Fast, Accurate and Robust Registration of Multiple Depth Sensors Without Need for RGB and IR Images

Andre Mühlenbrock, Roland Fischer, Christoph Schröder-Dering, René Weller and Gabriel Zachmann
Computer Graphics and Virtual Reality
University of Bremen, Germany
{*muehlenb, rfischer, schroeder.c, weller, zach*}*@cs.uni-bremen.de*

***Abstract*—Registration is an essential prerequisite for many applications when a multiple-camera setup is used. Due to the noise in depth images, registration procedures for depth sensors frequently rely on the detection of a target object in the color or infrared image. However, this prohibits use cases where neither method is available or there is no mapping between the pixels of the color, infrared and depth images, e.g. due to separate sensors or different projections.**

We present our novel registration method that requires only the point cloud resulting from the depth image of each camera. For feature detection, we propose a combination of a custom-designed 3D registration target and an algorithm that is able to reliably detect that target and its features in noisy point clouds. Our evaluation indicates that our lattice detection is very robust (with a precision of more than 0.99) and very fast (on average about 20 ms with a single core). We have also compared our registration method with known methods: Our registration method achieves an accuracy of 1.6 mm at a distance of 2 m using only the noise depth image, while the most accurate registration method achieves an accuracy of 0.7 mm requiring both the infrared and depth image.

***Keywords*-Point clouds; registration; extrinsic calibration; depth sensors;**

I. Introduction

Depth sensors (e.g., ToF cameras and LiDAR sensors) are widely used in research and industrial applications thanks to the variety of available and affordable products. These sensors are often utilized in telepresence and robotic applications for tasks such as 3D reconstruction, SLAM, or object recognition. In order to cover large spaces or to avoid occlusion, multiple cameras are used. In these cases, the sensors must be calibrated intrinsically (individually) and extrinsically (to each other) to obtain a common point cloud.

For multi-camera calibration and registration, the classical approach is based on feature detection on flat checkerboards since those features can be detected reliably and accurately in color and infrared images. The first step is to detect the checkerboard itself; then, the inner corners can be extracted using known corner detectors (e.g., [1]), which serve as point correspondences. Using these correspondences, a rigid transformation between different sensors is computed. This approach has been improved continuously (e.g., [2], [3]) and generally leads to robust and accurate results.

However, this method is not always applicable, for example, when infrared or color images are not available. In addition, Reyes-Aviles et al. [4] reported that, depending on the camera model, the infrared images and the corresponding depth images may have different projections (which they observed, for example, with different Orbbec RGB-D sensors) which leads to errors when infrared images are used for registration. On the other hand, if registration is performed directly on depth images or point clouds, the problem of inherently noisy depth data arises, making accurate feature detection difficult [5].

Our registration procedure, which we present here, requires only 3D point clouds obtained from depth images (no color or infrared data needed), works independently of scene brightness, can register both sensors facing in the same direction and opposing depth sensors, and is very easy and quick to perform. At the same time, we achieve excellent results in our evaluations with the Microsoft Azure Kinect, which are:

- low registration errors (avg. 1.6 mm at 2 m distance),
- robust target detection (PPV > 0.99),
- fast target detection (avg. 20 ms).

We accomplish this by using a custom-designed lattice-like 3D registration target that can be easily replicated and a pipeline designed to detect the target's features accurately in noisy depth images. Our implementation is available both as a self-contained small C++ library and as an Unreal Engine 4 plugin with a sample project, which also allows registering the depth sensors into a virtual world.

This paper is an extension of our CW 2021 paper [6]. In addition to a more detailed explanation of the algorithm and evaluation, we present a new experimental setup and perform an evaluation in which we compare the accuracy of our registration method to commonly known registration methods (see Section IV-A). Furthermore, we supplement this paper with a new small C++ library, which allows the use of our registration procedure in other software projects.

II. Related Work

As mentioned earlier, the calibration and registration of depth sensors is usually done via the accompanying IR or RGB sensor images: Macknojia et al. [7] synchronously

captured a checkerboard in the color and IR images of a Kinect for extrinsic calibration between RGB and depth sensors. Registration (or extrinsic calibration) between multiple Kinect cameras was similarly performed using the respective IR images. Chen et al. [8] captured a checkerboard in the color and IR images for homography-based calibration, while Darwish et al. [9] tracked two orthogonal checkerboards and aimed to improve depth accuracy.

In other publications, e.g. [10], [11] and [12], the use of the checkerboard approach is also described, but external optical tracking systems were added for depth correction and registration of multiple cameras in a common coordinate system, respectively. Although in this case the viewing areas of the cameras do not need to overlap, the need for a tracking system is a major limitation.

Herrera et al. [13] proposed a calibration approach that works directly in the depth image by detecting the outer corners of the checkerboard through a time-consuming manual selection. Reyes-Aviles et al. [4] proposed using a 3D checkerboard as a calibration target and a method that includes normal estimation, edge detection, and thresholding to detect it in the depth image. The registration method proposed by Song et al. [5] is based on a special checkerboard with regularly spaced hollow squares. Depth variations are handled by a model-based approach that considers the centers of the holes. Some works such as [14], [15] have replaced checkerboard-like registration targets with static marker-free 3D objects with known or previously scanned geometry, e.g., a stack of boxes that can be detected in depth images or point clouds. Furthermore, spherical targets have also been presented for camera calibration (e.g. [16], [17]) and registration in multiple-camera setup (e.g. [18]), in which the target was detected in the RGB image using ellipse fitting and in the depth image using background subtraction or a spherical area detection and sphere fitting of the point cloud.

Another popular approach is to perform a target-less registration directly on the 3D point clouds of the surrounding 3D scene obtained from multiple depth cameras. The most well-known algorithm is Iterative Closest Points (ICP), which alternatively searches for the closest point-to-point correspondences and an optimal rigid transformation. However, its main drawback is the tendency to converge to a local minimum and, therefore, its high dependency on good initial guesses [19]. Numerous variants were proposed in order to improve the convergence [20], the computational speed [21] and the robustness to noise [22]. Typical global methods for 3D registration are based on feature matching (including detection and description) and transformation via RANSAC. Generally, these methods tend to be less precise and, depending on the number of outliers, more time-consuming [23]. A major problem with target-less registration occurs when the point clouds do not overlap completely or only slightly. Different recent approaches try to obtain a registration even in these cases, e.g. [24] and [25]. However, if there is no overlap of the scene at all, e.g. because depth sensors are facing each other, these target-less methods are not applicable.

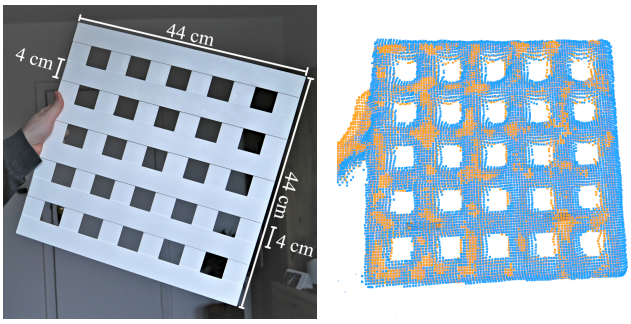


Figure 1: Left: photo of our lattice with 25 holes. Right: two point clouds registered using our method.

III. Our Approach

We designed a lattice-like 3D target and developed a lattice detection algorithm that allows for quick and easy registration (extrinsic calibration) of multiple depth sensors. Our lattice-like 3D target (see Fig. 1) consists of 12 bars of size 44 cm x 4 cm x 0.2 cm, available in a common DIY store. By leaving 4 cm of space between the bars in the vertical and horizontal directions, 25 holes of size 4 cm x 4 cm are created that can be detected in the point cloud generated by depth sensors.

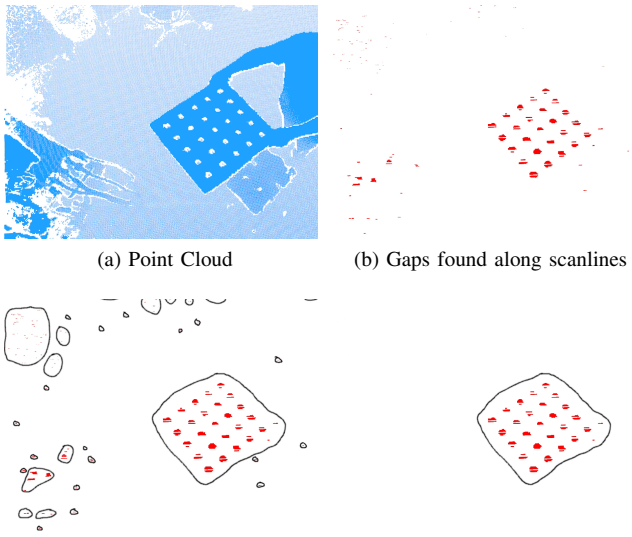
To perform the registration, the lattice has to be moved in the field of view of the depth sensors for a few seconds, while our lattice detection algorithm detects up to 25 point correspondences per frame for all depth sensors. Based on these point correspondences of multiple frames, we determine a rigid transformation matrix that describes the transformation of the sensors with respect to each other.

The major challenge in classical camera registration is the correct and accurate recognition of feature points. Since we only use depth data and no RGB or IR data, we cannot easily reuse the image-based recognition algorithms.

In the following, we present a novel approach that is fast and easy to implement while still achieving robust results. The detection of the lattice consists of the following steps:

1) Identification of plausible lattice candidates.
2) Detection of hole centers.
3) Identification of the center and axes of the lattice and outlier removal among hole centers.
4) Correspondence mapping.
5) Correspondence rejection.
6) SVD-based transformation estimation.

In the following, we will explain the individual steps of the lattice detection. Note that our algorithm expects an array of 3D coordinates as input — i.e. a point cloud — that is available in scanline order of the original depth image.



(a) Point Cloud (b) Gaps found along scanlines

(c) Clusters of gap segments (d) Plausible clusters

Figure 2: The steps in which we identify lattice candidates.

A. Lattice Candidates

Initially, we are given a point cloud in scanline order from the depth image, without indication as to whether a lattice is visible and where the lattice is located. To obtain clues as to where lattices might be located in the point cloud, we exploit the property that the lattice has many regularly spaced holes that result in many gaps in scanline order.

So, in the first step, we search for gaps along the scanlines which correspond to the regular geometry of the lattice (see Fig. 2). We do this by segmenting individual scanlines based on the Euclidean distance between neighboring points. Scanline segments that are at most as long as the diagonal of a hole and that are surrounded by two scanline segments of plausible length are identified as *gap segments*.

Now we have many individual segments lying along individual scanlines, each of which could be within a hole of the lattice. However, since we want to find a section of the point cloud that fully encloses the lattice, we cluster all *gap segments* using their *Virtual Gap Centers* [1] based on their proximity to each other. Using PCA, we calculate eigenvectors and eigenvalues for each cluster. Based on the proportions of the eigenvalues, we can efficiently discard clusters that obviously cannot represent lattice candidates: due to the symmetric structure of the lattice, we expect the first two eigenvalues to be similar in size, while the third eigenvector is many times smaller (we use a factor of 10 as the threshold), since the lattice is flat. All remaining clusters of gap segments are considered as lattice candidates.

[1]Since gap segments lie in holes of the lattice where the lattice has no geometry, the corresponding 3D points of the gap segment may lie behind the lattice or be invalid. So, to calculate 3D coordinates of the gap segments that lie on the plane of the lattice, we calculate a Virtual Gap Center for each gap segment by averaging two 3D points that lie directly to the left and the right of the gap segment since these are 3D points of the lattice's geometry. This resulting *Virtual Gap Center* (a) is located in the center of the gap segment in image space and (b) lies approximately on the lattice's plane in 3D space.

(a) Before filtering (b) After filtering

Figure 3: Side view of the lattice in the point cloud (blue) before and after filtering noise (e.g. due to the flying pixel effect). This is done by fitting a plane (black) via RANSAC into these points, then culling points by thresholding.

B. Hole Center Detection

Using the clusters left over from the previous step, we now know areas where a lattice may be located. In this step, we try to identify the exact hole centers of the lattice for each area found and discard lattice candidates that turn out to be implausible in the following.

We first determine all 3D points that potentially belong to the physical lattice due to their proximity to the *gap segments* using their *Virtual Gap Centers* (see footnote 1). A point is considered a lattice point if it is within a certain radius to at least one *Virtual Gap Center* (we use $r = 0.16\,\mathrm{m}$ to completely cover the lattice in case some holes were missed in the previous step).

To effectively filter out the noise that typically occurs with depth sensors (e.g., due to the flying pixel effect), we use RANSAC [26] to fit a plane to the point cloud section and define all points closer than a certain threshold to the plane as lattice points (see Fig. 3). We store the indices of these lattice points in an *inlier set*. All remaining points are defined as outliers.

To identify the holes of the lattice, we again iterate over all scanlines of the input point cloud, each from the first inlier to the last inlier. We create segments similar to Section III-A but this time we create segments of lattice points (which are contained in the *inlier* set defined above) and segments of outliers. All outlier segments, which are enclosed by inlier segments are assumed to be a part of a hole. Since we have only iterated over the horizontal scanlines so far and thus only have horizontal segments, we now vertically join adjacent outlier segments if they overlap horizontally to obtain one segment for each hole of the lattice. This vertical joining of the horizontally running outlier segments is done efficiently using the union-find structure.

The remaining and joined outlier segments represent the individual holes. However, since the points of these joined outlier segments do not lie in the plane of the lattice, we consider the directly adjacent inlier points in each scanline

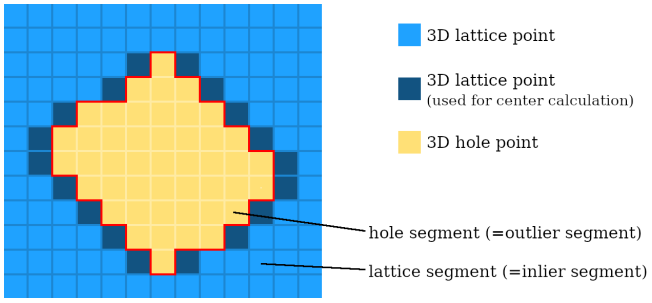


Figure 4: A hole of the lattice in image space. The red line visualizes the separation of both segments. The dark blue points are those 3D inlier points used to calculate the hole center by averaging them and projecting the average onto the estimated plane in which the lattice is located.

(see Fig. 4), project them onto the earlier fitted plane and use their average as hole centers.

C. Axes Detection and Outlier Removal

For each lattice candidate, we have found a set H of hole center candidates. However, there may be still incorrectly identified hole centers and additionally, we need correspondence points between multiple sensors, i.e. we have to match the found hole centers of different sensors.

For this purpose, we have developed a heuristic that can recognize the axes of the lattice based on the potential hole centers and that can cope even with quite noisy data. This heuristic works as follows:

1) Given the set of all found hole centers H, we now define the following set V of vectors:

$$V = \{n-m \mid d-\delta < dist(m,n) < d+\delta, \quad m,n \in H\}, \tag{1}$$

where d = *hole spacing* and δ is a tolerance (in our case $d = 8\,\text{cm}$ and $\delta = 2\,\text{cm}$ which represents the geometry of our lattice). These vectors can be interpreted as edges between the hole centers. Together, they form a proximity graph (see Fig. 5).

2) Sort the vectors $v \in V$ by their angle α they subtend with the x-axis:

$$\alpha(v) = \begin{cases} \text{atan2}(v_y, v_x) & \text{if atan2}(v_y, v_x) \geq 0 \\ \text{atan2}(v_y, v_x) + \pi & \text{otherwise} \end{cases} \tag{2}$$

3) Cluster these vectors based on their angle α. As can be seen in Fig. 5, this results in two very large clusters (color-coded by the violet and green edges) as well as several other very small clusters. Thinking of these vectors as edges of a proximity graph, we define the set of all "good" edges in one of those two largest clusters as G and the set of all "bad" edges in the remaining small clusters as B.

4) For each hole center $h \in H$ we now consider its incident edges $E(h)$ and remove hole centers based on the number of incident "good" edges and incident "bad" edges, leaving the following set:

$$H_{\text{filtered}} = \{h \mid \#E_G(h) > \#E_B(h), h \in H\}, \tag{3}$$

where $E_G(h) = E(h) \cap G$ and $E_B(h) = E(h) \cap B$. This way, we very reliably remove incorrect hole centers.

5) Using the remaining hole centers H_{filtered}, we look for the hole center candidate $h^* \in H$ that is closest to the average:

$$h' = \frac{1}{|H|} \sum_{h_i \in H} h_i, \tag{4}$$

The h^* closest to this h' will be considered the center of the lattice.

6) By selecting the median vector in both the largest clusters, we get two very stable vectors which points along the x-axis and the y-axis (see Fig. 5).

7) At this point, we do not know which of the two vectors represents the x-axis and which the y-axis as well as their signs. To resolve this ambiguity, we consider the points of the point cloud surrounding the lattice: these are usually the points of the hand and arms holding the lattice. So, we calculate a vector from h^* to the center of these hand points and flip both the previous found vectors and assign them so that the x-axis always points in the direction of the hands. Let us assume for the moment that always the front side of the lattice is visible in the depth image: Then we can set the plane normal found by RANSAC as the z-axis, which then determines the y-axis.

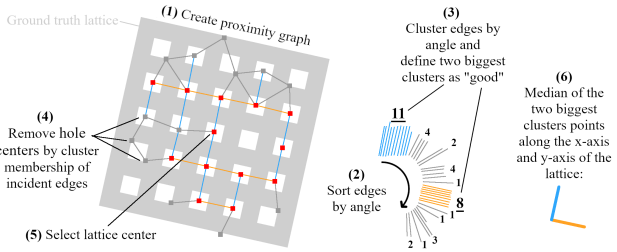


Figure 5: Visualization of our heuristic that can filter incorrectly detected hole centers (gray) and detect directions of x and y axes. In this example, the red and gray colored points where detected as hole centers previously.

If we find that the two vectors we determined in step (6) are not roughly orthogonal, or $\#H_{\text{filtered}}$ is too small, we discard this candidate, since it is more likely not to be the lattice in this case. Using this heuristic and making the preliminary assumption that the lattice is always visible from the front, we were able to determine the center and axes of the lattice as well as remove incorrect hole centers.

D. Point Correspondences

Generally, it would be possible to use only the centers of the lattice across multiple sensors as point correspondences

over multiple frames. However, the more point correspondences are used over a large space, the more accurate and stable the registration is expected to become. Therefore, it is desirable to use all the hole centers instead of just the center one. By using all found hole centers, we get up to 25 times more point correspondences over a larger space. This also considerably reduces the time needed to perform a registration.

However, since we still lack the information on whether the lattice in the original depth image is seen from the front or the back, we cannot yet establish a clear mapping of hole centers between multiple sensors. Therefore, we first perform a less precise registration with only two point correspondences per frame, namely (a) the lattice center as well as (b) the lattice center shifted in the direction of the x-axis, since the x-axis always points in the direction of the hands. After that rough registration, we can transform the z-axis vector of the lattice seen in sensor A into the coordinate system of sensor B and use $z'_B = \pm z_B$ as z-axis and $y'_B = \pm y_B$ as y-axis (since we created the y-axis using the z-axis), depending on which sign gives the dot product $z_A \cdot z_B$. In this way, we have resolved the ambiguity of the lattice side and ensure that lattices visible from the same side in different cameras have the same sign.

E. Correspondence Rejection and Registration

Up to this point, we have found point correspondences for which the 3D coordinates in the camera space of multiple sensors are known. However, since our lattice detection is not completely immune to errors, very rare errors in the point correspondences are possible. To ensure that in these cases the accuracy of the registration is not affected, we filter the point correspondences using the RANSAC-based correspondence rejection of the Point Cloud Library (PCL) [27]. Finally, we perform registration with the remaining point correspondences using SVD-based transformation estimation implemented by the PCL [27].

IV. Results

To investigate the accuracy, reliability, and runtime performance[2] of our registration procedure, we designed and conducted several experiments, which we present in this section. In all experiments, the Microsoft Azure Kinect was used as depth sensor. The following is a list of experiments we conducted, whereby experiments A and B are related to the whole registration procedure, and experiments C, D and E refer to the lattice detection in particular:

[2]Note that in terms of runtime, the lattice detection is the crucial part, because the lattice detection has to be executed for each sensor and for each frame compared to the rest of the pipeline, which is executed only once. The runtime for the rest of the pipeline is on the order of a couple of milliseconds including the correspondence rejection and SVD-based transformation estimation which is almost negligible when using many frames.

A **Accuracy Measurement and Comparison:** We present an experimental setup that allows for very accurate determination of the distance error of a point as well as an angular error between two registered depth sensors. We compare our lattice-based registration procedure with three variants of the well-known checkerboard registration procedure. Both depth sensors were synchronized in time, so that two matched frames were always recorded with a time offset of exactly 160 μs.

B **Common Coordinate System Registration:** We register a depth sensor into a ground truth coordinate system given by Optitrack, a high precision optical tracking system. The pose of the lattice is tracked both by Optitrack and by the depth sensor. The purpose of this experiment is also to determine the accuracy of the registration procedure, but in a different application – namely, the registration of a depth sensor with a third system. Major differences to experiment A are that (a) only the lattice center is taken as the point correspondence instead of all hole centers, since Optitrack isn't able to detect these, (b) there is no exact time synchronization of the frames between the two systems, and (c) the error is determined not only over one very accurate correspondence point, but over many correspondence points in a larger volume.

C **Rotational Robustness:** In this experiment, we rotate the lattice while its center position is fixed. This allows us to determine the minimum angle at which the lattice is still detected by our method and to detect possible systematic errors that depend on the angle of rotation. In this way, we can estimate whether our registration procedure is reliable in all situations (e.g., when the angle between the direction vectors of both depth sensors is very large) and whether higher errors can possibly be expected there than those we obtain in Experiments A and B.

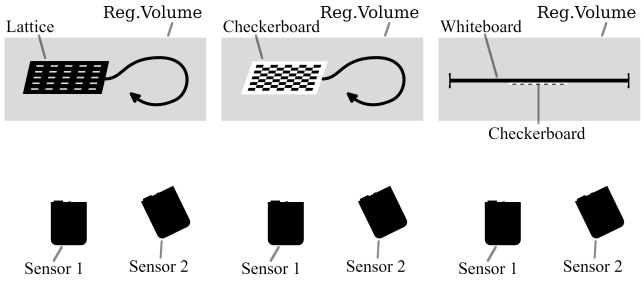
D **Runtime Performance:** We investigate the runtime performance of the lattice detection depending on the lattice distance since the lattice detection takes by far the largest part of the runtime (see footnote 2).

E **Precision and Recall:** The reliability of our registration method essentially depends on how reliably the lattice, including its hole centers, is found. Therefore, in this experiment, for three different scenarios, we look at how often the lattice was correctly detected when our algorithm detected something (Precision) and how often the lattice was correctly detected when a lattice should have been visible (Recall).

A. Accuracy Measurement and Comparison

With this experiment, we examine the accuracy of our lattice registration procedure and compare it to the accuracy of conventional checkerboard registration procedures.

The following registration procedures are evaluated:



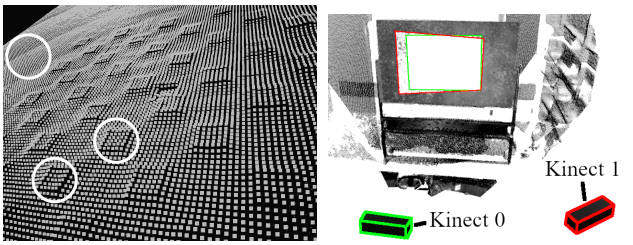
(a) Lattice Recording (b) Checkerboard Recording (c) Evaluation Recoding

Figure 6: Setup of the recordings made per run. In each of the 10 runs, we recorded (a) the moving lattice, (b) the moving checkerboard, and (c) the static whiteboard with a checkerboard in the center. The whiteboard in (c) was always located approximately in the middle of the registration volume (Reg. Volume) and on average about 2 meters away from the sensors.

- *Checkerboard (RGB):* We capture a moving *checkerboard* (with 8x8 inner corners) in the *RGB image*, detect its corners with *OpenCV's checkerboard corner detection* by Duda et al. [3] in image space, and then we use OpenCV's *stereoCalibrate* function [28] to obtain a registration for the RGB sensors.
- *Checkerboard (IR):* We capture a moving *checkerboard* (with 8x8 inner corners) in the *infrared image*, detect its corners with *OpenCV's checkerboard corner detection* by Duda et al. [3] in image space and then we use OpenCV's *stereoCalibrate* function [28] to obtain a registration for the infrared/depth sensors.
- *Checkerboard (IR+D):* We capture a moving *checkerboard* (with 8x8 inner corners) in the *infrared image*, detect its corners with *OpenCV's checkerboard corner detection* by Duda et al. [3] in image space, use the corresponding 3D points by the depth image, and apply the *correspondence rejection and SVD-based transformation estimation* implemented in PCL [27], which is also used by our method.
- *Lattice (D):* We capture a moving *lattice* in the point cloud given by the depth image, detect it using our lattice detection and perform registration which is based on the *correspondence rejection and SVD-based transformation estimation* implemented in PCL [27].

Note that the first three checkerboard registration methods (*RGB*, *IR* and *IR+D*) are well-known approaches in the community that we compare to our lattice-based registration (*D*).

Our experiment consists of 10 runs, for each of which we make (a) a recording with a moving lattice target, (b) a recording with a moving checkerboard target, and (c) a recording of a stationary whiteboard with a checkerboard



(a) Smoothed point cloud of the checkerboard glued onto the whiteboard (b) Points used for plane fitting

Figure 7: Uneven-perceived surface and plane fitting. In (a), the uneven-perceived surface of the checkerboard is visible. In (b), the area of points of the whiteboard (white) which we considered for plane fitting is shown.

pattern at its center (see Fig. 6). In each run, recordings (a) and recording (b) are used to perform a registration while recording (c) is an evaluation recording used to determine a point correspondence between the cameras' coordinate systems very precisely for distance error measurement. After each run, we slightly changed the position and orientation of both Azure Kinects to get a bit of variation while maintaining a distance of approximately 1.5 – 2.5 m to the center of the registration volume. Furthermore, we alternated the order whether the lattice recording (a) or the checkerboard recording (b) was done first. Both Azure Kinects were synchronized in time to avoid errors caused by a larger location offset of the registration target in the same frame of different sensors. However, to avoid interference between multiple Azure Kinects, the second Azure Kinect was delayed by 160µs as recommended by the manufacturer, which is negligible regarding the expected error.

To obtain a point correspondence for error measurement in recording (c), which is needed to determine the distance error, we proceed as follows:

1) We detect the checkerboard pattern glued to the whiteboard using OpenCV's checkerboard corner detection in the infrared image. According to [3], this yields subpixel accuracy.
2) Since, in the case of the Azure Kinect, the projections and sensors that generate the depth and IR image are identical, for each corner in the IR image, we obtain the 3D points of all corners, and average over all 3D corner points to get a mid point of the checkerboard.
3) While the 3D mid point is assumed to be very precise along the axes in image space, there may be small deviations along the depth axis due to the alternating colors of the checkerboard fields (see Fig. 7a). To control for that, we fit a plane to the 3D points rectangle that is centered to the checkerboard (see Fig. 7b). The previously calculated mid point is then projected onto this plane, which is finally used to estimate the error

Table I: Results of single registration runs. Error (mm) is the distance error measured by the correspondence point while Error (deg) is measured by the angle between the fitted plane normals.

	Error (mm)				Error (deg)			
Run	RGB	IR	IR+D	D	RGB	IR	IR+D	D
1	15.2	6.8	**1.4**	1.8	0.17	0.04	**0.0**	0.17
2	9.7	5.1	**0.6**	0.9	0.21	0.22	**0.09**	0.18
3	15.1	5.4	**0.6**	2.8	0.40	0.22	**0.06**	0.10
4	20.5	7.4	**0.7**	1.9	0.15	**0.02**	0.07	0.13
5	14.8	8.1	**1.2**	2.1	0.21	0.12	**0.09**	0.18
6	15.5	5.5	**0.4**	1.3	0.24	0.07	**0.04**	0.11
7	19.3	5.0	**0.3**	0.5	0.24	0.16	**0.15**	0.28
8	7.4	4.6	**1.0**	**1.0**	0.19	**0.02**	0.05	0.08
9	17.8	9.2	**0.5**	2.8	0.57	0.23	**0.10**	0.22
10	19.0	6.5	**0.6**	1.1	**0.08**	0.13	0.11	0.21
AVG	15.4	6.4	**0.7**	1.6	0.24	0.12	**0.08**	0.17
SD	4.0	1.4	**0.3**	0.7	0.13	0.08	**0.04**	0.06

between both point clouds.

Note that the registration transformation obtained by the Checkerboard (RGB) procedure differs somewhat from those obtained by the other procedures. This should be taken into account when comparing the 3D accuracy of the different calibration procedures. The reason for that arises from the fact that the Azure Kinect (and potentially many other RGB-D cameras) has two sensors: one sensor for both the IR and depth image, and a separate sensor for the RGB image. In order to perform a transformation of 3D points of Kinect B into the reference frame of Kinect A, the following concatenation of transformations should be used in case of the RGB calibration procedure:

$$T_{D_A \leftarrow D_B} = T_{D_A \leftarrow C_A} \cdot T_{C_A \leftarrow C_B} \cdot T_{C_B \leftarrow D_B}$$

where

- $T_{D_A \leftarrow D_B}$ denotes the transformation from the depth sensor of Kinect B to the depth sensor of Kinect A,
- $T_{C_B \leftarrow D_B}$ denotes the transformation from Kinect B's depth to its color sensor (given by the factory calibration),
- $T_{C_A \leftarrow C_B}$ denotes the transformation from Kinect B's color sensor to Kinect A's color sensor (known from the checkerboard registration),
- $T_{D_A \leftarrow C_A}$ is the transformation from Kinect A's color to its depth sensor.

Obviously, the error of the Checkerboard (RGB) registration procedure using the RGB sensors and the error of the factory calibration between depth and color sensors accumulate. Therefore, it is to be expected that the Checkerboard (RGB) registration has a higher error than the other registration procedures. This is verified by the results of our experiments (see Table I).

In all runs, the lattice was detected by both sensors in an average of 87.2 frames (SD: 32.0) of recording (a). In the RGB image of recording (b), the checkerboard was detected in 108.3 frames (SD: 39.7) in average and in the IR image in 87.4 frames (SD: 39.0) in average. Note that the recordings (a) and (b) are not identical – individual recordings of the checkerboard methods and the lattice method can therefore not be directly compared. Furthermore, the difference in the number of detected checkerboards in the RGB image and the IR image, both using recording (a), is due to the uneven brightness of the checkerboard at different distances in the IR image, so that the checkerboard was not detected in some cases in the IR image.

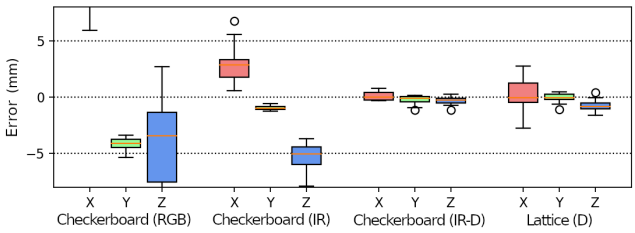
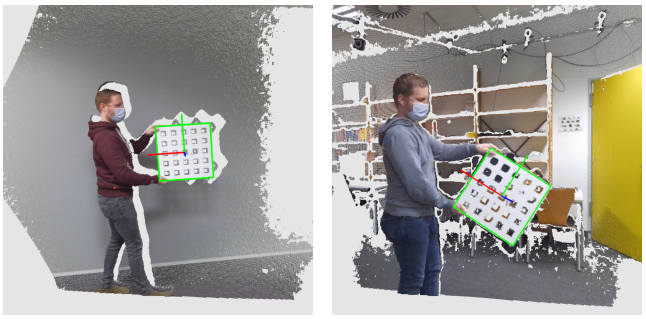


Figure 8: Distance error after registration by procedures and axes.

The results (see Table I) show that the average distance error of our method *Lattice (D)* (which only requires the depth image) is 1.6 mm and the average angular error is 0.17 deg. The *Checkerboard (IR+D)* method performs considerably better with an average distance error of 0.7 mm and an average angular error of 0.08 deg, but requires both the infrared image and the depth image. Compared to the *Checkerboard (IR)* method with an average error of 6.4 mm and the *Checkerboard (RGB)* method with an average error of 15.4 mm, our registration method *Lattice (D)* as well as the *Checkerboard (IR+D)* method perform significantly better.

Since both the depth image and the infrared image are combined as input in the Checkerboard (IR+D) method, this method is expected to be more accurate than the Lattice (D) method, which uses only the noisy depth image as input. However, even if the average error of the Checkerboard (IR+D) method is smaller than the average error of the Lattice (D) method, both errors are very small in absolute terms considering the accuracy of depth sensors, which are much noisier and suffer from distortions at edges or the flying pixel effect. Furthermore, one has to take into account that the lattice in the experiment has a thickness of 4 mm (two layers of 2 mm thick bars) and was built with a precision of about 1-2 mm. The motions of the lattice in the recordings, as well as the higher standard deviation in the X-direction for Lattice (D) in Figure 8, indicates that the thickness of the lattice may have mattered. Thinner and more precise lattices could improve the result of the Lattice (D) method.

Figure 8 shows that the *Checkerboard (RGB)* and *Checkerboard (IR)* procedures, in particular, contain a sys-



(a) Scenario A (b) Scenario B

Figure 9: Two scenarios used for ground truth evaluation, from the perspective of the depth sensor.

tematic error that could be caused by the factory calibration of the Kinect whose parameters are used in OpenCV's stereo calibration methods. In our experimental setup, the distance between two adjacent pixels in the registration volume is about 4 mm in physical space, so even small errors in the intrinsic calibration could have a significant impact on the distance error in both cases. On the other hand, these two methods do not use depth information, so a possible systematic offset in depth direction is not corrected by these methods. In the case of the *Checkerboard (RGB)* method, as described above, the errors in the transformations between the separate depth and RGB sensors also add up, leading to expected higher errors compared to the three other methods.

Finally, this experiment shows that our method is a valuable alternative registration method in cases where no IR or color image is available, and is also very accurate in terms of absolute error values, in regard to the accuracy of depth sensors.

B. Registration into a Common Coordinate System

In the previous section, we looked at the accuracy of a registration between two depth sensors using our registration method. In this section, we will examine the accuracy with which we can register a depth sensor into Optitrack's coordinate system using our method. To do so, we tracked the lattice using both Optitrack and a Microsoft Azure Kinect combined with our detection algorithm. To track the lattice with Optitrack, we attached seven markers to the lattice to achieve sufficient accuracy. These Optitrack markers were detected in the Azure Kinect depth image by our method as hole centers. However, our heuristic generally detected these hole centers as incorrect hole centers, resulting in no noticeable effect on the precision of the lattice detection.

Since Optitrack and the Kinect use the same infrared light at a wavelength of 850 nm, there was occasionally a pulsating noise throughout the depth image (see Fig. 10). We ran Optitrack at 30 fps (almost the same frame rate with which the Azure Kinect recorded), as the pulsating noise was least likely to show up this way. The pulsating noise caused a greatly increased runtime of the algorithm in those frames, since many lattice candidates were detected in flat background objects. However, all these false lattice candidates were successfully discarded by our detection algorithm.

Table II: Mean error between ground truth lattice center and detected lattice center after registration.

Scenario A	**Mean Error**	**SD**	**n**	**rem.**
Calibration Set	3.83 mm	2.10 mm	2401	20
Test Set	3.95 mm	1.69 mm	1880	1
Scenario B	**Mean Error**	**SD**	**n**	**rem.**
Calibration Set	4.38 mm	2.07 mm	1270	14
Test Set	4.40 mm	2.10 mm	871	5

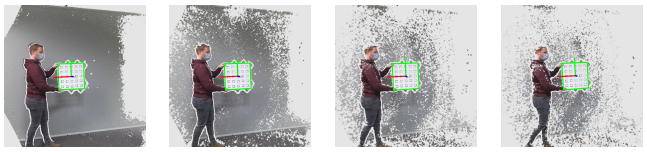
Note: Rare error detections with an higher error that 20 mm were excluded from the error calculation because they are removed during registration during outliner rejection anyway and generally have no influence on the registration result itself (the "n" column indicates the number of used frames whereas the "rem." column indicates the number of removed frames).

We performed the evaluation in front of two different backgrounds (see Fig. 9), hereafter also called scenarios, while slowly moving the lattice. In scenario B, the distance between the sensor and the lattice was between 1.0 m and 1.95 m while the center of the lattice stayed within a volume of about 0.6 m³.[3] In scenario A, the distance between the sensor and the lattice ranged from 0.85 m to 2.05 m, with the center of the lattice in a volume of about 1.2 m³ (the entire lattice was about 2.2 m³).

Using the lattice centers as point correspondences, we registered the Microsoft Azure Kinect's point cloud into Optitrack's coordinate system. We then measured the deviation of the registered center point from the center point detected by Optitrack. In both scenes we observed a quite similar error averaging only 3.83 mm to 4.40 mm (see Table II).

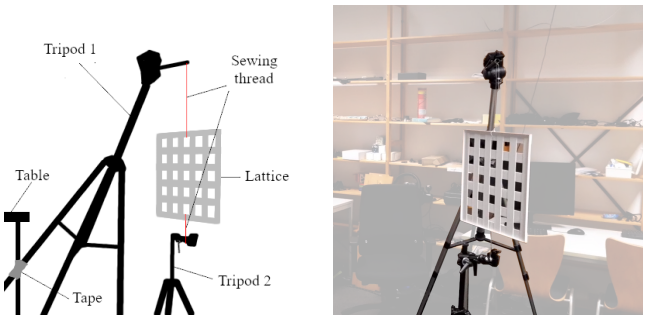
Note that, compared to experiment A (see Section IV-A), instead of just measuring the distance error of one point which was located in the center of the registration volume and was smoothed over time, we captured multiple correspondences in a specific volume which still were affected by the typical noise of the Azure Kinect. Additionally, Optitrack and the Kinect were not synchronized in time. Therefore we always searched for the closest Optitrack frame in time to a Kinect frame to find point correspondences. Although we set both Optitrack and the Kinect to 30 fps, they did not run at exactly the same speed. The closest frames in time between Optitrack and the Kinect were always time-shifted by 0 to about 1/60 second, averaging 1/120 second. With an average movement speed of 18.8 cm per second in scenario A, this gives an expected error of $0.0083\text{s} * 18.8\text{cm/s} = 1.56\text{mm}$.

[3]The entire lattice was in about 1.5 m³ in scenario B, but since we could only detect a single ground truth reference point via Optitrack, we only used the detected centers of the lattice as point correspondences between the two systems. For this reason, the smaller value is more relevant



(a) (b) (c) (d)

Figure 10: Pulsating noise in the recorded point cloud due to inference between Optitrack and the Azure Kinect, which both use infrared light with a wavelength of 850 nm. In the image sequence shown, the noise varies from no noise (a) to very intense noise (d).



(a) Set-up illustration (b) Set-up photo

Figure 11: Experiment setup to determine the deviation of the detected center point during the rotation of the lattice around its own axis.

The average error of the lattice detection by Optitrack was specified by Optitrack's Motive software as 0.7 mm.

C. Rotational Robustness

With this experiment, we tried to determine the robustness of our method with respect to the angle between the line of sight of the camera and the normal of the lattice. It is to be expected that at grazing angles (i.e., angle between line of sight and the lattice normal approaches 90 degrees), our registration procedure will fail.

We used a thin thread to hang the lattice as symmetrically as possible between two tripods, leaving one one degree of freedom (see Figure 11). In the experiment, the lattice then slowly rotated around its y-axis in the range $[-90, 90]$ degrees. Assuming the rotation axis accurately passes through the lattice's real center, the position of the detected lattice center is not expected to change regardless of the orientation of the lattice.

After recording the lattice at a distance of about 1.5 m, we obtained an average deviation from the mean center along the x-axis of 0.9 mm (SD: 1.0 mm), along the y-axis of 0.4 mm (SD: 0.3 mm), and along the z-axis of 1.8 mm (SD: 1.3 mm) over a range of -57.0 deg to 59.7 deg (see Fig. 12) — at larger angles the lattice was no longer detected. As

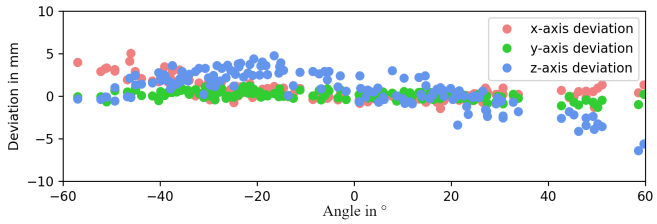


Figure 12: Deviation of the center point during rotation around the y-axis of a lattice, which was clamped between two tripods with sewing threads.

Figure 13: Scenario C for runtime measurement and precision and recall estimation.

expected, the deviation along the y-axis (upward axis) was very small. The slightly higher deviation along the z-axis compared to the x-axis could also be due, at least in part, to the expected error of the Azure Kinect camera depth values (which mainly affect the z-value). Also, our lattice has a thickness of 4 mm and was built with an accuracy of only about 1-2 mm.

D. Runtime Performance

We expect the runtime of the lattice detection to depend on the distance between the lattice and the sensor since fewer points of the point cloud have to be processed if the lattice is further away. Therefore, we created a recording in which we moved the lattice back and forth at a distance of about 0.9 m to 3.9 m (see Fig. 13).

On average, we observed an average runtime of 19.2 ms on a single core of an AMD Ryzen 9 3900X processor for the lattice detection per processed frame (SD 6.4 ms) in Scenario C. For only 198 of considered 4022 frames, the lattice detection took more than 33.33 ms (4.9%), while the maximum runtime in this scenario was 60.5 ms. As expected, the runtime clearly depends on the distance of the lattice to the sensor, see Fig. 14.

For completeness, we have also given the results of our runtime measurements for Scenario A and Scenario B in Table III. There it can be seen that the average runtimes of 19.8 ms and 24.5 ms are quite similar for other scenarios as well. However, a few frames of both recordings were

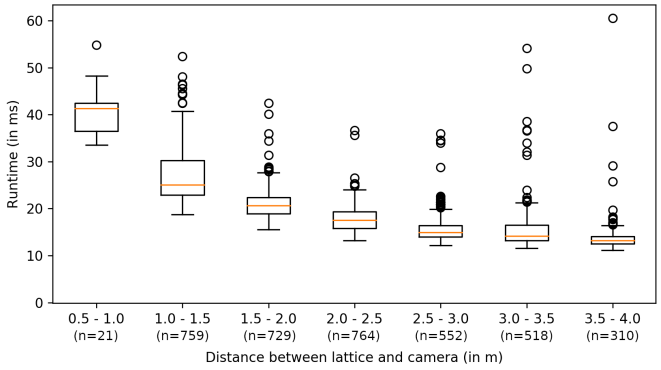


Figure 14: Dependence of the runtime on distance between lattice and sensor in scenario C only (considering frames where a lattice was detected).

Table III: Runtimes of our lattice detection algorithm in different scenarios (without parallelization).

Scenario A (n = 7034)	**Mean**	**SD**	**Max**
Candidate search	8.9 ms	15.9 ms[1]	230.1 ms[1]
Candidate processing	10.9 ms	13.06 ms[1]	344.2 ms[1]
Plausible candidates	0.79	0.56	5
Total	**19.8 ms**	**23.9 ms**[1]	457.3 ms[1]
Scenario B (n= 3931)	**Mean**	**SD**	**Max**
Candidate search	12.8 ms	8.4 ms[1]	67.8 ms[1]
Candidate processing	11.7 ms	7.75 ms[1]	62.84 ms[1]
Plausible candidates	1.18	0.73	6
Total	**24.5 ms**	**14.2 ms**[1]	116.1 ms[1]
Scenario C (n= 4022)	**Mean**	**SD**	**Max**
Candidate search	9.3 ms	2.1 ms	19.4 ms
Candidate processing	9.9 ms	5.0 ms	53.6 ms
Plausible candidates	1.20	0.48	4
Total	**19.2 ms**	**6.4 ms**	60.5 ms

1: Scenarios A and B where affected by occasional pulsating noise due to interference between Optitrack and Azure Kinect and are to be understood as extreme cases with regard to performance.

affected by pulsating noise due to interference between the Azure Kinect and Optitrack (see Fig. 10). As a result, many lattice candidates were detected in these frames, and although they were correctly rejected, the maximum runtime was abnormally high.

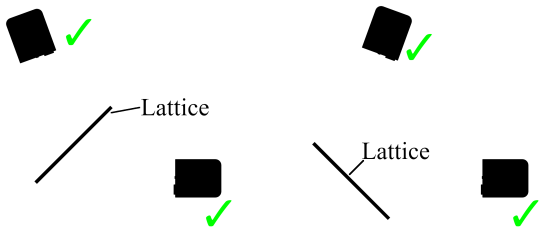
E. Precision and Recall

Another quality metric for registration methods is the robustness of the detection of the target object in the images, which can be measured by the well-known classification scores precision (defined as PPV $= \frac{\text{TP}}{\text{TP+FP}}$) and recall (defined as TPR $= \frac{\text{TP}}{\text{TP+FN}}$). In our case, precision gives the percentage of lattices detections that were correct, while recall describes the percentage of correct lattices that our

Table IV: Precision and recall of the lattice detection algorithm in Scenario C.

Scenario	**n**	**TP**	**FP**	**TN**	**FN**	**PPV**	**TPR**
A	6049	4281	21	0[1]	1747[1]	0.995	0.7[1]
B	3593	2141	19	0[1]	1433[1]	0.991	0.60[1]
C	4022	3629	24	0	369	0.992	0.91

1: Note that in scenario A the lattice sometimes leaves the camera's view frustum completely, while in scenario B the lattice is only partially visible in at least multiple frames. Since our lattice recognizer sometimes also detected edge cases, we could not clearly determine at what point an unrecognized lattice is counted as TN or FN. Therefore, we counted frames in which no lattice was detected as FN even if the lattice was only partially visible or not visible at all. So, the recall in these two scenarios is probably significantly higher in reality.



(a) Sensor's angle > 90 deg (b) Sensor's angle < 90 deg

Figure 15: Since the lattice can be detected from both sides, our registration method can be used in most applications despite the limited detection at some angles (see Section IV-C). As a rough rule of thumb, if angle between the lines-of-sight of two cameras is greater than 90 deg, the lattice should be positioned such that they see different sides of the lattice (a), while for camera angles less than 90 deg, the lattice should be held so that both sensors see the same side of the lattice (b).

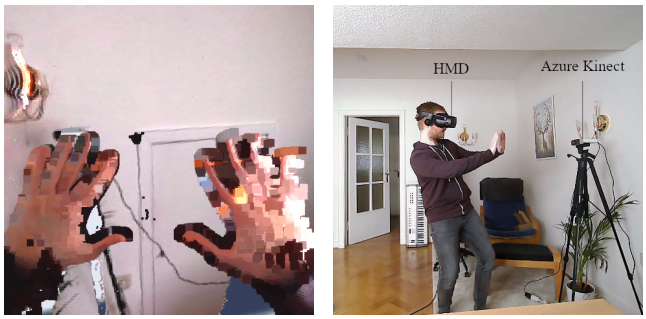
algorithm detected among all the visible, actual targets (i.e., in the camera's field of view). In Scenario C, we ensured that the lattice was fully visible in the camera's field of view at all times; hence, there are no true negatives in this case.

Our results can be found in Table IV.

F. Limitations

We observed that with the Azure Kinect, some of the holes of the lattice are occasionally invisible in the original depth image. In our experiments, we found that this seems to depend on the background behind the lattice (e.g., surface normal and reflectivity) and primarily occurs when the distance to the background is greater than the Azure Kinect's working range (we used the *NFOV unbinned* mode which has a working range of 0.5–3.86 m)[4]. We suspect this may be related to a filter in the Azure Kinect or Azure Kinect

[4]Given by the Microsoft Azure Kinect specifications: https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification



(a) View through the HMD (b) Physical setup

Figure 16: Rendering of the point clouds of two Kinects registered into a virtual scene seen from a first-person view in an HMD (a) and from a third perspective (b).

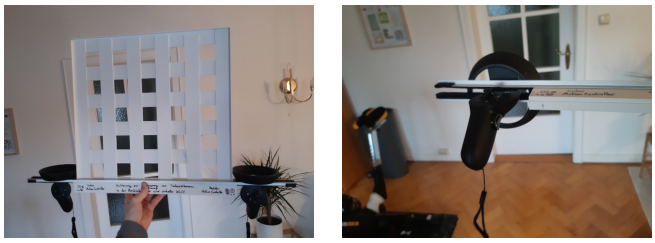


Figure 17: The lattice with the motion controllers of the HP Reverb G2 attached using a bracket allowing for precise registration of depth sensors with the virtual scene.

SDK that appears to bridge areas between invalid pixels. In scenario B, this effect likely had a significant impact on the number of false-negative detections and, consequently, the recall, due to the distant background (partially $> 6\,m$), while the effect was nearly negligible in scenarios A and C. Although this effect did not affect the registration success nor accuracy in our scenarios, there might be special application areas where registration with our method could be difficult when the Azure Kinect is operated outside its working range.

As shown in the experiment regarding rotational stability, our method can detect the lattice stable only if the angle between the lattice normal and the camera viewing direction is smaller than about 55 deg. However, since our method is robust against viewing the lattice from the front or from the back, this is only a minor limitation for most applications, as Figure 15 shows.

Finally, probably the most obvious and very minor limitation is the fact that our lattice needs to be held by one or two hands at only one side while registration is performed.

V. Virtual World Registration

Some applications require not only registration between depth sensors, but also registration of these depth sensors with a virtual world. For example, in VR applications, it might be required to stream a point cloud into the virtual world for the use as a user's avatar or for real objects to cast shadows by virtual lights (see Fig. 16).

To perform such a registration, we need to find correspondences between virtual world space and one of the depth sensors. To do so, we designed a physical bracket to which our lattice and two motion controllers can be attached. This allows to move the lattice and the motion controllers simultaneously while maintaining a fixed relative distance between them (see Fig. 17). We manually measured the transformations from the lattice to both motion controllers, $T_{\text{RightController}}$ and $T_{\text{LeftController}}$, resp. Given those, we can derive the position of the controllers in camera space. Since their position in virtual world space is also given (by the VR system's tracking), their positions in both reference frames can be collected simultaneously over multiple frames, which can then be used as corresponding point pairs. Thus, the virtual world and the camera reference frames can be registered using the SVD-based transformation estimation of the PCL [27].

Instead of manually measuring the transformation between the motion controllers and the lattice, it is also possible to estimate the transform using the rigid motions of both controller and lattice. This problem is known as hand-eye calibration and simply requires recording of multiple poses of the motion controllers and the lattice. For a feasibility check, we performed this calibration offline using the method of [29]. Instead of only calculating the transformation between controller and lattice, we could also directly compute the world to camera space transformation. As the norm of this transformation typically is larger, though, a small error in the lattice orientation leads to an overall higher calibration error.

VI. Source Code

We provide two implementations of our registration procedure. To register arbitrary depth sensors with each other, we provide a new small C++ library which can be found at https://gitlab.informatik.uni-bremen.de/cgvr_public/lattice_registration_library. Second, we provide a slightly larger Unreal Engine 4 project that allows for registration of multiple Microsoft Azure Kinects with each other as well as with a virtual world: https://gitlab.informatik.uni-bremen.de/cgvr_public/lattice_based_registration_ue4

VII. Conclusion

We presented a novel approach for the registration (extrinsic calibration) of depth sensors based exclusively on depth data. As a registration target, we designed a lattice-like board with regularly-spaced holes which are visible in the depth image. More importantly, we developed an algorithm that can detect such boards reliably and accurately in depth images and is very easy to implement.

In our test scenarios, which we performed using a Microsoft Azure Kinect under real-world conditions, we

achieved a precision of more than 0.99 with our lattice detection algorithm. At the same time, the lattice detection has an average running time of roughly 20 ms on a single core of an AMD Ryzen 9 3900X per frame. Using the features of the detected lattices for registration, we measured an average registration error of 1.6 mm between two point clouds in the middle of the registration volume at a capture distance of approximately 2 m from the sensors.

We provide an open, small C++ library that can be used to register any kind of depth sensors. Furthermore, we make an Unreal Engine 4 project available that is capable of registering depth sensors with a virtual world; it works exemplary with the Microsoft Azure Kinect, but can be adapted easily to other depth cameras as well.

In future work, our method could be made even more accurate by integrating the work of Deng et al. [30], which is able compensate slight distortions of the depth camera within a larger volume. Additionally, one could implement the optimization presented by Beck et al. [12] for better registration results with hardware which is not synchronized in time. Finally, it could be useful to extend the method to optionally detect the lattice in an IR or color image as well. This might improve the accuracy a bit more, e.g. if one wants to register depth-only sensors or LiDAR sensors, which do not provide an infrared or color image, with RGB-D sensors, which have such an additional image.

Statements and Declarations

Partial financial support was received from BMBF grant 13GW0264D. The authors have no financial or proprietary interests in any material discussed in this article.

References

[1] C. G. Harris, M. Stephens *et al.*, "A combined corner and edge detector." in *Alvey vision conference*, vol. 15, no. 50. Citeseer, 1988, pp. 10–5244.

[2] M. Rufli, D. Scaramuzza, and R. Siegwart, "Automatic detection of checkerboards on blurred and distorted images," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3121–3126.

[3] A. Duda and U. Frese, "Accurate detection and localization of checkerboard corners for calibration," in *29th British Machine Vision Conference. British Machine Vision Conference (BMVC-29), September 3-6, Newcastle, United Kingdom*, 2018.

[4] F. Reyes-Aviles, P. Fleck, D. Schmalstieg, and C. Arth, "Improving rgb image consistency for depth-camera," *Journal of WSCG*, vol. 28, pp. 105–113, 01 2020.

[5] X. Song, J. Zheng, F. Zhong, and X. Qin, "Modeling deviations of rgb-d cameras for accurate depth map and color image registration," *Multimedia Tools and Applications*, vol. 77, 06 2018.

[6] A. Mühlenbrock, R. Fischer, R. Weller, and G. Zachmann, "Fast and robust registration of multiple depth-sensors and virtual worlds," in *2021 International Conference on Cyberworlds (CW)*, 2021, pp. 41–48.

[7] R. Macknojia, A. Chavez-Aragon, P. Payeur, and R. Laganiere, "Calibration of a network of kinect sensors for robotic inspection over a large workspace," 01 2013, pp. 184–190.

[8] C. Chen, B. Yang, S. Song, M. Tian, J. Li, W. Dai, and L. Fang, "Calibrate multiple consumer rgb-d cameras for low-cost and efficient 3d indoor mapping," *Remote Sensing*, vol. 10, p. 328, 02 2018.

[9] W. Darwish, W. Li, S. Tang, B. Wu, and W. Chen, "A robust calibration method for consumer grade rgb-d sensors for precise indoor reconstruction," *IEEE Access*, vol. 7, pp. 8824–8833, 2019.

[10] R. Avetisyan, M. Willert, S. Ohl, and O. Staadt, "Calibration of depth camera arrays," 06 2014.

[11] S. Beck and B. Froehlich, "Volumetric calibration and registration of multiple rgbd-sensors into a joint coordinate system," in *2015 IEEE Symposium on 3D User Interfaces (3DUI)*, 2015, pp. 89–96.

[12] ——, "Sweeping-based volumetric calibration and registration of multiple rgbd-sensors for 3d capturing systems," in *2017 IEEE Virtual Reality (VR)*, 2017, pp. 167–176.

[13] D. Herrera C., J. Kannala, and J. Heikkilä, "Joint depth and color camera calibration with distortion correction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 10, pp. 2058–2064, 2012.

[14] T. Deng, J. Cai, T.-J. Cham, and J. Zheng, "Multiple consumer-grade depth camera registration using everyday objects," *Image and Vision Computing*, vol. 62, pp. 1 – 7, 2017.

[15] A. Papachristou, N. Zioulis, D. Zarpalas, and P. Daras, "Markerless structure-based multi-sensor calibration for free viewpoint video capture," 01 2018.

[16] H. Liu, D. Qu, F. Xu, F. Zou, J. Song, and K. Jia, "Approach for accurate calibration of rgb-d cameras using spheres," *Opt. Express*, vol. 28, no. 13, pp. 19 058–19 073, Jun 2020. [Online]. Available: http://www.osapublishing.org/oe/abstract.cfm?URI=oe-28-13-19058

[17] A. N. Staranowicz, G. R. Brown, F. Morbidi, and G.-L. Mariottini, "Practical and accurate calibration of rgb-d cameras using spheres," *Computer Vision and Image Understanding*, vol. 137, pp. 102–114, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314215000703

[18] P.-C. Su, J. Shen, W. Xu, S.-C. S. Cheung, and Y. Luo, "A fast and robust extrinsic calibration for rgb-d camera networks," *Sensors*, vol. 18, no. 1, 2018. [Online]. Available: https://www.mdpi.com/1424-8220/18/1/235

[19] P. Li, R. Wang, Y. Wang, and W. Tao, "Evaluation of the icp algorithm in 3d point cloud registration," *IEEE Access*, vol. 8, pp. 68 030–68 048, 2020.

[20] H. Maron, N. Dym, I. Kezurer, S. Kovalsky, and Y. Lipman, "Point registration via efficient convex relaxation," *ACM Transactions on Graphics*, vol. 35, pp. 1–12, 07 2016.

[21] J. Han, P. Yin, Y. Q. He, and F. Gu, "Enhanced icp for the registration of large-scale 3d environment models: An experimental study," *Sensors*, vol. 16, p. 228, 02 2016.

[22] E. Altantsetseg, O. Khorloo, and K. Konno, "Rigid registration of noisy point clouds based on higher-dimensional error metrics," *The Visual Computer*, vol. 34, no. 6, pp. 1021–1030, Jun 2018. [Online]. Available: https://doi.org/10.1007/s00371-018-1534-6

[23] V. Morell-Gimenez, M. Saval-Calvo, J. Azorin-Lopez, J. Garcia-Rodriguez, M. Cazorla, S. Orts-Escolano, and A. Fuster-Guillo, "A comparative study of registration methods for rgb-d video of static scenes," *Sensors*, vol. 14, no. 5, pp. 8547–8576, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/5/8547

[24] Y. Song, W. Shen, and K. Peng, "A novel partial point cloud registration method based on graph attention network," *The Visual Computer*, Feb 2022. [Online]. Available: https://doi.org/10.1007/s00371-021-02391-0

[25] S. Huang, Z. Gojcic, M. M. Usvyatsov, A. Wieser, and K. Schindler, "Predator: Registration of 3d point clouds with low overlap," *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4265–4274, 2021.

[26] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, 1981.

[27] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[28] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[29] F. Furrer, M. Fehr, T. Novkovic, H. Sommer, I. Gilitschenski, and R. Siegwart, *Evaluation of Combined Time-Offset Estimation and Hand-Eye Calibration on Robotic Datasets*. Cham: Springer International Publishing, 2017.

[30] T. Deng, J.-C. Bazin, T. Martin, C. Kuster, J. Cai, T. Popa, and M. Gross, "Registration of multiple rgbd cameras via local rigid transformations," in *2014 IEEE International Conference on Multimedia and Expo (ICME)*, 2014, pp. 1–6.