**left**), first subset of A and B raw frames (**center**), second subset of A and B frames (**right**). **Row3:** All data with application of r_c (combined poly5/linear) to the tap B records. Averaged depth of all eight raw frames (**left**), first subset of A and B raw frames (**center**), second subset of A and B frames (**right**). S2 results (bottom right) look best.

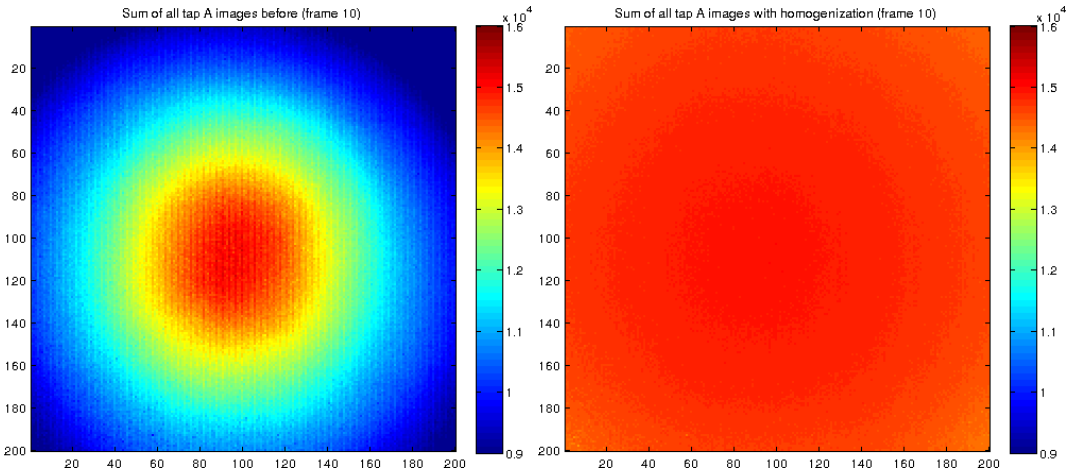


Figure A.2.: raw-frame homogenization applied to intensity image of the *exposure ramp* (frame 10). Both images use the same value visualization range (colormap). Line artifacts and brightness decrease to the borders are almost completely removed after homogenization.

Figure A.3.: Raw intensity frame I_1 before (**left**) and after (**right**) homogenization. Like in Figure A.2, the fixed pattern noise (stripes introduced by tap reconstruction) is removed. Now the brightness even increases towards the edges. This is a slight over-correction because of different distance of the scene objects and the flat wall of the calibration frames but does not influence the flow computation.

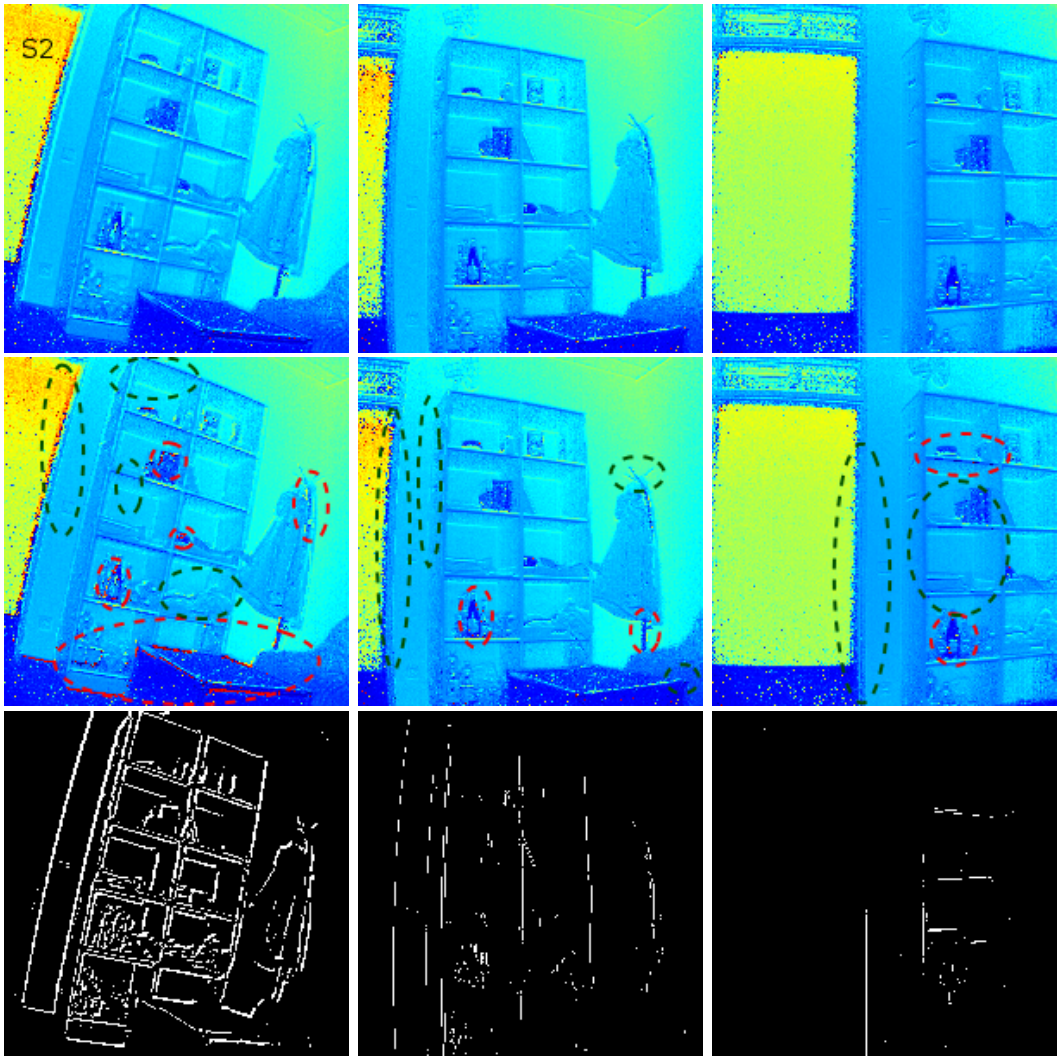


Figure A.4.: Motion compensation method by Schmidt (BID approach). All pictures show phases computed using second subset. Value range is the full non-ambiguity range R_N **Row1:** before correction. **Row2:** with BID correction and marks of good/bad (green/red) results. **Row3:** pixels changed by BID methods (white). Same picture as in the paper but with additional correction marks (bottom row).

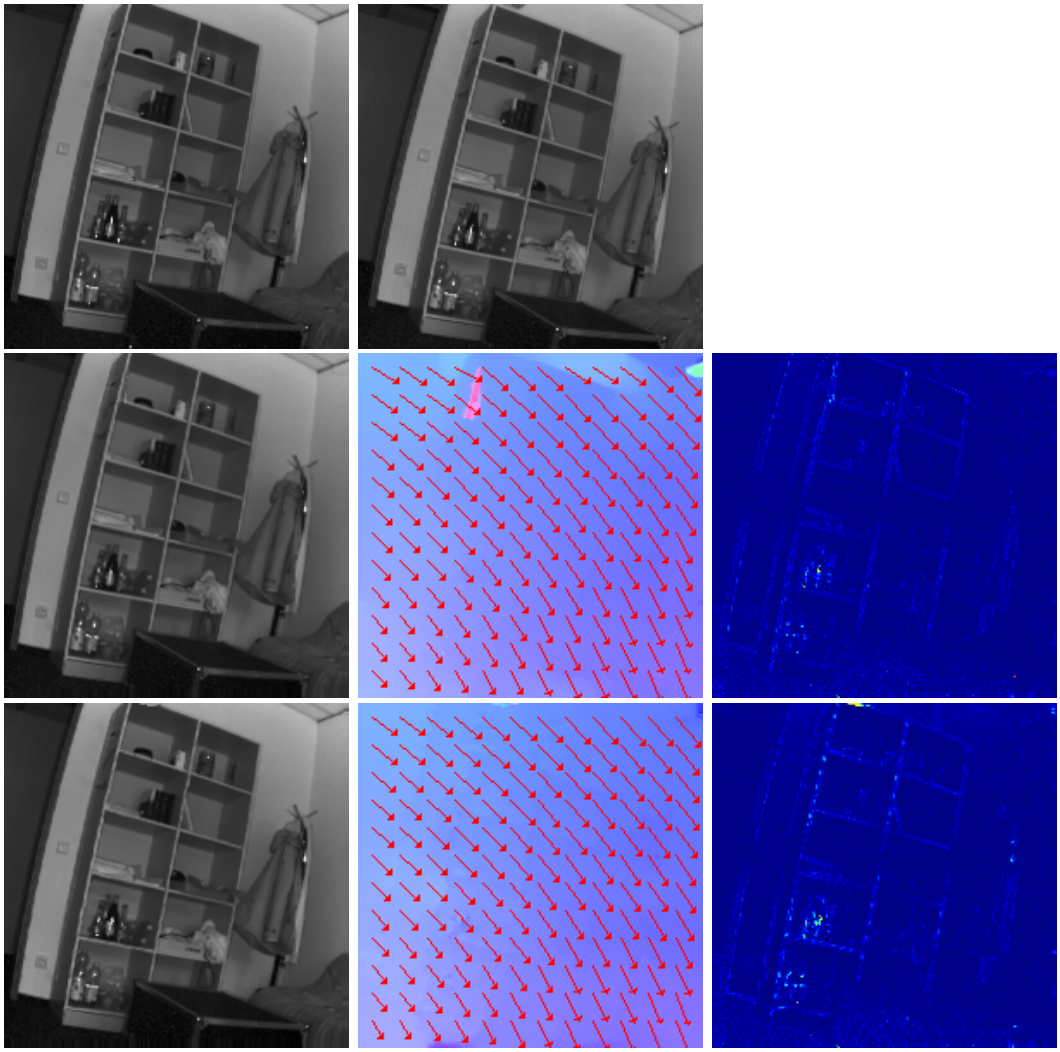


Figure A.5.: Optical Flow between first (I_1 **top left**) and last (I_4 **top center**) sub-exposure (homogenized tap A+B). **Rows:** Different optical flow methods: **Row1:** used data **Row2:** TV-L^1 [ZPB07] **Row3:** Brox [Bro+04] **Colums:** (except first row) warped frame I_4 **(left)** flow visualization (**center**) residual (difference between warped I_4 and I_1, **right**). TV-L^1 method works best (low residual). Continued in Figure A.6.

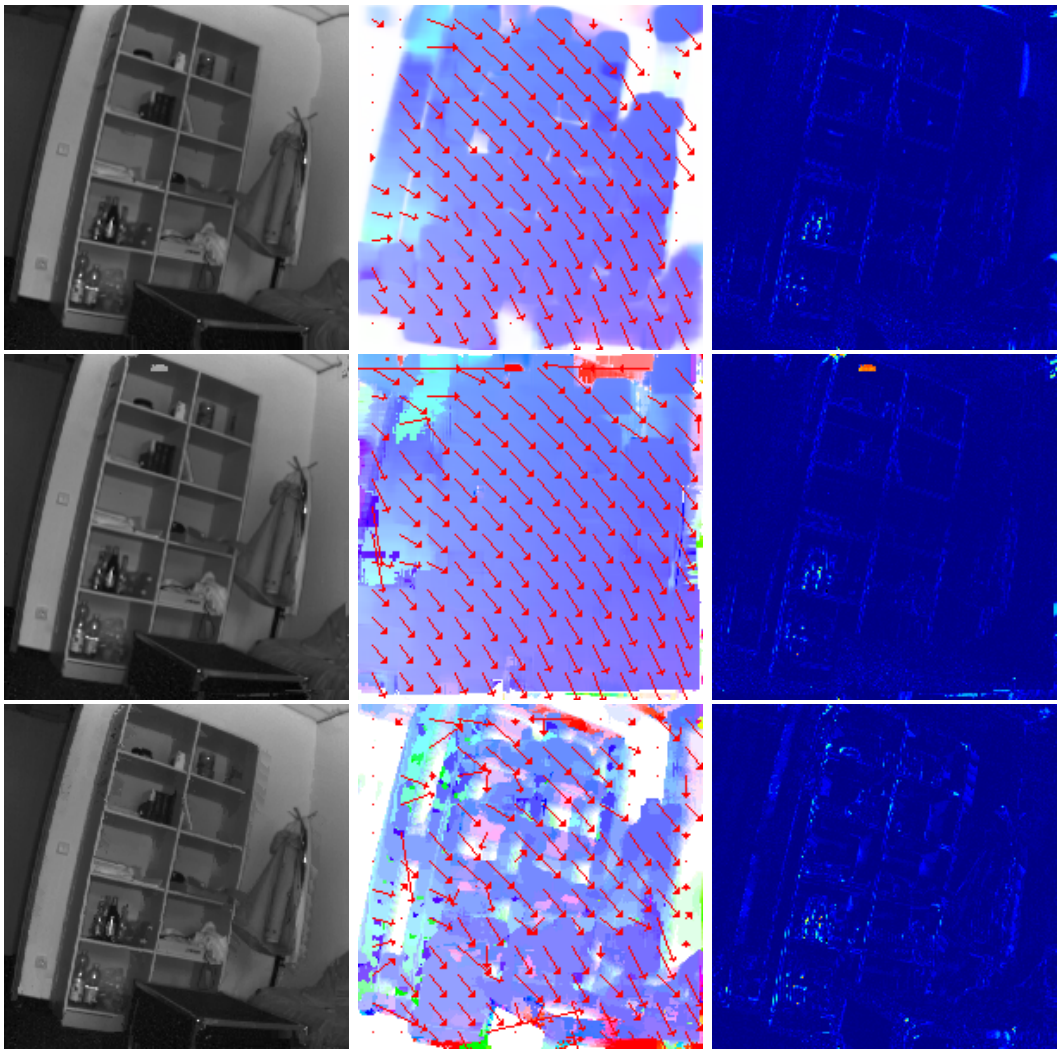


Figure A.6.: Continuing Figure A.5. **Rows:** Different optical flow methods: **Row1:** Farneback [Far00] **Row2:** PyrLK [LK81] **Row3:** Block Matching **Colums:** warped frame I_4 **(left)** flow visualization (**center**) residual (difference between warped I_4 and I_1, **right**). All results presented here have higher residual than in Figure A.5. Also, the flow visualization show more discontinuities and regions without or with wrong flow values.

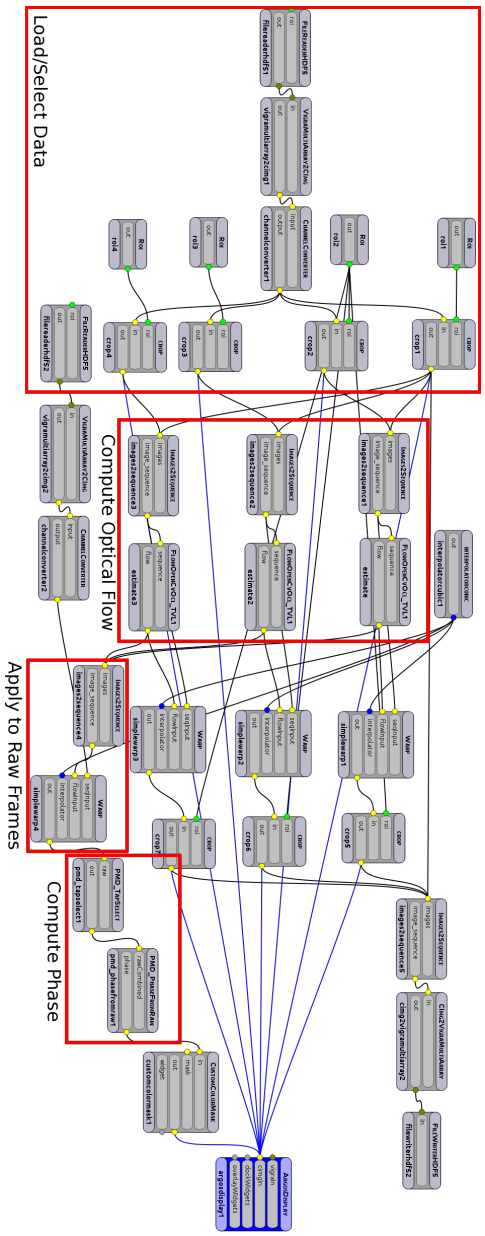


Figure A.7.: Visualization of the used charon workflow representing flow based motion compensation method by Lindner and Kolb. Boxes represent computing nodes, i.e. parts of the algorithm, lines visualize data flow. Important parts are grouped by red boxes. Best viewed using zoom in the PDF version of this document.

A.4. Rendered Rotor Sequence

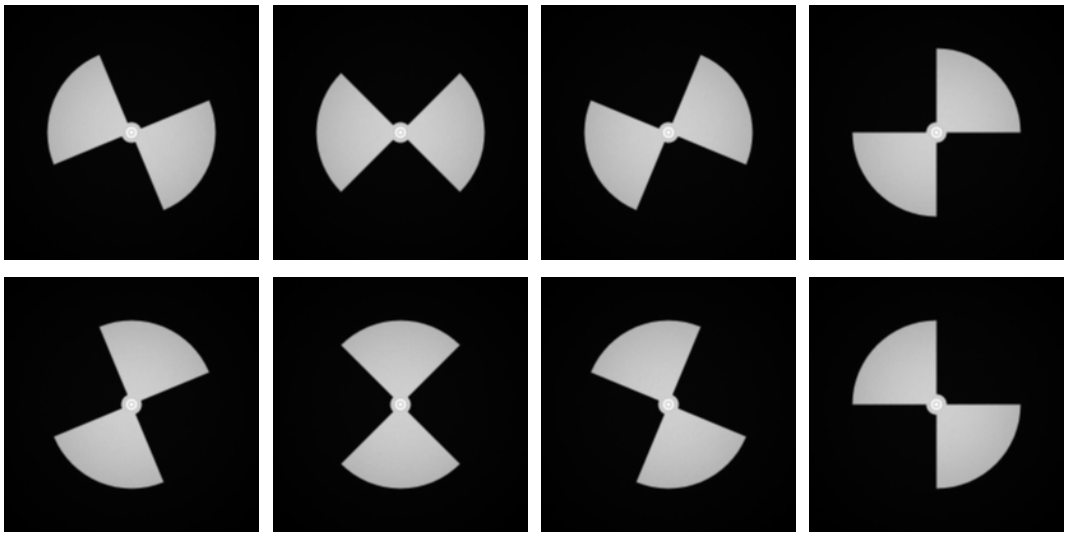


Figure A.8.: Rendered rotor sequence (without texture). **Rows:** Frame 0 (**top**) Frame 1 (**bottom**) **Columns:** Sub-frames 0 (**left**) to 3 (**right**). Rotor turns counter-clockwise with an angular velocity of $\omega = \frac{\pi}{2} \frac{\text{rad}}{\text{frame}} = \frac{\pi}{8} \frac{\text{rad}}{\text{sub-frame}}$.

A.5. Code and Applications

In this section, there is a collection of some of the source code files of the applications referred in the main part of this work.

A.5.1. PMD Freerun Capturing Software

This C-code file shows the OpenMP tuned command line capturing software able to process up to 50 FPS in freerun mode. To compile it, an OpenMP capable compiler (make sure it is enabled) and the PMD SDK (v2) are needed. The used preprocessor definition `PMDSDK2_MODULE_DIR` has to point to the directory containing the needed source and processing plugins from the SDK.

Listing A.1: rawCaptureFreerun.c

```
1  #include <string.h>
   #include <stdlib.h>
   #include <sys/time.h>
   #include <stdio.h>
5  #include <omp.h>
   #include <time.h>
   #include <assert.h>
   #include <pmdsdk2.h>

10 #define SOURCE_PLUGIN PMDSDK2_MODULE_DIR"/camcube3"
```

```
#define SOURCE_PARAM ""
#define PROC_PLUGIN PMDSDK2_MODULE_DIR"/camcubeproc"
#define PROC_PARAM ""

void pmdCheckCommand(int res, const char* errormsg, PMDHandle hnd) {
  if (res != PMD_OK) {
    const size_t _maxLen = 128u;
    char err[_maxLen];
    pmdGetLastError(hnd, err, _maxLen);
    pmdClose (hnd);
    fprintf(stderr, "%s\n", errormsg);
    fprintf(stderr, "some error occurred: %s\n", err);
    fprintf(stderr, "error code: %d", res);
    exit(EXIT_FAILURE);
  }
}

int errorcode = 0;

// record binary camera data as a sequence
int main (int argc, char *argv[]) {
  assert(sizeof(struct PMDDataDescription)==128u);
  if (argc < 4) {
    fprintf(stderr, "Expected three arguments: "
        "'frames to capture' "
        "'output filename' "
        "'integration time in us'"
        "['modulation frequency in MHz']\n");
    return EXIT_FAILURE;
  }

  PMDHandle hnd;
  unsigned int IntTime;
  unsigned int modFreq = 20000000;
  size_t frames=0;
  char oFileName[255];
  memset(oFileName,0,255u);

  // parse arguments
  {
    frames=atol(argv[1]);
    if (strlen(argv[2])>=255u) {
      fprintf(stderr,"filename too long");
      return EXIT_FAILURE;
    }
    strcpy(oFileName,argv[2]);
    IntTime = atoi(argv[3]);
    if (argc >= 5) {
      modFreq = atol(argv[4]) * 1000000;
    }
  }

  if (!frames) {
    fprintf(stderr, "zero frames requested, nothing to do\n");
    return EXIT_SUCCESS;
  }

  char* buffer;
  unsigned int bufSize;
  unsigned int datSize;
  const size_t bufFrames = 64u;
```

```
  struct PMDDataDescription dd[bufFrames];

75  int doWrite=1;
  if (strlen(oFileName) == 0 || strcmp(oFileName, "0") == 0) {
    printf("deactivated file output\n");
    doWrite=0;
  }
80
  {
    // establish connection
    printf("initializing device\n");
    pmdCheckCommand(
85    pmdOpen(&hnd,
          SOURCE_PLUGIN,SOURCE_PARAM,
          PROC_PLUGIN,PROC_PARAM),
      "Could not connect to device.", hnd);

90  // set integration time
    pmdCheckCommand(
      pmdSetIntegrationTime (hnd, 0, IntTime),
      "Could not set integration time", hnd);
    pmdCheckCommand(
95    pmdGetIntegrationTime (hnd, &IntTime, 0),
      "Could not check integration time.", hnd);
    printf("integration time: %u\n", IntTime);

    pmdCheckCommand(pmdUpdate(hnd),"Could transfer data.", hnd);
100
    pmdCheckCommand(
      pmdGetSourceDataDescription (hnd, dd),
      "Could get data description.", hnd);
    if (dd->subHeaderType != PMD_IMAGE_DATA) {
105   fprintf(stderr,"Source data is not an image!");
      pmdClose(hnd);
      exit(EXIT_FAILURE);
    }

110 // freerun mode to get max fps
    pmdCheckCommand(
      pmdSourceCommand(hnd, 0, 0, "SetTriggerMode Freerun"),
      "Could not set trigger mode freerun.", hnd);

115 // normal frequency mode
    pmdCheckCommand(
      pmdSourceCommand (hnd, 0, 0, "SetFrequencyMode Normal"),
      "Could not set normal frequency mode.", hnd);
    pmdCheckCommand(
120   pmdSetModulationFrequency(hnd,0,modFreq),
      "Could not set modulation frequency", hnd);
    pmdCheckCommand(
      pmdGetModulationFrequency(hnd,&modFreq,0),
      "Could not read modulation frequency", hnd);
125 printf("using modulation frequency: %d\n", modFreq);
  }

  datSize = dd->size;
  bufSize = datSize*bufFrames;
130 buffer = (char*) malloc(bufSize);

  size_t bufRead=0;
  size_t bufWrite=0;
```

```
135 #pragma omp parallel sections num_threads(2) \
      shared(errorcode,bufRead,bufWrite)
    {
      printf("working using %d threads\n", omp_get_num_threads());
#pragma omp section
140   {
        printf("start buffering (thread %d)...\n", omp_get_thread_num());
        struct timeval time1,time2;
        size_t i=0;
        gettimeofday(&time1,0);
145     pmdUpdate(hnd);

        for(i=0; i<frames;i++) {
          pmdUpdate(hnd);
          pmdGetSourceDataDescription(hnd,dd+(i%bufFrames));
150       pmdGetSourceData(
              hnd,buffer+(i%bufFrames)*datSize,datSize);
          bufWrite=i;
          #pragma omp flush(bufWrite, bufRead)
          if(doWrite && ((bufWrite-bufRead) > bufFrames)) {
155         errorcode = 1;
            #pragma omp flush(errorcode)
            fprintf(stderr,"Buffer overflow at frame %zu\n",i);
            break;
          }
160       if (errorcode) {
            printf("stopping capture (%zu frames buffered)\n",i);
            break;
          }
        }
165     gettimeofday(&time2,0);
        pmdClose(hnd);
        printf("buffering done, closing connection\n");
        long dTime=((long)time2.tv_sec-(long)time1.tv_sec)*1000000
            +((long)time2.tv_usec-(long)time1.tv_usec);
170     printf("capture took %f sec, FPS=%f\n",
             dTime/1.e6,1.e6*(frames-1)/dTime);
      }

#pragma omp section
175   if (doWrite) {
        printf("start writing   (thread %d)...\n", omp_get_thread_num());
        FILE* oFile = fopen(oFileName,"wb");
        size_t i, bOut;
        for (i=0; i < frames;i++) {
180       do {
            #pragma omp flush(bufWrite, bufRead, errorcode)
          }
          while((bufRead==bufWrite) && !errorcode);
          if (errorcode) {
185         printf("stopping writing (%zu frames written)\n",i);
            break;
          }
          bOut = fwrite(
              dd+(i%bufFrames),
190           sizeof(struct PMDDataDescription), 1, oFile);
          if (bOut != 1u) {
            fprintf(stderr, "failure writing data description\n");
            exit(EXIT_FAILURE);
          }
195       bOut = fwrite(
              buffer+(i%bufFrames)*datSize,
```

```
            sizeof(char), datSize, oFile);
        if (bOut != datSize) {
          fprintf(stderr, "failure writing image data\n");
          exit(EXIT_FAILURE);
        }
        bufRead = i;
      }
      fclose(oFile);
      printf("writing done\n");
    }
    else {
      printf("skip  writing   (thread %d)...\n", omp_get_thread_num());
    }
  }
  free(buffer);

  if (errorcode) {
    fprintf(stderr,"some error occured, write may be incomplete\n");
    return errorcode;
  }
  else {
    printf("%zu frames captured - success.\n",frames);
  }

  return EXIT_SUCCESS;
}
```

A.5.2. Unpacking the PMD Captured Binary Raw Data

This C-Code file shows the command line application used to unpack the binary data written by the capturing tool presented above. Selection of the interleaved tap data to write into a dedicated dataset is done in an elegant way using HDF5 hyperslabs. This application needs the PMD SDK (v2) and the HDF5 libraries. Tap hyperslab setup is at l. 167f, sub-frame pointers at l. 253ff, data writing into the tap datasets happens at l. 323ff.

Listing A.2: bin2h5.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <assert.h>
#include <pmdsdk2.h>
#include <hdf5.h>
#include <hdf5_hl.h>

#define PROC_PLUGIN PMDSDK2_MODULE_DIR"/camcubeproc"
#define PROC_PARAM ""

void pmdCheckCommand(int res, const char* errormsg, PMDHandle hnd) {
  if (res != PMD_OK) {
    const size_t _maxLen = 128u;
    char err[_maxLen];
    pmdGetLastError(hnd, err, _maxLen);
    pmdClose (hnd);
    fprintf(stderr, "%s\n", errormsg);
    fprintf(stderr, "some error occurred: %s\n", err);
    fprintf(stderr, "error code: %d\n", res);
    exit(EXIT_FAILURE);
```

```
  }
}

// convert binary camera data to raw and component data
int main (int argc, char *argv[]) {
  if (argc < 3) {
    printf("some arguments missing!\n");
    printf("Usage: %s rawFile outFile [skip]\n", argv[0]);
    exit(EXIT_FAILURE);
  }
  unsigned skip=0;
  if (argc > 3) {
    skip=atoi(argv[3]);
  }

  struct PMDDataDescription dd;
  assert(sizeof(struct PMDDataDescription)==128u);
  {
    // read data description
    printf("read data description...");
    fflush(stdout);
    FILE* ddin = fopen(argv[1], "rb");
    size_t r = fread(&dd, sizeof(struct PMDDataDescription), 1, ddin);
    fclose(ddin);
    if (r != 1u) {
      fprintf(stderr, "failure reading data description\n");
      exit(EXIT_FAILURE);
    }
    if (dd.subHeaderType != PMD_IMAGE_DATA) {
      fprintf(stderr,"invalid source data description\n");
      exit(EXIT_FAILURE);
    }
    printf("done\n");
  }

  size_t imageWidth = dd.img.numColumns;
  size_t imageHeight = dd.img.numRows;
  size_t source_size = dd.size;
  size_t imageSize = imageWidth*imageHeight;

  printf("image size:   %zux%zu\n", imageWidth, imageHeight);
  printf("header size: %8zu (%8.3f kB)\n",
       sizeof(struct PMDDataDescription),
       sizeof(struct PMDDataDescription)/1024.f);
  printf("source size: %8zu (%8.3f kB)\n", source_size, source_size/1024.f);
  printf("data/frame : %8zu (%8.3f kB)\n",
       source_size+sizeof(struct PMDDataDescription),
       (source_size+sizeof(struct PMDDataDescription))/1024.f);
  printf("exposures:     %6u\n",dd.img.numSubImages);
  if ((skip+1)*4 > dd.img.numSubImages) {
    fprintf(stderr, "skip value too large!\n");
    exit(EXIT_FAILURE);
  }

  char* source = (char*) malloc (source_size);

  // initialize PMD processing plugin
  printf("open processing plugin...");
  fflush(stdout);
  PMDHandle hnd;
  pmdCheckCommand(pmdOpenProcessingPlugin (&hnd, PROC_PLUGIN, PROC_PARAM),
    "Could not connect to processing plugin", hnd);
```

```
    printf("done\n");
85
    printf("load binary data...");
    fflush(stdout);
    FILE* dataB = fopen (argv[1], "rb");
    fseek(dataB, 0, SEEK_END);
90  printf("done\n");

    if (dd.img.numSubImages > 4) {
      printf("selected config:%2u/%2u\n", skip+1,dd.img.numSubImages/4);
    }
95
    long dataSize = ftell(dataB);
    printf("data size:%11ld (%8.3f MB)\n", dataSize, dataSize/1024.f/1024.f);
    size_t frames = dataSize/(source_size+128);
    assert(dataSize % (source_size+128) == 0);
100 printf("frames:%14zu\n", frames);
    fseek(dataB, 0, SEEK_SET);
    assert(ftell(dataB) == 0);

    // set up output file
105 hid_t h5file = H5Fcreate(
        argv[2],H5F_ACC_TRUNC,H5P_DEFAULT,H5P_DEFAULT);
    if (h5file <= 0) {
      fprintf(stderr, "Error creating output file.\n");
      exit(EXIT_FAILURE);
110 }

    hid_t gPmd=H5Gcreate(h5file,"pmd",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
    H5LTset_attribute_string(
          h5file, "pmd", "description",
115       "processed using the processing plugin from PMD SDK");
    hid_t gRaw=H5Gcreate(h5file,"raw",H5P_DEFAULT,H5P_DEFAULT,H5P_DEFAULT);
    H5LTset_attribute_string(
          h5file, "raw", "description",
          "data directly retrieved from cam raw data "
120       "containing the raw frames (taps and phase shifts)");

    H5LTset_attribute_string(
          h5file,"/","description", "PMD CamCube ToF data");
    H5LTset_attribute_int(
125       h5file,"/","integration time [us]",
          dd.img.integrationTime,dd.img.numSubImages/4);
    H5LTset_attribute_int(
          h5file,"/","modulation frequency [Hz]",
          dd.img.modulationFrequency,dd.img.numSubImages/4);
130 H5LTset_attribute_int(
          h5file,"/raw","selected exposure config",&skip,1);

    // set up data an memory spaces
    hsize_t mDimsD[3] = {1, imageHeight, imageWidth};
135 hsize_t dDimsD[3] = {frames, imageHeight, imageWidth};
    hsize_t dDimsT[2] = {frames, 2};
    hsize_t mDimsT[2] = {1, 2};
    hsize_t cDimsD[3] = {imageHeight,imageWidth,3};
    hsize_t cSlabD[3] = {imageHeight,imageWidth,1};
140 hsize_t cOffs0[3] = {0, 0, 0};
    hsize_t cOffs1[3] = {0, 0, 1};
    hsize_t cOffs2[3] = {0, 0, 2};
    hsize_t curOff[3] = {0, 0, 0};
    hsize_t taDims[3] = {imageHeight,imageWidth,2};
145 size_t iSize = imageSize*sizeof(float);
```

```
  float *vals = (float*) malloc (3*iSize);
  if (!vals) {
    fprintf(stderr, "could not allocate memory");
    fclose(dataB);
  }

  hid_t dSpace3 = H5Screate_simple(3, dDimsD, 0);
  hid_t mSpace3 = H5Screate_simple(3, mDimsD, 0);
  hid_t mSpaceX = H5Screate_simple(3, cDimsD, 0);
  hid_t mSpaceY = H5Screate_simple(3, cDimsD, 0);
  hid_t mSpaceZ = H5Screate_simple(3, cDimsD, 0);
  hid_t dSpacTr = H5Screate_simple(2, dDimsT, 0);
  hid_t mSpacTr = H5Screate_simple(2, mDimsT, 0);
  hid_t dSpacTf = H5Screate_simple(2, dDimsT, 0);
  hid_t mSpacTf = H5Screate_simple(2, mDimsT, 0);
  H5Sselect_hyperslab(mSpaceX,H5S_SELECT_SET,cOffs0,0,cSlabD,0);
  H5Sselect_hyperslab(mSpaceY,H5S_SELECT_SET,cOffs1,0,cSlabD,0);
  H5Sselect_hyperslab(mSpaceZ,H5S_SELECT_SET,cOffs2,0,cSlabD,0);

  hid_t mSpaceA = H5Screate_simple(3, taDims, 0);
  hid_t mSpaceB = H5Screate_simple(3, taDims, 0);
  H5Sselect_hyperslab(mSpaceA,H5S_SELECT_SET,cOffs0,0,cSlabD,0);
  H5Sselect_hyperslab(mSpaceB,H5S_SELECT_SET,cOffs1,0,cSlabD,0);

  hid_t prDC = H5Pcreate(H5P_DATASET_CREATE);
  hid_t prDA = H5Pcreate(H5P_DATASET_ACCESS);
  hid_t dsA = H5Dcreate(
        gPmd, "A", H5T_NATIVE_FLOAT,
        dSpace3, H5P_DEFAULT, prDC, prDA);
  H5LTset_attribute_string(
        gPmd, "A", "description",
        "amplitude of the returned light");
  hid_t dsB = H5Dcreate(
        gPmd, "B", H5T_NATIVE_FLOAT,
        dSpace3, H5P_DEFAULT, prDC, prDA);
  H5LTset_attribute_string(
        gPmd, "B", "description",
        "brightness or intensity image");
  hid_t dsR = H5Dcreate(
        gPmd, "R", H5T_NATIVE_FLOAT,
        dSpace3, H5P_DEFAULT, prDC, prDA);
  H5LTset_attribute_string(
        gPmd, "R", "description",
        "radial distance from camera");
  hid_t dsX = H5Dcreate(
        gPmd, "X", H5T_NATIVE_FLOAT,
        dSpace3, H5P_DEFAULT, prDC, prDA);
  H5LTset_attribute_string(
        gPmd, "X", "description",
        "X-value of pixel's 3D point coordinate");
  hid_t dsY = H5Dcreate(
        gPmd, "Y", H5T_NATIVE_FLOAT,
        dSpace3, H5P_DEFAULT, prDC, prDA);
  H5LTset_attribute_string(
        gPmd, "Y", "description",
        "Y-value of pixel's 3D point coordinate");
  hid_t dsZ = H5Dcreate(
        gPmd, "Z", H5T_NATIVE_FLOAT,
        dSpace3, H5P_DEFAULT, prDC, prDA);
  H5LTset_attribute_string(
        gPmd, "Z", "description",
        "Z-value of pixel's 3D point coordinate, "
```

```
      "i.e. distance parallel to view direction of camera");

// dataset for timestamps
hid_t dTstpR = H5Dcreate(
      gRaw,"timestamps", H5T_NATIVE_UINT32,
      dSpacTr, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
H5LTset_attribute_string(
      gRaw,"timestamps","description",
      "unix time (seconds after epoch) and microseconds");
hid_t dTstpF = H5Dcreate(
      gPmd, "timestamps", H5T_NATIVE_UINT64,
      dSpacTf, H5P_DEFAULT, H5P_DEFAULT, H5P_DEFAULT);
H5LTset_attribute_string(
      gPmd,"timestamps","description",
      "microseconds after first exposure and between exposures");

// datasets for raw data
hid_t dsA000 = H5Dcreate(
      gRaw, "A0", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsA090 = H5Dcreate(
      gRaw, "A90", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsA180 = H5Dcreate(
      gRaw, "A180", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsA270 = H5Dcreate(
      gRaw, "A270", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsB000 = H5Dcreate(
      gRaw, "B0", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsB090 = H5Dcreate(
      gRaw, "B90", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsB180 = H5Dcreate(
      gRaw, "B180", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);
hid_t dsB270 = H5Dcreate(
      gRaw, "B270", H5T_NATIVE_UINT16,
      dSpace3, H5P_DEFAULT, prDC, prDA);

// pointers to subframe raw data:
// interleaved tapA/B uint16 big endian values
// with a preceeding header of 0x200 bytes
void* const skipSrc = source+(0x200+imageSize*4)*skip*4;
void* const sframe1 = skipSrc+0x200;
void* const sframe2 = sframe1+0x200+imageSize*4;
void* const sframe3 = sframe2+0x200+imageSize*4;
void* const sframe4 = sframe3+0x200+imageSize*4;

// process source binary data
size_t br;
int64_t fTimeStmpHi=dd.img.timeStampHi;
int64_t fTimeStmpLo=dd.img.timeStampLo;
int64_t curStampAndDiff[3]={0,0,0};
size_t ii;
for(ii=0;ii<frames;ii++) {
  br = fread(&dd, sizeof(struct PMDDataDescription), 1, dataB);
  if (br != 1u) {
    fprintf(stderr, "read description failed at frame %zu\n", ii);
    break;
```

```
    }
    br = fread(source, sizeof(char), source_size, dataB);
    if (br != source_size) {
      fprintf(stderr, "read image data failed at frame %zu\n", ii);
      break;
    }

    // select slice to write
    curOff[0] = ii;
    H5Sselect_hyperslab(dSpace3,H5S_SELECT_SET,curOff,0,mDimsD,0);
    H5Sselect_hyperslab(dSpacTr,H5S_SELECT_SET,curOff,0,mDimsT,0);
    H5Sselect_hyperslab(dSpacTf,H5S_SELECT_SET,curOff,0,mDimsT,0);

    // write timestamp
    H5Dwrite(
      dTstpR,H5T_NATIVE_UINT32,mSpacTr,dSpacTr,
      H5P_DEFAULT,&(dd.img.timeStampHi));
    curStampAndDiff[0]=((int64_t)dd.img.timeStampHi-fTimeStmpHi)*1e6
        +((int64_t)dd.img.timeStampLo-fTimeStmpLo);
    curStampAndDiff[1]=curStampAndDiff[0]-curStampAndDiff[2];
    curStampAndDiff[2]=curStampAndDiff[0];
    H5Dwrite(
      dTstpF,H5T_NATIVE_INT64,mSpacTf,dSpacTf,
      H5P_DEFAULT,&curStampAndDiff);

    // get Intensities
    pmdCheckCommand(pmdCalcIntensities(hnd,vals,iSize,dd,source),
      "Could not get intensities", hnd);
    H5Dwrite(dsB,H5T_NATIVE_FLOAT,mSpace3,dSpace3,H5P_DEFAULT,vals);

    // get Amplitude
    pmdCheckCommand(pmdCalcAmplitudes(hnd,vals,iSize,dd,source),
      "Could get amplitude: %s\n", hnd);
    H5Dwrite(
      dsA,H5T_NATIVE_FLOAT,mSpace3,dSpace3,H5P_DEFAULT,vals);

    // get Distances
    pmdCheckCommand(pmdCalcDistances(hnd,vals,iSize,dd,source),
      "Could get distances: %s\n", hnd);
    H5Dwrite(
      dsR,H5T_NATIVE_FLOAT,mSpace3,dSpace3,H5P_DEFAULT,vals);

    // get 3D coordinates
    pmdCheckCommand(pmdCalc3DCoordinates(hnd,vals,3*iSize,dd,source),
      "Could get 3D coordinates: %s\n", hnd);
    H5Dwrite(
      dsX,H5T_NATIVE_FLOAT,mSpaceX,dSpace3,H5P_DEFAULT,vals);
    H5Dwrite(
      dsY,H5T_NATIVE_FLOAT,mSpaceY,dSpace3,H5P_DEFAULT,vals);
    H5Dwrite(
      dsZ,H5T_NATIVE_FLOAT,mSpaceZ,dSpace3,H5P_DEFAULT,vals);

    // write raw data
    H5Dwrite(dsA000,H5T_STD_U16BE,mSpaceA,dSpace3,H5P_DEFAULT,sframe1);
    H5Dwrite(dsB180,H5T_STD_U16BE,mSpaceB,dSpace3,H5P_DEFAULT,sframe1);
    H5Dwrite(dsA090,H5T_STD_U16BE,mSpaceA,dSpace3,H5P_DEFAULT,sframe2);
    H5Dwrite(dsB270,H5T_STD_U16BE,mSpaceB,dSpace3,H5P_DEFAULT,sframe2);
    H5Dwrite(dsA180,H5T_STD_U16BE,mSpaceA,dSpace3,H5P_DEFAULT,sframe3);
    H5Dwrite(dsB000,H5T_STD_U16BE,mSpaceB,dSpace3,H5P_DEFAULT,sframe3);
    H5Dwrite(dsA270,H5T_STD_U16BE,mSpaceA,dSpace3,H5P_DEFAULT,sframe4);
    H5Dwrite(dsB090,H5T_STD_U16BE,mSpaceB,dSpace3,H5P_DEFAULT,sframe4);
  }
```

```
  // cleanup
  H5Dclose(dsA);      H5Dclose(dsB);      H5Dclose(dsR);
  H5Dclose(dsX);      H5Dclose(dsY);      H5Dclose(dsZ);
  H5Dclose(dTstpR);
  H5Dclose(dsA000);   H5Dclose(dsA090);   H5Dclose(dsA180);   H5Dclose(dsA270);
  H5Dclose(dsB000);   H5Dclose(dsB090);   H5Dclose(dsB180);   H5Dclose(dsB270);
  H5Sclose(mSpace3);  H5Sclose(mSpaceX);  H5Sclose(mSpaceY);  H5Sclose(mSpaceZ);
  H5Sclose(mSpaceA);  H5Sclose(mSpaceB);  H5Sclose(mSpacTr);  H5Sclose(dSpacTr);
  H5Pclose(prDC);     H5Pclose(prDA);
  H5Sclose(dSpace3);
  H5Gclose(gPmd);     H5Gclose(gRaw);
  H5Fclose(h5file);
  fclose(dataB);
  free(source);       free(vals);
  pmdClose(hnd);

  return EXIT_SUCCESS;
}
```

A.5.3. Estimation of Intrinsic Camera Parameters

This python script has been used to determine the camera parameters used by the PMD SDK for computation of 3D coordinates. It depends on the python packages `numpy` and `h5py` and runs with `python2` and `python3`.

Listing A.3: getCamParams.py

```
#!/usr/bin/env python
"""
retrieve camera parameters from pmd dataset
===========================================

By reverse engineering of the radial distance and 3D coordinates,
this script tries to recover the camera parameters cx, cy, fx, fy
"""

from __future__ import print_function, division
import argparse
import textwrap

import h5py as h5
import numpy as np
from math import atan, pi

def parse_args():
  """parse command line args, use -h for help"""
  parser = argparse.ArgumentParser(
    description=textwrap.dedent(__doc__),
    formatter_class=argparse.ArgumentDefaultsHelpFormatter)
  parser.add_argument(
    'inputFile', help='input hdf5 file')
  parser.add_argument(
    '-i', '--inputGroup', default='/pmd',
    help='input group containing radial distance and 3D coordinate datasets')
  return parser.parse_args()

def estimate_params(rr, xx, yy, zz):
  sh = rr.shape
  xx /= zz
```

```
  yy /= zz
  fx = 1. / np.mean(np.diff(xx, axis=2))
  fy = 1. / np.mean(np.diff(yy, axis=1))
  xx *= fx
  yy *= fy
  xc, yc = np.meshgrid(range(sh[1]), range(sh[2]))
  xc = np.tile(xc, (sh[0], 1, 1))
  yc = np.tile(yc, (sh[0], 1, 1))
  cx = np.mean(xc-xx)
  cy = np.mean(yc-yy)
  return fx, fy, cx, cy

def load_data(in_file, in_group):
  with h5.File(in_file, 'r') as h5f:
    assert in_group in h5f, 'input group not found in %s' % in_file
    h5g = h5f[in_group]
    assert 'R' in h5g, 'input group has to contain a R dataset'
    assert 'X' in h5g, 'input group has to contain a X dataset'
    assert 'Y' in h5g, 'input group has to contain a Y dataset'
    assert 'Z' in h5g, 'input group has to contain a Z dataset'
    return h5g['R'][:], h5g['X'][:], h5g['Y'][:], h5g['Z'][:]

def main():
  args = parse_args()
  rr, xx, yy, zz = load_data(args.inputFile, args.inputGroup)
  fx, fy, cx, cy = estimate_params(rr, xx, yy, zz)
  print('fx =', fx, 'fy =', fy, 'cx =', cx, 'cy =', cy)
  print(
    'fov: ',
    atan(0.5*(rr.shape[2]-1)/fx)*360/pi,
    atan(0.5*(rr.shape[1]-1)/fy) * 360/pi
  )

if __name__ == '__main__':
  main()
```

Appendix B.

Lists and Directories

B.1. Used Abbreviations and Acronyms

AU arbitrary units

BCCE brightness constancy constraint equation

BID burst internal detection

BM block matching

CCD charge-coupled device

CMOS complementary metal–oxide–semiconductor

CPU central processing unit of a computer

CT computer tomography

CUDA compute unified device architecture,
parallel computing platform created by NVIDIA

CWIM continuous wave intensity modulation

DU digital units

DSNU dark signal non-uniformity

FOV field of view, opening angle of the cone where objects are visible to the camera

FPS frames per second

GPU graphics processing unit

HDF5 hierarchical data format

HSV color space with three channels (Hue, Saturation, Value)

IR infrared

OFC optical flow constraint

PCA principal component analysis

PMD photon mixing device,
PMDTechnologies is the manufacturer of the used *CamCube* devices

PRNU photo response non-uniformity

PyrLK pyramid variant of the method by Lucas and Kanade [LK81]

RFMC range flow motion constraint, cf. eq. (3.8)

RGB color space with three channels (Red, Green, Blue)

RGBD multi-modal color (RGB) and depth data

ROI region of interest

SDK software development kit

ToF Time-of-Flight

TV-L^1 total variation regularizer with L^1-norm data term

USB universal serial bus, a standard for connections e.g. between computers and peripherals

VGA video graphics array, a graphics standard with a resolution of 640×480 pixels

B.2. List of Figures

B.3. List of Tables

B.4. List of Listings

B.5. References

[Bak+07] Simon Baker et al. "A Database and Evaluation Methodology for Optical Flow." In: *ICCV*. IEEE, 2007, pp. 1–8. URL: `http://vision.middlebury.edu/flow`.

[Ber+00] Marcelo Bertalmio et al. "Image inpainting." In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 417–424.

[Bes88] Paul J. Besl. "Active, optical range imaging sensors." In: *Machine vision and applications* 1.2 (1988), pp. 127–152. ISSN: 0932-8092.

[BF82] Stephen T Barnard and Martin A Fischler. "Computational stereo." In: *ACM Computing Surveys (CSUR)* 14.4 (1982), pp. 553–572.

[Bra+00] Gary Bradski et al. "The OpenCV library." In: *Doctor Dobbs Journal* 25.11 (2000), pp. 120–126. URL: `http://opencv.org`.

[Bro+04] Thomas Brox et al. "High Accuracy Optical Flow Estimation Based on a Theory for Warping." In: *ECCV (4)*. Ed. by Tomás Pajdla and Jiri Matas. Vol. 3024. Lecture Notes in Computer Science. Springer, 2004, pp. 25–36. ISBN: 3-540-21981-1.

[BS70] Williard S Boyle and George E Smith. "Charge coupled semiconductor devices." In: *Bell System Technical Journal* 49.4 (1970), pp. 587–593.

[Bur] Nicolas Burrus. *Kinect Calibration - Calibrating the depth and color camera*. URL: `http://nicolas.burrus.name/index.php/Research/KinectCalibration`.

[DW68] Rudolph H Dyck and Gene P Weckler. "Integrated arrays of silicon photodetectors for image sensing." In: *IEEE Transactions on Electron Devices* 15.4 (1968), pp. 196–201.

[Erz11] Michael Erz. "Charakterisierung von Laufzeit-Kamera-Systemen für Lumineszenz-Lebensdauer-Messungen." Dissertation. IWR, Fakultät für Physik und Astronomie, Universität Heidelberg, 2011.

[Far00] Gunnar Farnebäck. "Fast and Accurate Motion Estimation Using Orientation Tensors and Parametric Motion Models." In: *ICPR*. 2000, pp. 1135–1139.

[Fot15] *Fotonic E-Series Datasheet*. Version 2015:1. Fotonic. 2015. URL: `http://www.fotonic.com/wp-content/uploads/2015/10/Datasheet_Fotonic_E-series_2015_1-2.pdf` (visited on 08/15/2016).

[GFG11] Jens-Malte Gottfried, Janis Fehr, and Christoph S. Garbe. "Computing Range Flow from Multi-modal Kinect Data." In: *ISVC (1)*. Ed. by George Bebis et al. Vol. 6938. Lecture Notes in Computer Science. Springer, 2011, pp. 758–767. ISBN: 978-3-642-24027-0. DOI: 10.1007/978-3-642-24028-7_70.

[GG84] Stuart Geman and Donald Geman. "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images." In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 6 (1984), pp. 721–741.

[GK12] Jens-Malte Gottfried and Daniel Kondermann. "Charon Suite Software Framework." In: *IPOL 2012 Meeting on Image Processing Libraries*. 2012. URL: http://www.ipol.im/event/2012_imlib/.

[GMK] Jens-Malte Gottfried, Stephan Meister, and Daniel Kondermann. *Charon modular algorithm framework*. URL: http://charon-suite.sf.net.

[Got+14] Jens-Malte Gottfried et al. "Time of Flight Motion Compensation Revisited." In: *IEEE International Conference on Image Processing 2014 (ICIP 2014)*. Paris, France, Oct. 2014.

[GYB04] Salih Burak Göktürk, Hakan Yalcin, and Cyrus Bamji. "A time-of-flight depth sensor-system description, issues and solutions." In: *Computer Vision and Pattern Recognition Workshop (CVPRW'04)*. Vol. 3. IEEE. 2004, p. 35.

[Han+12] Miles Hansard et al. *Time of Flight Cameras: Principles, Methods, and Applications*. SpringerBriefs in Computer Science. Springer, Nov. 2012.

[HE10] Stephan Hussmann and Torsten Edeler. "Pseudo-four-phase-shift algorithm for performance enhancement of 3D-ToF vision systems." In: *Instrumentation and Measurement, IEEE Transactions on* 59.5 (2010), pp. 1175–1181.

[HHE11] Stephan Hussmann, Alexander Hermanski, and Torsten Edeler. "Real-Time Motion Artifact Suppression in TOF Camera Systems." In: *IEEE Transactions on Instrumentation and Measurement* 60.5 (2011), pp. 1682–1690.

[HLK13] Thomas Hoegg, Damien Lefloch, and Andreas Kolb. "Real-Time Motion Artifact Compensation for PMD-ToF Images." In: *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications - Dagstuhl 2012 Seminar on Time-of-Flight Imaging and GCPR 2013 Workshop on Imaging New Modalities*. Ed. by Marcin Grzegorzek et al. Vol. 8200. Lecture Notes in Computer Science. Springer, 2013, pp. 273–288. ISBN: 978-3-642-44963-5. DOI: 10.1007/978-3-642-44964-2_13. URL: http://dx.doi.org/10.1007/978-3-642-44964-2_13.

[HRF13] Evan Herbst, Xiaofeng Ren, and Dieter Fox. "Rgb-d flow: Dense 3-d motion estimation using color and depth." In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 2276–2282.

[HS81] Berthold K. P. Horn and Brian G. Schunck. "Determining Optical Flow." In: *Artif. Intell.* 17.1-3 (1981), pp. 185–203.

[Jäh05] Bernd Jähne. *Digitale Bildverarbeitung*. 6th ed. Heidelberg: Springer Verlag, 2005. ISBN: 3-540-24999-0.

[Jai+15] Mariano Jaimez et al. "A primal-dual framework for real-time dense RGB-D scene flow." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 98–104.

[JPM14] David Jimenez, Daniel Pizarro, and Manuel Mazo. "Single frame correction of motion artifacts in PMD-based time of flight cameras." In: *Image and Vision Computing* 32.12 (2014), pp. 1127–1143.

[Lee+12] Seungkyu Lee et al. "Motion blur-free time-of-flight range sensor." In: *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics. 2012, 82980U–82980U.

[LHK13] D. Lefloch, T. Hoegg, and A. Kolb. "Real-time motion artifacts compensation of ToF sensors data on GPU." In: *Proc. SPIE, Three-Dimensional Imaging, Visualization, and Display*. Vol. 8738. SPIE, 2013.

[LK09] Marvin Lindner and Andreas Kolb. "Compensation of Motion Artifacts for Time-of-Flight Cameras." In: *Dynamic 3D Imaging*. Ed. by Andreas Kolb and Reinhard Koch. Vol. 5742. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 16–27. ISBN: 978-3-642-03777-1.

[LK81] Bruce D. Lucas and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision." In: *IJCAI*. Ed. by Patrick J. Hayes. William Kaufmann, 1981, pp. 674–679.

[Lot+07] Oliver Lottner et al. "Movement artefacts in range images of time-of-flight cameras." In: *International Symposium on Signals, Circuits and Systems, 2007. ISSCS 2007*. Vol. 1. IEEE. 2007, pp. 1–4.

[Mar10] Hector Martin. *OpenKinect project - Drivers and libraries for the Xbox Kinect device*. 2010. URL: `http://openkinect.org`.

[MKS13] *Kinect for Windows SDK*. Version 1.8. Microsoft. 2013. URL: `https://www.microsoft.com/en-us/download/details.aspx?id=40278` (visited on 01/11/2016).

[MNK13] Stephan Meister, Rahul Nair, and Daniel Kondermann. "Simulation of Time-of-Flight Sensors using Global Illumination." In: *Vision, Modeling & Visualization*. The Eurographics Association. 2013, pp. 33–40.

[Nob68] Peter JW Noble. "Self-scanned silicon image detector arrays." In: *IEEE Transactions on Electron Devices* 15.4 (1968), pp. 202–209.

[OK93] Masatoshi Okutomi and Takeo Kanade. "A Multiple-Baseline Stereo." In: *IEEE Trans. Pattern Anal. Mach. Intell.* 15.4 (1993), pp. 353–363. DOI: 10.1109/34.206955. URL: http://dx.doi.org/10.1109/34.206955.

[Pap+06] Nils Papenberg et al. "Highly Accurate Optic Flow Computation with Theoretically Justified Warping." In: *International Journal of Computer Vision* 67.2 (2006), pp. 141–158.

[PMD16] *Sensing the world the way we do – the world's first Project Tango Smartphone. Lenovo and Google redefine the mobile device category using pmd's unique 3D technology.* Lenovo Tech World '16. Press Release. Siegen/Germany: PMDTechnologies AG, June 2016. URL: http://pmdtec.com/html/pdf/press_release/PR_20160609_pmd_part_of_Lenovo_Tango_phone.pdf (visited on 09/10/2016).

[Sch+95] Rudolf Schwarte et al. "A new active 3D-Vision system based on rf-modulation interferometry of incoherent light." In: *Photonics East-Intelligent Systems and Advanced Manufacturing, Proceedings of the SPIE* 2588 (1995).

[Sch00] Hanno Scharr. "Optimale Operatoren in der digitalen Bildverarbeitung." PhD thesis. Universität Heidelberg, 2000.

[Sch11] Mirko Schmidt. "Analysis, Modeling and Dynamic Optimization of 3D Time-of-Flight Imaging Systems." Dissertation. IWR, Fakultät für Physik und Astronomie, Uni Heidelberg, 2011.

[SJB02] Hagen Spies, Bernd Jähne, and John L. Barron. "Range Flow Estimation." In: *Computer Vision and Image Understanding* 85.3 (2002), pp. 209–231.

[Spi+95] Thomas Spirig et al. "The lock-in CCD-two-dimensional synchronous detection of light." In: *IEEE Journal of quantum electronics* 31.9 (1995), pp. 1705–1708.

[SRB10] Deqing Sun, Stefan Roth, and Michael J. Black. "Secrets of optical flow estimation and their principles." In: *CVPR*. IEEE, 2010, pp. 2432–2439.

[SS06] Agustín Salgado and Javier Sánchez. "A Temporal Regularizer for Large Optical Flow Estimation." In: *Proceedings of the International Conference on Image Processing, ICIP 2006, October 8-11, Atlanta, Georgia, USA*. IEEE, 2006, pp. 1233–1236.

[Sun+08] Deqing Sun et al. "Learning Optical Flow." In: *ECCV (3)*. Ed. by David A. Forsyth, Philip H. S. Torr, and Andrew Zisserman. Vol. 5304. Lecture Notes in Computer Science. Springer, 2008, pp. 83–97. ISBN: 978-3-540-88689-1.

[Ved+99] Sundar Vedula et al. "Three-Dimensional Scene Flow." In: *ICCV*. 1999, pp. 722–729.

[Zha99] Zhengyou Zhang. "Flexible camera calibration by viewing a plane from unknown orientations." In: *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Vol. 1. Ieee. 1999, pp. 666–673.

[ZPB07] Christopher Zach, Thomas Pock, and Horst Bischof. "A Duality Based Approach for Realtime TV-L^1 Optical Flow." In: *DAGM-Symposium*. Ed. by Fred A. Hamprecht, Christoph Schnörr, and Bernd Jähne. Vol. 4713. Lecture Notes in Computer Science. Springer, 2007, pp. 214–223. ISBN: 978-3-540-74933-2.