Real-Time Motion Artifact Compensation for PMD-ToF Images

Thomas Hoegg[1], Damien Lefloch[2], and Andreas Kolb[2]

[1] Christ-Elektronik GmbH, Alpenstr. 34, 87700 Memmingen, Germany
thoegg@christ-elektronik.de
[2] University of Siegen, Hoelderlinstr. 3, 57076 Siegen, Germany
damien.lefloch@uni-siegen.de,andreas.kolb@uni-siegen.de

Abstract. Time-of-Flight (ToF) cameras gained a lot of scientific attention and became a vivid field of research in the last years. A still remaining problem of ToF cameras are motion artifacts in dynamic scenes. This paper presents a new preprocessing method for a fast motion artifact compensation. We introduce a flow like algorithm that supports motion estimation, search field reduction and motion field optimization. The main focus lies on real-time processing capabilities. The approach is extensively tested and compared against other motion compensation techniques. For the evaluation, we use quantitative (ground-truth data, statistic error comparison) and qualitative (real environments, visual comparison) test methods. We show, that our proposed algorithm runs in real-time within a GPU based processing hardware (using NVIDIA Cuda) and corrects motion artifacts in a reliable way.

Keywords: PMD, ToF, Motion Artifacts

1 Introduction

Time-of-Flight (ToF) sensors as the PMD camera [1] offer an elegant way to measure depth data. They become more and more important for the computer vision and graphics domain and also for industrial applications [2]. Having advantages such as high performance and no mechanical overhead compared to e.g. laser scanners, they also posses many problems, especially in accuracy and noise behavior. Most of these errors can be corrected well by applying good calibration models [3] and pre-filtering (e.g. low-pass filtering). However, artifacts arising from dynamic scenes are still not resolved satisfactorily. Moving objects in scenes result in a blur effect (motion artifacts) in acquired depth images. A fast movement leads to strong artifacts, related to the sensor's working principle which is based on the sequential acquisition of four so-called phase images in order to generate a depth map (see also Sec. 3). Artifacts occur in areas where corresponding phase image values do not align to each other, resulting in a incorrect distance calculation.

In this paper we propose a new algorithm to perform a fast real-time motion

compensation with high frame rates (above 50 FPS). We focus on high flexibility to allow the algorithm to be either computed parallelized on a GPU using CUDA [4] or to simply port it to small devices like an FPGA preprocessing platform. To fulfill these requirements, a linear movement with constant motion between the four consecutive phase images is assumed. Hence, the algorithm still allows an arbitrary degree of freedom assuming this linear behavior for each individual pixel (see Sec. 5). Invalid pixels are replaced by corresponding values of the spatial neighborhood. This leads to simpler and faster processing compared to standard methods shown in Sec. 2. For the evaluation, a PMD CamCube 3.0 with a resolution of 200×200 pixel is used. The big advantages of the proposed method are the possibility of an automatic motion detection, a search direction restriction, the repeatability of results in different applications and also the system performance.

The remainder of this paper is organized as follows. Sec. 2 discusses the related work. In Sec. 3 we describe the working principle of PMD cameras. Sec. 4 introduces our proposed algorithm for the motion compensation. In Sec. 5 we show our results. Sec. 6 concludes this paper.

2 Related Work

In the last years, several methods have been proposed to detect and compensate motion artifacts.

Hussmann et al. [5] introduce a motion compensation for linear object motion on a conveyor belt. Areas of motion artifacts are identified using phase image differences. These areas are binarized for each individual difference image using a threshold. The length of motion is determined by processing each line of the binary images and counting the lines with white pixels. Once knowing the length, every phase image is moved accordingly before the distances are calculated. The algorithm is implemented exemplary on an FPGA platform, but it is restricted to a linear motion in a range between $90 - 100cm$ due to the small object size and the camera field of view.

Schmidt [6] proposes a method handling motion artifacts as disturbances in the raw data. Motion artifacts are calculated for each phase image using a temporal derivative. High temporal derivatives of the raw data are then replaced by previously valid values. An advantage compared to Hussmann et al. is the arbitrary degree of freedom. Lee et al. [7] propose a similar approach where they detect motion artifacts by temporal-spatial coherence of neighboring pixels directly on the hardware level.

Another method was proposed by Lindner et al. [8]. This method computes a dense optical flow to compensate spatial shifts between subsequent phase images (three flow calculations). Lefloch et al. [9] proposed a method improving this approach. Necessary computation steps can be reduced to two flow calcu-

lations. The missing step is replaced by a polynomial approximation. One big disadvantage is the system performance. The optical flow computation is a very time consuming task and thus is a heavy burden for real-time processing, if further pocessing tasks need to be performed.

Our proposed method uses particular parts from Lindner and Lefloch [8, 9] (flow field) and Hussmann [5] (binarization of the motion area). The algorithm restricts the motion to blurred areas only and optimizes the flow field detection.

3 The Time-of-Flight Principle

The following section gives a brief introduction to the functionality of ToF cameras serving as a basis for the method proposed in this paper.

An intensity modulated, incoherent infrared (IR) light is emitted with a modulation frequency f using the cameras' illumination units to determine the phase shift between outgoing and incoming optical signals. Each camera pixel correlates the incoming signal $s(t)$ with the reference signal $r(t)$ to estimate the correlation function. This process is repeated four times for every pixel with different internal phase shift $\tau_i = i \cdot \pi$ in order to sample the correlation function between s and r.

The PMD camera for instance is a two-tap sensor. It allows the acquistion of two corresponding values for the same pixel at the same time, represented in the camera as two phases P_{A_i}, P_{B_i}. In theory, if there is no motion and other influences, the phases are complementary, i.e. measuring phase values at two positions with 180° difference: $P_{B_i} = P_{A_{(i+2) \mod 4}}$. Internally, the phase image is represented by the difference $P_i = P_{A_i} - P_{B_i}$ in order to compensate for hardware inaccuracies. Using the four phase images P_i, the phase shift ϕ, the amplitude A and the intensity I can be calculated:

$$\phi = \arctan 2(P_0 - P_2, P_3 - P_1) \tag{1}$$

$$I = \frac{P_0 + P_1 + P_2 + P_3}{4} \tag{2}$$

$$A = \frac{1}{2} \cdot \sqrt{(P_3 - P_1)^2 + (P_0 - P_2)^2}. \tag{3}$$

The resulting distance D is then received using the angular modulation frequency $\omega = 2\pi f$ and the speed of light $c \approx 3 \cdot 10^8 ms^{-1}$:

$$D = \frac{c}{2\omega}\phi. \tag{4}$$

4 A Method for Fast Linear Motion Compensation

In this section we start with an analysis of the origin of motion artifacts and continue with a detailed description of our proposed method.

4.1 Problem Analysis

ToF-cameras as the PMD camera have the advantage to be able to acquire full distance-/depth-images of the whole scene at a time. This is done using a sequence of four phase images, as described in Sec. 3 and shown in Fig. 1.

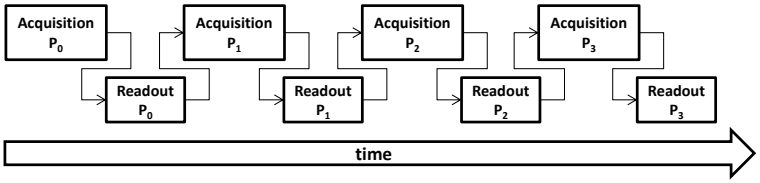


Fig. 1: Schematic view of the acquisition process of a PMD frame using four phase images.

One full phase acquisition is split in two parts: acquisition and readout. The acquisition time is equal to the integration time set, the readout time of actual PMD cameras is stated as about $3.5ms$. Ideally all four phase images would simultaneously be recorded. In reality the acquisition is sequentially done (see Fig. 1). Motion artifacts typically arise in areas of unmatching raw phase values due to motion (see. Fig. 2). It mainly occurs at object boundaries and in regions of inhomogeneous reflection. This effect becomes more extensive the faster an object moves, the closer the object is to the camera and the higher the scene is exposed (high integration times) [8].

Fig. 2 shows the default demodulation of a car (left images), moving from right to the left and of a moving hand (right images). In both scenes, the blurred areas are marked red. It can be seen that especially these areas contain many motion artifacts.

Blurred areas in depth maps lead to incorrect distance computations. The goal of motion compensation approaches is the elimination of these areas to minimize errors. The motion during a single acquisition is not considered here and is nearly negligible for small integration times ($< 1ms$).

4.2 The Motion Compensation Approach

The proposed method works on a per pixel basis allowing arbitrary motion directions. It is divided into several steps, starting with a phase normalization. The normalization is done to compensate the sensor's pixel gains and to equalize the image illumination. This is necessary due to the block-matching like working principle of the approach and to obtain comparable raw values. In a second step the area of motion is estimated to improve the processing time. The motion direction is then determined with a correspondence search in the spatial neighborhood. Once knowing this kind of flow field, the raw values can be corrected. The processing pipeline can be seen in Fig. 3 and will be explained in the following sections.

Fig. 2: Top: Demodulation of the car's phase image sequence and a corresponding closeup. Bottom: A moving hand scene and a corresponding closeup. Left: Motion areas are marked red. Right: The closeups of the red marked motion areas.

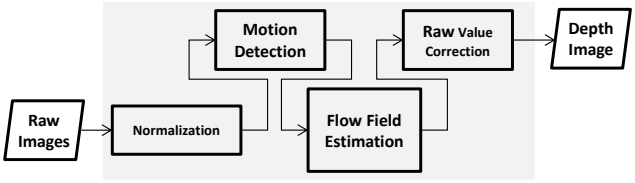


Fig. 3: The motion detection and processing pipeline used for our algorithm.

Phase Normalization According to the behavior and design of PMD cameras, there are several aspects for the pixel correspondence search, which have to be taken care of. One point is the radial light attenuation. Images become darker from the center to the border. Another aspect is the difference in pixel gains, which has to be individually corrected for each sensor and every pixel.

To compensate these two problems, a pixel adjustment is performed using the method proposed by Lindner et al. [8] by applying a pixel-wise intensity correction function

$$f_{P_A}(P_{A_i}) = \tilde{P}_{A_i},\ f_{P_B}(P_{B_i}) = \tilde{P}_{B_i} \text{ with } i = 0 \dots 3 \tag{5}$$

to minimize

$$\sum_{i=0}^{3} (\tilde{P}_{A_i} + \tilde{P}_{B_i}) = h_{\text{ref}}. \tag{6}$$

The brightest pixel in a homogeneous surface is taken and used as reference intensity, the fitting functions are assumed to be logarithmic as $f_X(X_i) = a\sqrt{X_i + b} + cX_i + d$.

Applying these corrections improve the search as shown by Lindner et al.

Motion Detection Since the motion estimation is a computation intensive task, an important preprocessing step is to detect areas of apparent motions first. Motion can be detected using the changes in the total per-pixel intensity for the subsequent phase images, i.e.

$$P_i^+ = P_{A_i} + P_{B_i} \tag{7}$$

$$M = \sum_{i=1}^{3} \left| P_i^+ - P_0^+ \right| \tag{8}$$

In a next step, the estimated motion image M is binarized

$$B = M > \theta \tag{9}$$

where B is the binary image and θ a threshold value that is determined experimentally. In our experiments we found that for $\theta = 650$ (about 1% of the maximum of $P_{A_i/B_i} = 65535$) we get reliable results. Fig. 4 shows the motion image and its corresponding binary image. White areas (ones) on the right side indicate unmatching raw values.

Fig. 4: The moving hand from Fig. 2 with extracted motion artifacts. Left: The motion image M calculated using Eq. 8. Right: The binarized image B thresholded using Eq. 9.

Motion Direction Estimation Inspired by the idea of the optical flow motion estimation, we propose a simpler way to determine a 2D vector displacement map (U, V) without subpixel precision. Each pixel value represents a unique displacement vector $(u, v)^T$.

Our approach assumes a linear motion between all raw phase images with a

constant velocity (see Sec. 5). A pixel-wise motion displacement is estimated for all detected pixels in B. Therefore a motion window around every invalid pixel is defined, which limits the detectable motion around these pixels. The window is assumed to be squared with an odd size between 3 and 11 pixels (**M**otion **W**indow **S**ize, MWS). Vectors from the center (the invalid pixel) to all neighbors are calculated and scaled according to the phase image index. Let (dx, dy) be a single delta for a possible pixel correspondence shift between adjacent phase images, then the respective shifted phase values for the i-th phase image are given as:

$$P_{shifted,i}(x, y) = P_i(x + i \cdot dx, y + i \cdot dy), i \in \{0, 1, 2, 3\} \tag{10}$$

P_i and $P_{shifted,i}$ represent the particular phase image with index i. dx and dy are the applied deltas from the the center (see also Fig. 6) with a maximum value of:

$$dx_{max} = dy_{max} = (MWS - 1)/2 \tag{11}$$

and an odd Motion Window Size (MWS). The maximum euclidean pixel distance l between two corresponding points of phase image P_0 and P_3 is given as:

$$l = 3 \cdot \left|\left|\overrightarrow{(dx_{max}, dy_{max})}\right|\right| \tag{12}$$

In compliance with Eq. 12 and some knowledge about the expected motion in a scene, the motion window size can be preset to optimize the system performance. In our examples we set $MWS = 5$, yielding reliable results in most of the situations (see Sec. 5). Fig. 5 shows, how search vectors are defined and introduces the coordinate system exemplarily for a 5×5 motion window.

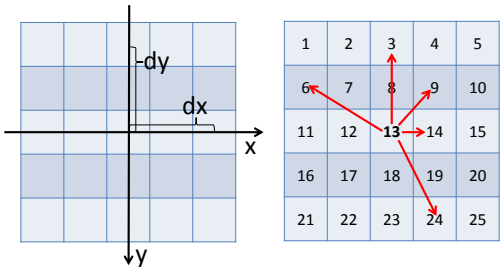


Fig. 5: Left: The coordinate system and an example offset $dx = dy = 2$. Right: The grid cells are numbered in a row-wise order. Index '13' indicates the start position. Five sample vectors are presented here.

For the estimation of the best corresponding flow, the Sum of Squared Differences (ssd) for all possible flow vectors within the defined motion window is calculated:

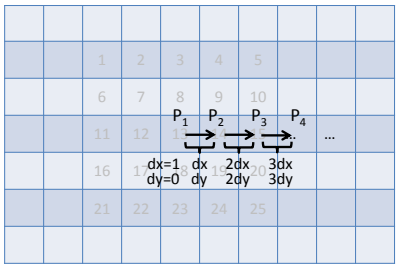


Fig. 6: An example flow for a shift $dx = 1$ and $dy = 0$. This leads to a maximum shift between the four phase images of 3 pixels.

$$ssd_{index}(x, y, dx, dy) = \sum_{1}^{3} (P_0(x, y) - P_{shifted,i}(x, y, dx, dy))^2 \tag{13}$$

Accordingly, the best correspondence value has the minimal deviation from the first phase image. So the final flow vector for the currently processed pixel can be expressed as:

$$(u, v)^T = \text{argmin}\,(ssd(x, y, dx, dy)) \tag{14}$$

The number of possible vectors MWS^2 is leading to a time complexity $T(n) = \mathcal{O}(n^2)$. A quadratic complexity allows only small motion window sizes (about 11×11) to perform the algorithm in real-time. To overcome this problem, the search direction can be restricted to an initial or mean direction from a previous frame.

4.3 Search Space Reduction

An additional performance optimization can be achieved using a search space reduction as can be seen in Fig. 7. Therefore the mean direction angle of a previous frame is used as initial guess for the current motion. The direction angle $\varphi_{(u,v)^T}$ for one pixel is calculated between the positive x-axis $\overrightarrow{x}$ and the corresponding flow vector $\overrightarrow{f(u, v)}$. The mean motion direction angle φ is defined as average of all estimated flow vector direction angles. Assuming a small motion between two consecutive frames, the amount of change of the mean direction angle is small. Now using this assumption, all pixel (x', y') in the search window whose position vector has an angle in the range of $\varphi \pm \rho/2$ ($0° \leq \rho \leq 359°$) are taken into consideration for the motion estimation. The raw phase value correction is then applied to the reduced flow field as described in Sec. 4.5.

All possible flow direction angles in a motion window can be precalculated. So it is also easy to port this to small platforms as e.g. an FPGA by using lookup-tables.

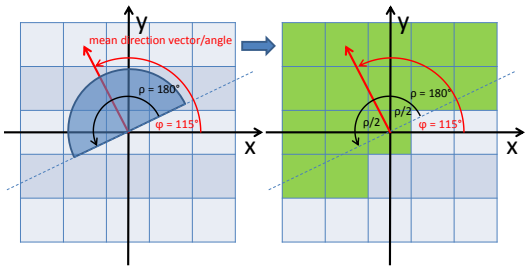


Fig. 7: Left: The mean motion vector (115° in this example) calculated in a previous frame (red arrow) and an exemplary search space reduction to 180°. Right: Valid motion vectors resulting out of the reduction are marked green.

4.4 Flow Field Optimization

In order to improve the robustness of the approach, a possible optimization is optionally applied taking the spatial neighborhood into account. A median filter is used to filter outliers. Therfore all motion vectors in a neighborhood with size MWS are considered. Median filtering is then performed using the direction angle as defined in Sec. 4.3. The vector length is kept.

4.5 Raw Phase Value Correction

Once the flow field (U, V) is determined, it can be applied to the raw phase values $P_{A_{0-3}}$ and $P_{B_{0-3}}$ according to Eq. 10. Each vector of the flow field is applied to its corresponding raw value. After the data correction has been applied, the depth values can be reconstructed according two the principle described in Sec. 3. The results and evaluation can be seen in Sec. 5.

5 Results

The following subsections give detailed information about the motion compensation results obtained with our proposed method. To perform a comprehensive analysis, the evaluation is split in two parts. In the first part, a quantitative evaluation is done, to allow the comparison of results against ground-truth data. Using artificial scenes gives reproducable and reliable results. In the second part, we do a qualitative evaluation in real scenes. Having the disadvantage that generally no ground-truth models are available, a visual evaluation makes it possible to see if the correction could successfully be applied.

5.1 Quantitative Results

Our method has been tested in a variety of scenes of different complexities (simulated and real environments). A robustness and peformance evaluation can be done using simulated data (see Table 2 and Table 3). Similar to Lefloch et al.

[9] we use different data sets generated with a simulator [10]. The statistic evaluations are done with the tool CloudCompare [3]. The first data set is a buddha figure, the second is a dragon. Both figures are used as input for the simulator. An artificial, planar wall is placed in a distance of 4 meters. In front of this wall, in a distance of about 3 meters, the figures are placed. This setup provides reliable ground-truth data. To obtain motion data, we acquire several different images at different camera positions. The camera is transformed between each individual phase acquisition. For the buddha, we use a simple lateral camera translation of 1cm (about $2m/s$ motion speed). In the dragon figure setup, the camera is rotated 1 degree (about $200deg/s$ angular velocity) around the z-axis (line of sight).

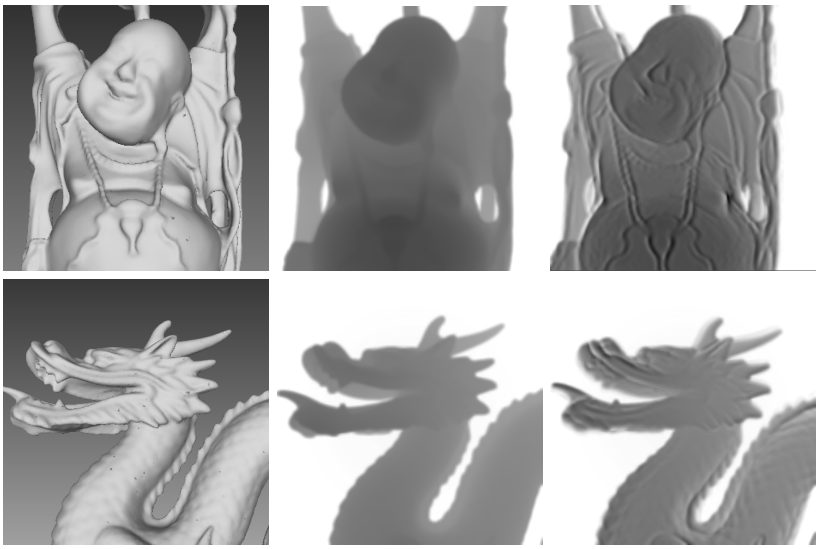
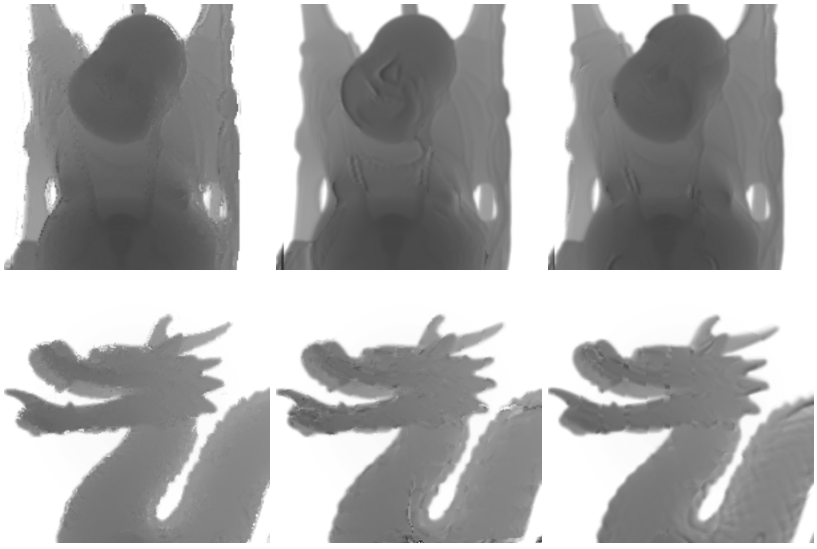


Fig. 8: Two different data sets (buddha (top) and dragon (bottom)) that have been used for the robustness evaluation of our approach; Left: The ground-truth 3D model. Center: The cartesian distance image without any motion. Right: The cartesian distance image with motion.

Using the ground-truth of Fig. 8, we can easily generate comparable results between the different motion compensation approaches (see also Fig. 9).

[3] http://www.danielgm.net/cc/



Proposed method Lindner et al. Lefloch et al.

Fig. 9: The results of the three evaluated methods. As can be seen, all the methods give good results visually comparing them to the ground-truth distance with no motion shown in Fig. 8.

Scenes	**Distance errors from Ground-truth (cm)**			
	Static (no motion)		Dynamic (with motion)	
	Mean	Sigma	Mean	Sigma
Buddha	0.64	3.12	5.96	9.32
Dragon	1.25	4.62	7.75	14.37

Table 1: The deviation of the ground-truth depth data (flying pixels included) of the buddha and dragon scene from the underlying meshs. It shows the mean distance error and the deviation for the static and the dynamic scene.

Table 1 shows a statistic evaluation of the buddha and dragon scene without motion compensation; here, static background pixels are discarded. The dynamic scene is created as previously described.

For further evaluation, several different setups are created to verify the quality of the proposed method. The first test shows the behavior of the algorithm with different settings for the Motion Window Size (MWS), neighborhood filtering (NH), motion area estimation (θ) and also search space restriction (ρ) (see Sec. 4.2). Table 2 and Table 3 contain the test results of the different setups and show the detailed behaviour of the proposed algorithm using different parameter sets. Especially the remaining depth error compared to the ground-truth data and the system performance is highlighted. It can be seen that as expected, the

best results are given without any limitation and restricition of the search space ($\theta = 0$, $\rho = 0$). The mean error of the buddha motion scene is reduced from $5.96cm$ ($\pm 9.32cm$) to $1.14cm$ ($\pm 3.02cm$), the dragon scene is corrected from a mean error of $7.75cm$ ($\pm 14.37cm$) to $2.07cm$ ($\pm 5.65cm$). Furhtermore it can be seen that with an increasing θ the correction performance gets better, but the quality decreases. Another fact that gets visible is, that using the neighborhood flow smoothing also improves the mean error, but with the disadvantage of losing performance: With the same settings and neighborhood filtering we can correct the buddha scene in $16.13ms$, without neighborhood filtering it takes $11.57ms$ only. A similar behavior can be seen for the dragon scene in Table 3. In addition, restricting the algorithm to a maximum direction deviation also improves the correction quality (mean error) and the system performance. This can be achieved by the rejection of a large number of search vectors in the motion window. We reject up to 36% of the possible directions ($MWS = 7$, $\rho = 90°$, 1000000 direction search vectors, rejected directions between 110770 and 716877) in the buddha scene and up to 30% ($MWS = 7$, $\rho = 90°$, 1000000 direction search vectors, rejected directions between 118916 and 583470) in the dragon scene. Please note that the execution time is an average value of 100 measurements. Furthemore it can be seen that the mean motion direction φ most closely approximates the expected linear translation of the buddha scene of 180°.

Additionally we compare our algorithm against the methods proposed by [8] and [9]. Our approach reduces the mean error of the buddha scene to 1.14cm ($\pm 3.02cm$), compared to Lindner $1.13cm$ ($\pm 4.39cm$) and Lefloch $1.46cm$ ($\pm 4.40cm$). For the dragon scene, the remaining mean error with our method is $2.07cm$ ($\pm 5.65cm$), for Lindner $2.26cm$ ($\pm 7.57cm$) and for Lefloch $3.14cm$ ($\pm 8.00cm$). The results between the three compared methods are nearly equal, but our method can score with the execution time, which is about half the time of the method from Lefloch et al. and an eight of Lindner et al.

The result of our tested setups is a good correction compared to the input mean error that can be seen in Table 1. Furthermore in comparison with Lindner and Lefloch, our method gives slightly better (dragon scene) or nearly equal (buddha scene) results and is also suitable for real-time applications with a framerate of 50–100 FPS allowing additional data processing as requested. Note: Compared to the evaluation of Lefloch et al. [9], we use a smaller clamping distance ($3.85m$) to remove the wall, explaining the slightly different mean and sigma values. In our opinion a good default parameter set is a threshold $\theta = 650$ and $MWS = 5$. Another helpful setting is a direction restriction to the mean motion direction. The default settings and the search area restriction significantly optimize the system performance and the motion compensation quality. Our tests were executed on an Intel Core i7-3770K CPU @ 3.50 GHz and an NVIDIA GeForce GTX 680, 2GB graphics card.

5.2 Qualitative Results

This part of the evaluation shows the behavior of real environments and applications. Two different setups are built. Unfortunately, there are no ground-

Buddha Scene				**Distance errors from Ground-truth corrected**				
MWS	NH	θ	ρ (°)	**Mean** (cm)	**Sigma** (cm)	φ (°)	Rejected Directions	⌀ Time (ms)
5	-	0	-	**1.15**	3.10	-	0	12.58
5	-	650	-	1.70	3.54	-	0	11.57
5	-	3276	-	3.49	4.70	-	0	10.59
5	-	5243	-	4.22	5.18	-	0	10.13
5	x	650	-	1.38	3.05	-	0	16.13
5	-	650	90	1.34	3.33	206.36	118916	**9.79**
5	x	650	90	1.46	3.24	223.98	110770	13.18
5	-	650	180	1.34	3.38	179.21	212408	11.08
5	x	650	180	1.27	3.04	190.33	212408	13.49
7	-	0	-	**1.14**	3.02	-	0	25.04
7	-	650	-	1.16	3.08	-	0	23.54
7	x	650	-	1.34	3.03	-	0	31.86
7	-	650	90	1.35	3.35	174.34	716877	**12.15**
7	x	650	90	1.35	3.13	178.90	716877	21.16
7	-	650	180	1.35	3.37	179.74	477918	15.18
7	x	650	180	1.37	3.04	187.10	477918	23.53
Method Lindner et al.								
-	-	-	-	1.13	4.39	-	-	71.87
Method Lefloch et al.								
-	-	-	-	1.46	4.40	-	-	25.60

Table 2: The statistic evaluation of the buddha scene and the behavior of the mean error in relation to different parameters. Statistics are shown for different motion windows sizes, neighborhood filtering (NH) on(x) and off(-), binarization thresholds θ and a search space restriction ρ.

truth values for these real world data sets, therefore they are limited to a visual comparison. The first scene shows a moving hand as can be seen in Fig. 10. The hand is moved very fast from one side to the other. The figure shows how the blurred images are corrected using our proposed method. Furthermore the images also show the motion area and direction restriction ($MWS = 5, \theta = 650, angle = 90°$). It can be seen that the blur is fully corrected.

The second evaluated scene contains a car moving lateral in front of the camera. Motion occurs mainly on edges, the mirror and the wheels. The visual determined movement direction is about 270°. The algorithm is parameterized with $MWS = 5, \theta = 650, angle = 90°$. Area, direction and also the correction is successfully applied and leads to the expected results as can be seen in Fig. 11.

6 Conclusion

In this paper we presented a new method for a fast motion artifact compensation for Time-of-Flight cameras. The approach is based on several assumptions

Dragon Scene				**Distance errors from Ground-truth corrected**				
MWS	NH	θ	ρ (°)	**Mean** (cm)	**Sigma** (cm)	φ (°)	Rejected Directions	⌀ Time (ms)
5	-	0	-	**2.08**	5.70	-	0	12.09
5	-	650	-	2.09	5.71	-	0	11.97
5	-	3276	-	2.43	6.04	-	0	11.57
5	-	5243	-	2.62	6.01	-	0	10.89
5	x	650	-	2.22	5.51	-	0	14.31
5	-	650	90	2.12	5.49	186.68	259320	**9.40**
5	x	650	90	2.37	5.75	206.36	118916	11.89
5	-	650	180	2.16	5.68	182.36	172880	9.83
5	x	650	180	2.33	5.70	198.20	131326	13.13
7	-	0	-	**2.07**	5.65	-	0	21.44
7	-	650	-	2.07	5.65	-	0	20.55
7	x	650	-	2.35	5.48	-	0	26.45
7	-	650	90	2.44	5.63	210.65	176064	**11.30**
7	x	650	90	2.17	5.66	184.70	583470	17.61
7	-	650	180	2.18	5.77	184.02	388980	14.28
7	x	650	180	2.42	5.64	201.35	206714	20.67
Method Lindner et al.								
-	-	-	-	2.26	7.57	-	-	80.76
Method Lefloch et al.								
-	-	-	-	3.14	8.00	-	-	24.07

Table 3: The statistic evaluation of the dragon scene and the behavior of the mean error in relation to different parameters. Statistics are shown for different motion windows sizes, neighborhood filtering (NH) on(x) and off(-), binarization thresholds θ and a search space restriction ρ.

such as linear motion between the four consecutive phase images of the PMD camera. Our algorithm uses a thresholding and binarization method to restrict the artifact correction area to spaces where in fact motion occurs. Furthermore we propose an approach to find pixel correspondences in a local neighborhood (motion field estimation), a local search area minimization by tracking the mean motion direction of a previous frame and an optional motion field smoothing. We show that the algorithm gives good results for simulated data (linear and non linear motion) and also for real data. Furthermore we show that we get comparable results in the mean value correction (compared to Lindner et al. [8] and Lefloch et al. [9]) and the algorithm can work in real-time (execution time about 10 ms and a frame rate of up to 100 FPS).

The proposed method still has a high potential to be optimized so that it also supports phase image motion correction. Furhthermore the threshold θ can be automatically adapted via statistics of the observed scene. The algorithm is also designed in a way that allows for easy porting to smaller hardware as an FPGA (no subpixel flow, possibility of lookup-tables for the search area reduction and parallelization).

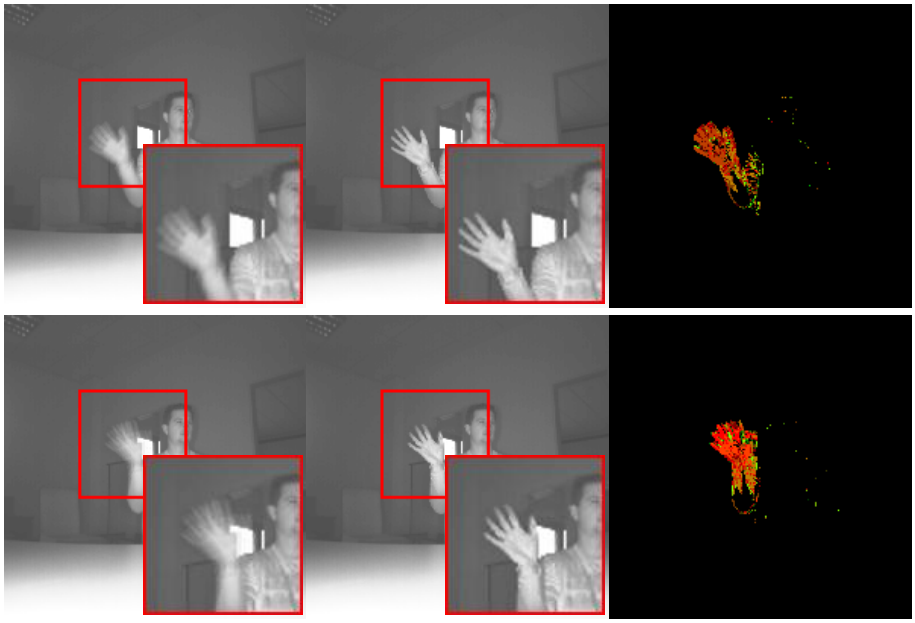


Fig. 10: Hand scene: the left column contains images with motion artifacts, the middle column contains the corresponding motion compensated images using our proposed method and the right column contains the related flow images (red: horizontal motion, green: vertical motion). The mean estimated motion direction for the top row is $\varnothing = 125.08°$, for the bottom row $\varnothing = 91.91°$.

References

1. pmdtechnologies GmbH: www.pmdtec.com (May 2013)
2. Prusak, A., Melnychuk, O., Roth, H., Schiller, I.: Pose estimation and map building with a time-of-flight-camera for robot navigation. Int. Journal Intell. Sys. Techn. & App **5**(3) (2008) 355–364
3. Marvin, L., Ingo, S., Andreas, K., Reinhard, K.: Time-of-flight sensor calibration for accurate range sensing. Comput. Vis. Image Underst. **114**(12) (December 2010) 1318–1328
4. NVIDIA: www.nvidia.com (May 2013)
5. Hussmann, S., Hermanski, A., Edeler, T.: Real-time motion artifact suppression in tof camera systems. Instrumentation and Measurement, IEEE Transactions on **60**(5) (2011) 1682–1690
6. Schmidt, M.: Analysis, Modeling and Dynamic Optimization of 3D Time-of-Flight Imaging Systems. PhD thesis, IWR, Fakultät für Physik und Astronomie, Univ. Heidelberg (2011)
7. Lee, S., Kang, B., Kim, J.D., Kim, C.Y.: Motion blur-free time-of-flight range sensor. In: Proceedings of the SPIE Electronic Imaging. (2012)
8. Lindner, M., Kolb, A.: Compensation of motion artifacts for time-of-flight cameras. In Kolb, A., Koch, R., eds.: Dynamic 3D Imaging. Volume 5742 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 16–27

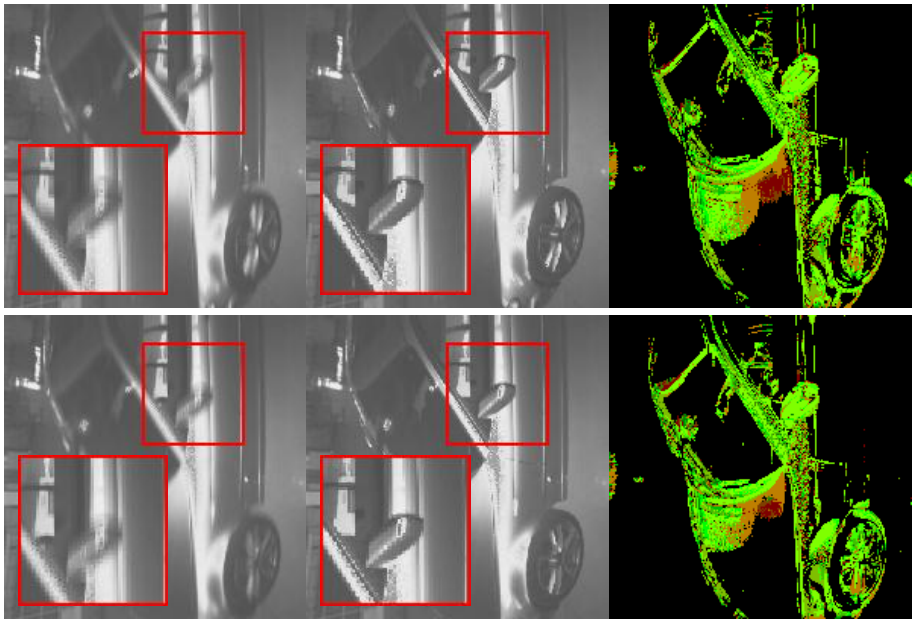


Fig. 11: Car scene: The left column contains images with motion artifacts, the middle column contains the corresponding motion compensated images using our proposed method and the right column contains the related flow images (red: horizontal motion, green: vertical motion). The mean estimated motion direction for the top row is $\varnothing = 228.20°$, for the bottom row $\varnothing = 232.71°$.

9. Lefloch, D., Hoegg, T., Kolb, A.: Real-time motion artifacts compensation of tof sensors data on gpu. In: Proceedings of SPIE Volume 8738. (2013)
10. Keller, M., Kolb, A.: Real-time simulation of time-of-flight sensors. J. Simulation Practice and Theory **17** (2009) 967–978