Fast and Robust Registration of multiple Depth-Sensors and Virtual Worlds

Andre Mühlenbrock, Roland Fischer, René Weller and Gabriel Zachmann
Computer Graphics and Virtual Reality
University of Bremen
Bremen, Germany
Email: muehlenb@uni-bremen.de

***Abstract*—The precise registration between multiple depth sensors is a crucial prerequisite for many applications. Previous techniques frequently rely on RGB or IR images and checkerboard targets for feature detection. However, this prohibits the usage for use-cases where neither is available or where IR and depth images have different projections.**

Therefore, we present a novel registration approach that uses depth data exclusively for feature detection, making it more universally applicable while still achieving robust and precise results. We propose a combination of a custom 3D registration target — a lattice with regularly-spaced holes — and a feature detection algorithm that is able to reliably extract the lattice and its features from noisy depth images. In addition, we have integrated the registration procedure to a publicly available Unreal Engine 4 plugin that allows multiple point clouds captured by several depth cameras to be registered in a virtual environment. Despite the rather noisy depth images, we are able to quickly obtain a robust registration that yields an average deviation of 3.8 mm to 4.4 mm in our test scenarios.

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

I. Introduction

RGB-D and range cameras are widely used throughout the research community and for industrial applications, thanks to the multitude of available and affordable products. These cameras are often utilized for telepresence and robotic applications for tasks such as 3D reconstruction, SLAM, or object recognition. In case of large spaces or to avoid occlusions, often multiple cameras are used. In these cases, the sensors have to be calibrated intrinsically (individually) and extrinsically (registered to each other).

The classical approach for calibration and registration of multiple cameras is based on feature detection on planar checkerboards because they provide a good target in color and IR images. In a first step, the checkerboard itself has to be detected; then, the inner corners can be extracted using well-known corner detectors (e.g. [1]), to serve as point correspondences. Using these correspondences, a rigid transformation between the different sensors is calculated. This approach was continuously improved upon (e.g. [2], [3]) and generally yields robust and precise results.

However, this method is not always applicable, e.g., if no IR or color images are available. Also, Reyes-Aviles et al. [4] reported that, depending on the camera model, the IR images and the corresponding depth images may have different projections, which leads to errors if the IR images are used for registration. Performing the registration directly on depth images or point clouds, on the other hand, leads to the issue of inherently noisy depth data, which makes accurate feature detection difficult [5].

In this paper, we present a registration procedure that is exclusively based on 3D point clouds, therefore avoiding the need for color or IR images, yet:

- achieving precise registration results,
- being very robust in real use cases,
- taking only a few seconds of recorded images,
- working independently of scene brightness and
- easy to implement.

We achieve this by using a specially designed 3D checkerboard-like registration target which can be easily replicated, and a pipeline designed for the precise recognition of the target's reference points in noisy depth images. Furthermore, we have made our source code publicly available.

II. Related Work

As we already mentioned, usually, calibration and registration of depth sensors is done via the accompanying IR or RGB sensor images: Macknojia et al. [6] synchronously captured a checkerboard in a Kinect's color- and IR images for extrinsic calibration between the RGB and depth sensor. Extrinsic calibration (or registration) between multiple Kinect cameras was done similarly using the respective IR images. Chen et al. [7] captured a checkerboard in the color and IR images for homography-based calibration, while Darwish et al. [8] tracked two orthogonal checkerboards and aimed at improving the depth accuracy.

Some authors, e.g., [9], [10] and [11], also applied the checkerboard approach but added external optical tracking systems for depth correction and registration of multiple cameras into a joint coordinate system, respectively. Although, in this case the cameras' viewing frusta do not have to overlap, the requirement for a tracking system is a severe limitation.

Herrera et al. [12] proposed a calibration approach that works directly in the depth image by detecting the outer checkerboard corners using a time-consuming manual selection. Kreylos [13] exchanged the usual black and white

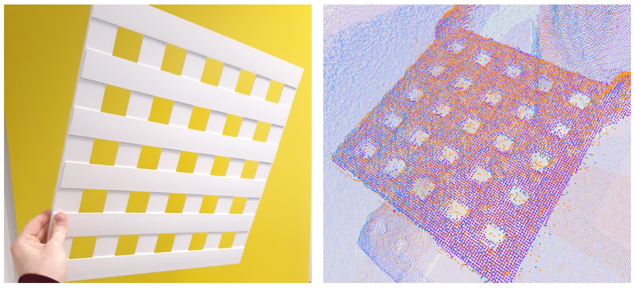

(a) Photo (b) Registered point cloud

Figure 1: Our 3D registration target: a lattice-like board with 25 holes.

checkerboard with a semitransparent that can be observed directly in the depth image. Similarly, Reyes-Aviles et al. [4] proposed to use a 3D checkerboard as calibration target and a process, involving normal estimation, edge detection and thresholding, to detect it in the depth image. The registration process proposed by Song et al. [5] relies on a special checkerboard with regularly-spaced hollow squares. Depth deviations are handled by a model-based approach that considers the center points of the holes. Some works like [14], [15] exchanged checkerboard-like registration targets with static marker-free 3D objects with known or previously scanned geometry, e.g. a stack of boxes, which can be detected in depth maps or point clouds.

III. Our Approach

A. Overview

We have constructed a lattice-like 3D target (see Fig. 1) and developed a lattice detection algorithm that allows us to quickly and easily perform registration of multiple depth sensors. To do this, the lattice is briefly moved in the field of view of the depth sensors, while our lattice detection algorithm detects up to 25 point correspondences per frame for all depth sensors. Using those point correspondences, we determine a rigid transformation matrix that describes the transformation of the sensors with respect to each other.

The main challenge of the classical camera registration procedure is the correct recognition of the board's features in the image. Since we use only depth data instead of an RGB image, we cannot simply reuse the image-based detection algorithms. In the following, we present a novel approach that is fast and easy to implement yet achieves 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 lattice center and lattice axes.
4) Outlier removal among hole centers.
5) Correspondence mapping.
6) Correspondence rejection.
7) SVD-based transformation estimation.

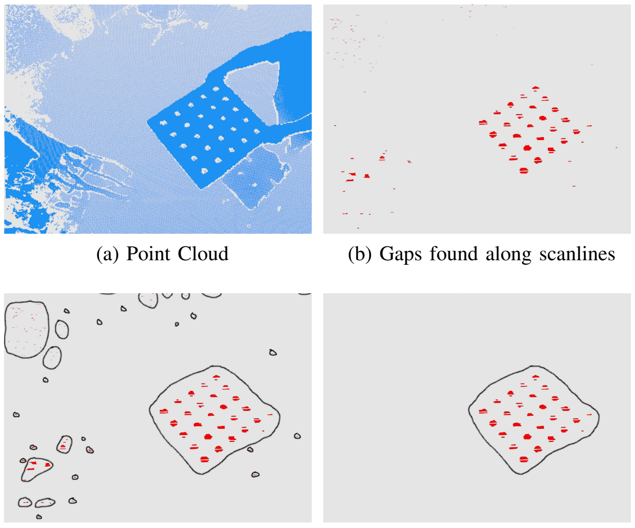

(a) Point Cloud (b) Gaps found along scanlines (c) Clusters of gap segments (d) Plausible clusters

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

As a target, we have designed a lattice-like board consisting of 12 bars of size 44 cm x 4 cm x 0.2 cm, which are available in a typical DIY store (see Fig. 1). By leaving 4 cm space between the bars in vertical and horizontal direction, 25 holes of size 4 cm x 4 cm are formed, which can be recognized by depth sensors.

Note that all parts of our recognition algorithm directly work on an array of 3D coordinates (basically, a point cloud) that is in scanline order of the original depth image, and not on the original depth image itself. In this way, parameters can be defined directly in centimeters and are independent of the lattice distance to the sensor.

B. Lattice Candidates

In the first step, we search for gaps along the scanlines of the original depth image which correspond to the regular geometry of the lattice (see Fig. 2). We do this by segmenting individual scanlines based on the Euclidean distance of individual points. Segments that are at most as long as the diagonal of a hole and are surrounded by two scanlines of plausible length are identified as gap segments. In the next step, we cluster all gap segments based on their proximity to each other. Using PCA, we calculate eigenvectors and eigenvalues for each cluster. For the sake of simplicity, we use only the midpoints of the gap segments for proximity calculation and the PCA. 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.

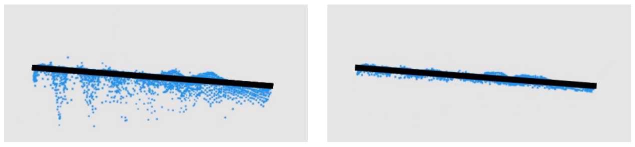


Figure 3: Side view of the lattice in the point cloud (blue) before and after filtering noise (e.g. due to the flying pixel effect). This is done by fitting a plane (black) via RANSAC into these points, then culling points by thresholding.

C. Hole Center Detection

The next step is the identification of the exact hole centers for each lattice candidate. To do so, we first determine all 3D points that potentially belong to the physical lattice based on their proximity to the gap segments. A point is considered to be a lattice point if it is within a certain radius to at least one segment of a gap center (we use $r = 0.16\,\text{m}$ to cover the lattice completely if some holes were missed in the previous step).

To effectively filter out noise that typically occurs with depth sensors (e.g. due to the flying pixel effect), we fit a plane to the point cloud section using RANSAC and define all points closer than a certain threshold[1] to the plane as inliers (see Fig. 3). We store the indices of these inliers 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-B but this time we mark both, inlier segments in which all points are in the inlier set as well as outlier segments. All outlier segments, which are enclosed by inlier segments are assumed to be a part of a hole. Assuming that these segments are horizontally aligned, we now vertically join adjacent outlier segments if they overlap horizontally to obtain a segment for each hole of the lattice. This vertical merging of the horizontally running outlier segments is done efficiently using the union-find structure.

The remaining segments represent the individual holes. However, since the points of these outlier segments do not lie in the plane of the lattice, we consider the directly adjacent inlier points in scanline order, project them onto the plane computed via RANSAC and use their mean value as hole centers.

D. Axes Detection and Outlier Removal

To summarize, 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

[1]In our actual implementation we use a threshold of $1.5\,\text{cm}$ because our lattice is somewhat bendable.

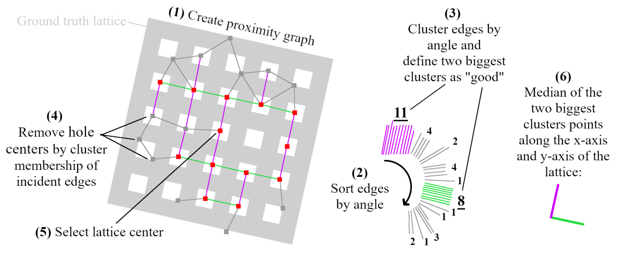

Ground truth lattice
(1) Create proximity graph
(3) Cluster edges by angle and define two biggest clusters as "good"
(6) Median of the two biggest clusters points along the x-axis and y-axis of the lattice:
(4) Remove hole centers by cluster membership of incident edges
(2) Sort edges by angle
(5) Select lattice center
11
4
2
4
1
8
1
1
2
1
3

Figure 4: 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.

want to match several point clouds, i.e. we want to use the potential hole centers as correspondence points.

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:

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

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. 4).

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

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

3) Cluster these vectors based on their angle α. As can be seen in Fig. 4, this results in two very large clusters (color coded by the violet and green edges) as well as several other very small clusters, assuming that it is indeed a lattice. 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:

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

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. 4).
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.

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.

E. *Point Correspondences*

In principle, 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 becomes. Therefore, it is desirable to use all the hole centers instead of just the center one. By using all hole centers, we get up to 25 times more point correspondences over a larger space in the same time.

However, since we still lack the information whether the lattice in the original depth image is seen from the front or from the back, we cannot yet establish a clear mapping 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 $z_A \cdot z_B$. 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.[2]

Now we generate ideal positions of the 25 hole centers using h^* and the axes of the lattice. We assign IDs from 0 to 24 to these hole centers along the axes. However, in order to also take into account small distortions of the point cloud caused by the depth sensor, we now search for the respective nearest hole center actually found. If there is none within a small radius around the ideal hole center, this hole center will not be considered as a correspondence point. We now define hole centers with the same ID in both sensor A and sensor B as points correspondences.

F. *Correspondence Rejection and Registration*

Up to this point, we have found point correspondences for which the 3D coordinates in camera space of several cameras 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 degraded, we filter the point correspondences using the RANSAC-based correspondence rejection of the Point Cloud Library [16]. Finally, with the remaining point correspondences, we perform the registration using the SVD-based transformation estimation.

IV. Results

In this section, we present our results regarding the accuracy of our registration into a ground truth coordinate system (see Section IV-A), the stability of lattice detection when the lattice is rotating (see Section IV-B), the runtime of lattice detection (see Section IV-C), and further observations, such as the precision and recall of lattice detection (see Section IV-D).

A. *Ground-Truth Evaluation*

To investigate the accuracy of our registration, 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 (see Fig. 5). However, our heuristic generally detected these hole centers as incorrect hole centers, resulting in no noticeable effect on the detection of the lattice.

[2]In fact, we could likewise correctly assign the orientation vectors of the lattice obtained via our heuristic to the x-axis and y-axis via the rough registration without taking into account the positions of the hands. However, this would have two disadvantages: On the one hand, for a rough estimation of the transformation we could only use the center of the lattice and not the center shifted in the x-direction, which leads to the need for more frames and movement along different axes for a stable result. On the other hand, when registering the depth cameras to the world coordinate system (see Section V), we would need to know the direction of the motion controllers relative to the lattice center anyway.

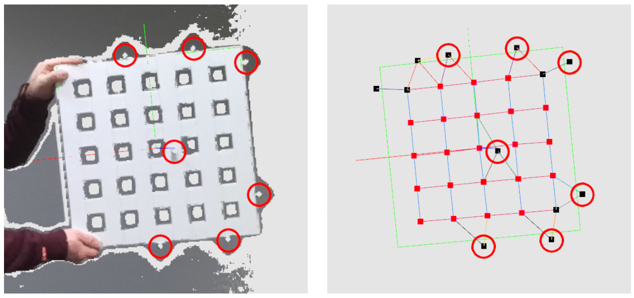

(a) Point cloud (b) After applying our heuristic

Figure 5: Seven Optitrack markers (highlighted in red) attached to the lattice to record ground truth data. Left: the recorded point cloud; right: the generated proximity graph in which the clusters of edges are color coded. The Optitrack markers generate false hole center candidates, but these are consistently filtered by our heuristics and do not affect the lattice detection.

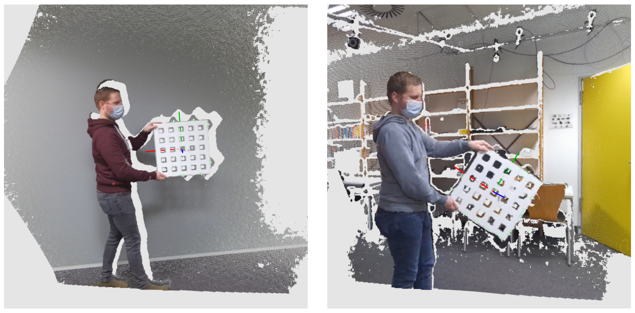

(a) Scenario A (b) Scenario B

Figure 6: Two scenarios used for ground truth evaluation, from the perspective of the depth sensor.

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. 9). We ran Optitrack at 30fps (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, this lattice candidates were successfully discarded by our detection algorithm and does not seem to have influence on our results.

We conducted the evaluation in front of two different backgrounds (see Fig. 6), hereafter also called scenarios, while moving the lattice slowly. In scenario B, the distance between the sensor and the lattice was between 1.0 m and 1.95 m, with the center of the lattice residing in a volume of roughly 0.6 m³.[3] In scenario A, the distance between the sensor and the lattice was between 0.85 m and 2.05 m, with the center of the lattice residing inside a volume of roughly 1.2 m³.

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

Table I: 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).

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 I). This deviation is much smaller than the systematic error of up to 11 mm + 0.1% of the distance (as given by the Azure Kinect's technical specifications) and a random error of up to 17 mm [17]. The average error of the ground truth data was specified by Optitrack's Motive software as 0.7 mm. The expected error due to the lack of synchronization between Optitrack and the Azure Kinect is 1.56 mm.[4]

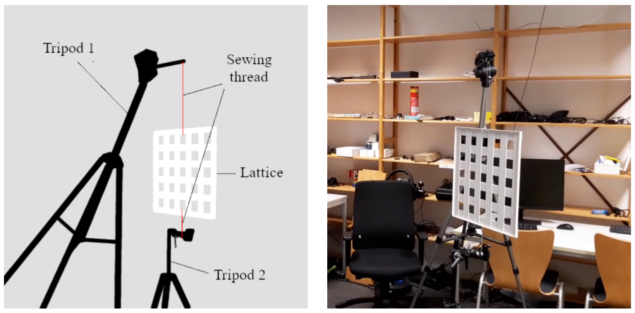
Overall, the error of our registration method is well within the accuracy of the depth sensor we use. We hypothesize that our registration method is capable of achieving even better precisions with more accurate depth cameras.

B. Rotational stability

In this experiment, we tried to determine the robustness of our method w.r.t. the angle between camera's line of sight an the lattice's normal. Using a thin thread, we clamped the lattice as symmetrically as possible between two tripods and let it rotate slowly around the y-axis of the lattice (see Fig. 7) in the range from $-90°$ to $90°$. This was recorded with an Azure Kinect. Assuming that the lattice is clamped exactly, the expectation is that the detected lattice center location will not change regardless of the orientation of the lattice.

After recording the lattice at a distance of approximately 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 from $-57.0°$ to $59.7°$ (see Fig. 8) — at larger angles, the lattice was no longer detected. The deviation along the y-axis (up-axis), was very small,

[4]This arises from an average grid movement speed of 18.8 cm/s in scenario A and 30 fps, which yields 188 mm/s $\times$ 0.0083 s.



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

Figure 7: Experiment setup to determine the deviation of the detected center point during the rotation of the lattice around its own axis.

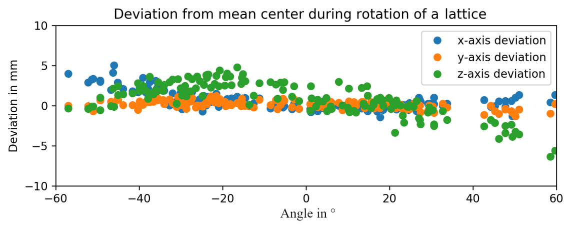


Figure 8: Deviation of the center point during rotation around the y-axis of a lattice, which was clamped between two tripods with sewing threads.

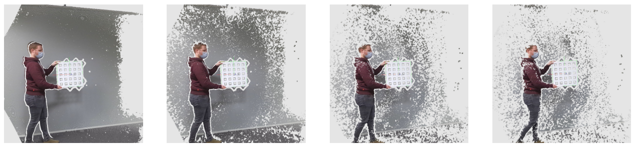
as expected. The slightly higher deviation along the z-axis compared to the x-axis could also be at least partly due to the expected error of Azure Kinect camera's depth values (which mainly influence the z-value) [17]. Furthermore, our lattice has a thickness of 4 mm and was built with an accuracy of only about 1-2 mm.

C. Runtime

Since pulsating noise in scenario A and scenario B strongly affected the runtime of our lattice detection algorithm in individual frames due to interference between Optitrack and the Microsoft Azure Kinect (see Fig. 9), we performed a recording in a third scenario C, in which no additional tracking system was present that could have caused interference (see Fig. 10). Nevertheless, we have presented the average running times of the algorithm for all three scenarios in Table II.

While Scenario B has a more complex background, which leads to a slightly higher mean runtime of 24.5 ms (SD: 14.2 ms) compared to Scenario A with 19.8 ms (SD: 23.9 ms), the impact of the occasional pulsating noise was particularly greater in scenario A due to the planar background. As expected, in scenario C the standard deviation (SD) and the maximum runtime (Max) are significantly lower because there is no pulsating noise due to the interference of the depth sensor with a tracking system.

In addition, we observed that the runtime of the algorithm decreased the farther the lattice was from the camera (see



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

Figure 9: 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).

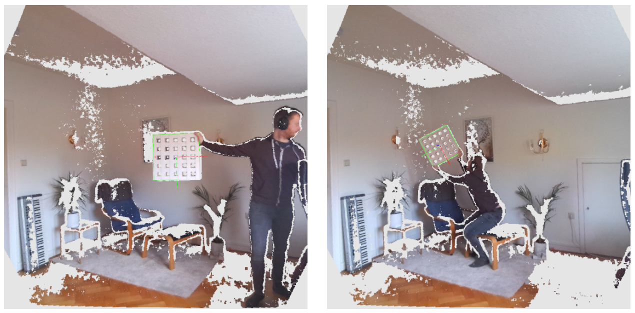


Figure 10: Scenario C which was not affected by occasional pulsating noise by avoiding the additional tracking system.

Table II: 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	230.1 ms
Candidate processing	10.9 ms	13.06 ms	344.2 ms
Plausible candidates	0.79	0.56	5
Total	**19.8 ms**	**23.9 ms**	457.3 ms
Scenario B (n= 3931)	**Mean**	**SD**	**Max**
Candidate search	12.8 ms	8.4 ms	67.8 ms
Candidate processing	11.7 ms	7.75 ms	62.84 ms
Plausible candidates	1.18	0.73	6
Total	**24.5 ms**	**14.2 ms**	116.1 ms
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

Note: Scenarios A and B where affected by occasional pulsating noise due to interference between Optitrack (which was used as ground truth in our experiments) and Azure Kinect and are to be understood as extreme cases with regard to performance.

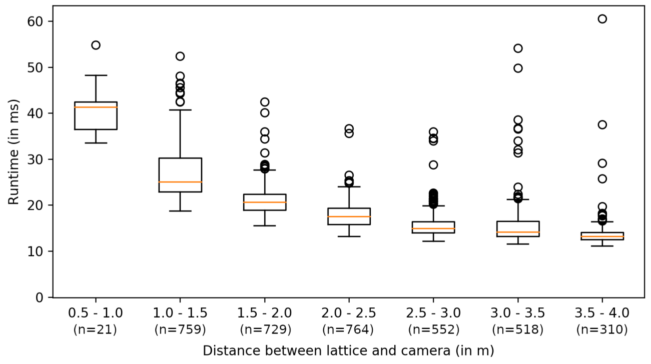


Figure 11: Dependence of the runtime on distance between lattice and sensor in scenario C only (considering frames where a lattice was detected).

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

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

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 some 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, especially in scenario A.

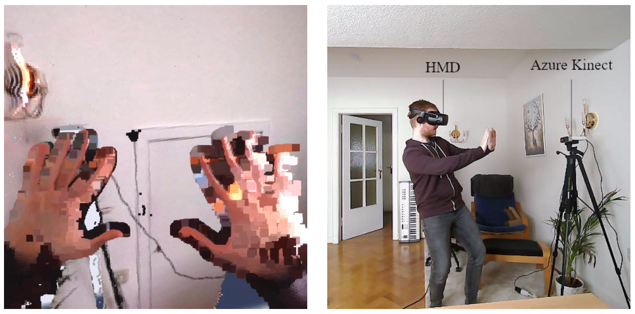
Fig. 11). This can be simply explained by the fact that the further away the lattice is, the fewer points of the point cloud have to be processed.

Note that our algorithm is currently not parallelized and was running on a single core of an AMD Ryzen 9 3900X processor in all of our evaluations.

D. Further observations

Furthermore, we calculated the precision (PPV $= \frac{\text{TP}}{\text{TP+FP}}$) and recall (TPR $= \frac{\text{TP}}{\text{TP+FN}}$) for all three scenarios (see Table III).

We have observed that with the Azure Kinect, individual holes of the lattice are occasionally invisible in the original depth image. We noticed in our experiments that this depends on the background behind the lattice (e.g. normal of the surface and reflectivity) and occurs primarily only when the distance to the background is greater than the operating range of the Azure Kinect (in the *NFOV unbinned* mode we used, the operating range is 0.5 m – 3.86 m)[17]. We suspect that this might be related to a filter of the Azure Kinect or the Azure Kinect SDK, which seems to bridge areas between



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

Figure 12: 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).

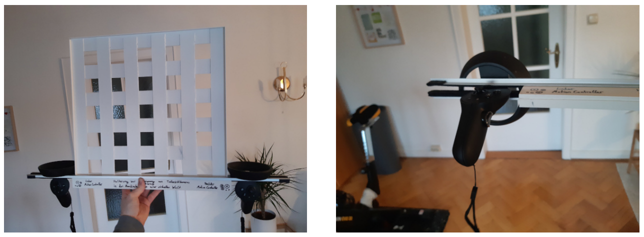


Figure 13: The lattice with the motion controllers of the HP Reverb G2 attached using a bracket allowing for precise registration of depth sensors with the virtual scene.

invalid pixels. In scenario B, this effect presumably had a significant impact on the number of false negative detections as well as on recall due to the distant background (partly $> 6\,m$), while the effect was almost negligible in scenarios A and C. However, this effect had no impact on the success of fast and accurate calibration in any of the scenarios tested.

V. Unreal Engine 4 Plugin

We implemented our registration process in a publicly available Unreal Engine 4 plugin that can be used for arbitrary depth sensors. In addition, we have created a sample Unreal project for use with the Microsoft Azure Kinect, in which any number of Azure Kinects can be registered with each other. Furthermore, the depth sensors can be registered into the virtual world (see Fig. 12).

To enable the depth sensors to also be registered in a virtual world, we designed a 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. 13). We determined the fixed offsets to the motion controllers in the lattice coordinate system by measuring the distance of the lattice center to the origin of the motion controller along each axis manually. Using the pose of the lattice returned by our detection algorithm and the fixed offsets to both motion controllers, we obtain the position of the motion controllers in the depth sensor coordinate system. By recording these positions with simultaneous positions of the motion controllers in the coordinate system of the virtual world, we can register all depth sensors into the world coordinate system of the virtual world using SVD-based transformation estimation again.

The Unreal Engine 4 plugin and the sample project can be found at: https://gitlab.informatik.uni-bremen.de/cgvr_public/lattice_based_registration

VI. Conclusion

We have presented a novel approach for the registration 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. Our algorithm is able to detect the board reliably in depth images of real-world applications and is very easy to implement. In our three test scenarios, which we recorded using a Microsoft Azure Kinect, we are able to achieve a precision of more than 0.99 with our lattice detection algorithm, while the algorithm only needs an average runtime of roughly 20 ms on a single core of an AMD Ryzen 9 3900X. The average deviation of the lattice from the ground truth measurement (obtained using Optitrack) was 3.8 mm to 4.4 mm. In addition, we integrated our registration procedure to a published Unreal Engine 4 plugin, which also allows multiple depth sensors to be registered into a virtual world.

In the future, we plan to improve our registration method with additional techniques: We are considering using multiple rigid transforms as proposed by Deng et al. [18] to better compensate slight distortions due to the depth camera in a larger volume. In addition, we can use depth cameras that do not support hardware synchronization, e.g., with the optimization presented by Beck et al. [11]. Finally, we are thinking about developing a simple detection of the lattice in the color image to optionally register RGB sensors with the depth sensor as well.

Acknowledgment

This work was partially supported by BMBF grant 13GW0264D.

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] 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.

[7] 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.

[8] 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.

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

[10] 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.

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

[12] 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.

[13] O. Kreylos, "Kinect camera calibration," 2013, http://doc-ok.org/?p=289.

[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] 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.

[17] Microsoft Corporation, "Azure kinect dk hardware specifications," 2020, https://docs.microsoft.com/en-us/azure/kinect-dk/hardware-specification.

[18] 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.