Vive for Robotics: Rapid Robot Cell Calibration

Morten Andre Astad[1], Mathias Hauan Arbo[1], Esten Ingar Grøtli[2], and Jan Tommy Gravdahl[1]

***Abstract*— The use of an HTC Vive; a virtual reality (VR) system and its innovative tracking technology is explored in order to create an approximate one-to-one mapping to the virtual representation of a robot cell. The mapping is found by performing hand-eye calibration, establishing a spatial relationship between the inertial frames of the robot cell and the tracking system. One of the main contributions of this article is the development of an open-source Robotic Operating System (ROS) package for VR devices such as the Vive. The package includes automated calibration procedures such that the devices gives a centimetric measurement error in the robot cell. The calibrated system has problems that are related to specific issues of the tracking technology. This article outlines these issues, their cause, and potential fixes in a concise manner. A simple assembly scenario is presented, where the outline of objects in the robot cell are defined by registering points with a Vive tracker. The potential use cases of the calibrated system are limited by its accuracy, and depends on the required tolerances.**

***Index Terms*— Virtual reality (VR), Lighthouse tracking, Robotic Operating System (ROS), Industrial robots, Robot cell calibration, Hand-eye calibration, Manufacturing**

I. INTRODUCTION

Industrial robots are often too inflexible for the current market demands of small- and medium-sized enterprises (SMEs). As a part of the EU funded research project SMErobotics, [1] suggested that one of the main challenges preventing the adoption of industrial robots in SMEs is that current robot programming techniques are not suitable for frequent changes of often highly customized products manufactured in small batches.

This article explores the use of an HTC Vive, a virtual reality (VR) system codeveloped by Valve and HTC, for rapid robot cell calibration. By creating an approximate one-to-one mapping to the virtual representation of a robot cell, one can quickly place objects and obstacles as necessary. The innovative technology that allows for positioning in a room-scaled environment is called lighthouse tracking. This technology is able to track the user's hands, head, or other objects in real-time through tracked devices. The devices have sub-millimeter precision within an area whose diagonal is up to 5 meters in length.

The outside-in tracking system of the Vive sweeps the room horizontally and vertically with 850 nm infrared (IR) laser lines at a fixed frequency, from one or multiple stationary base stations in the room. Light sensors on the tracked devices are hit periodically by the laser lines, and their position and orientation (pose) is reconstructed by solving a problem that is similar to the Perspective-n-Point (PnP) problem [2]. The tracked devices also contain an inertial measurement unit (IMU) that provides faster updates than the ones from the lighthouse tracking. This gives the devices low frequency measurements of absolute position and orientation, and faster updates of their relative motion.

In [3], the accuracy and viability of a Vive are described for scientific research. They concluded that the Vive was unsuited for scientific experiments if loss of tracking was likely, as a large systematic error was observed, which changes whenever the tracking was completely lost and regained. This error makes it difficult to establish a calibration procedure that aligns the real and virtual coordinate space. However, it is possible to avoid this error by taking measures against its cause, which will be presented together with a calibration procedure that does not depend on the choice of coordinates.

The main contributions of this article are: elaborating on the tracking issues related to the Vive, development of open-source software for calibration with respect to an industrial robot, development of open-source software for defining points, planes and boxes in a virtual environment, as well as presenting an assembly use-case example.

The article is split into 3 main parts. Section II describes how the Vive was integrated in a software environment for robots, and includes the theory and methods that was used to automatically calibrate the tracking system. Section III gives an overview on how the tracking system was set up, calibrated and evaluated in a robot cell, and also presents a simple assembly scenario that was defined with collidable objects using a tracked device. Section IV discusses specific issues of the lighthouse tracking, as they are prevalent, and performing reliable measurements without understanding these issues and how to fix them can be quite challenging.

This article is based on the master thesis of Astad [4], which we refer the reader to for more in-depth implementation details.

II. SYSTEM INTEGRATION AND THEORY

A. Hand-Eye Calibration

The standard hand-eye calibration problem was formulated in [5], where the problem was stated as an equation of homogeneous transformations:

$$\mathbf{AX} = \mathbf{XB}, \qquad \mathbf{A}, \mathbf{B}, \mathbf{X} \in \mathrm{SE}(3) \tag{1}$$

[1]Morten Andre Astad (morten_astad@hotmail.com), Mathias Hauan Arbo, and Jan Tommy Gravdahl are with the Department of Engineering Cybernetics, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

[2]Esten Ingar Grøtli is with SINTEF Digital, Trondheim, Norway

The work presented in this paper was partially funded by the Research Council of Norway through the projects SFI Manufacturing (contract number: 237900) and Dynamic Robot Interaction and Motion Compensation (contract number: 270941).

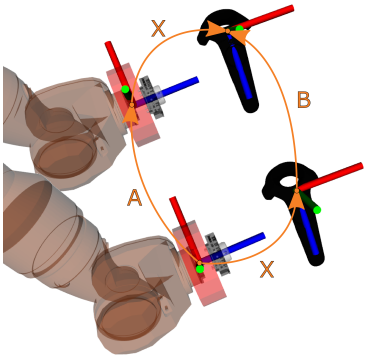

Fig. 1. Geometric interpretation of the $\mathbf{AX} = \mathbf{XB}$ problem, showing two different robot states.

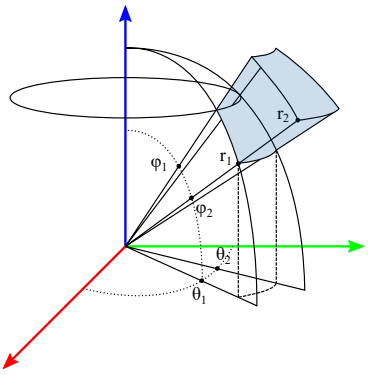

Fig. 2. Spherical volume element that is defined by the spherical coordinates $(r \in [r_1,\, r_2]\,,\, \theta \in [\theta_1,\, \theta_2]\,,\, \phi \in [\phi_1,\, \phi_2])$ in a right-handed coordinate system.

where $\mathbf{A}$ represents a change in the robot's tool pose, $\mathbf{B}$ is the resulting sensor displacement from changing the tool pose, and $\mathbf{X}$ is an unknown transformation relating the tool frame to the sensor frame. The unknown transformation $\mathbf{X}$ is constant under the assumption that the sensor is rigidly attached to the robot and its tool frame. Fig. 1 shows a geometric interpretation of this problem.

The hand-eye calibration was solved by utilising the closed form solution in [6]. The input to this method is $N \in \mathbb{N}_{>1}$ measured pairs of transformations $(\mathbf{A}_i,\, \mathbf{B}_i) \in \mathrm{SE}(3)$, as defined by the deviation between $N+1$ consecutive samples of tool $\{t\}$ and sensor $\{s\}$ poses:

$$\mathbf{A}_i = \mathbf{T}_{t_i}^{-1}\, \mathbf{T}_{t_{i+1}}, \qquad \mathbf{T}_{t_i},\, \mathbf{T}_{t_{i+1}} \in \mathrm{SE}(3) \tag{2a}$$

$$\mathbf{B}_i = \mathbf{T}_{s_i}^{-1}\, \mathbf{T}_{s_{i+1}}, \qquad \mathbf{T}_{s_i},\, \mathbf{T}_{s_{i+1}} \in \mathrm{SE}(3) \tag{2b}$$

The method solves the rotational part first before using it to solve the translational part, which propagates an error from rotation to translation. An extra optimization step was added to reduce this error, where the following cost function was minimized with the closed form solution to the hand-eye calibration as an initial guess for the solver:

$$\min_{\mathbf{X} \in \mathrm{SE}(3)} \sum_{i=1}^{N} \log\left((\mathbf{A}_i \mathbf{X})^{-1} \mathbf{X} \mathbf{B}_i \right) \tag{3}$$

Here, N is the number of measured pairs $(\mathbf{A}_i,\, \mathbf{B}_i)$, $(\,\cdot\,)^{-1}$ is the $\mathrm{SE}(3)$ group inverse and $\log(\,\cdot\,)$ is the matrix logarithm, which maps elements in the group of rigid transformations $\mathrm{SE}(3)$ into elements of its tangent space $se(3)$. This step significantly reduced the error when using fewer than 10 measured pairs.

B. Generating Sample Poses

For automatic calibration, tool poses are generated from a spherical volume element for sampling the necessary poses with a robot. Their position is selected at random within the volume element, and their orientation is computed from the normal vector at this position, as from the surface of a sphere that is centered at the robot's base. The positions can take on any value within the spherical volume element that is parameterized as shown in Fig. 2. Each tool pose can then be represented as a homogeneous transformation from the robot's base frame $\{b\}$, to tool frame $\{t\}$:

$$\mathbf{T}_b^t = \begin{bmatrix} \mathbf{R}_{z,\,\theta} \mathbf{R}_{x,\,\phi} & \mathbf{R}_{z,\,\theta} \mathbf{R}_{x,\,\phi} \begin{bmatrix} 0 & 0 & r \end{bmatrix}^T \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{4}$$

where $\mathbf{R}_{x,\,\phi}$ and $\mathbf{R}_{z,\,\theta}$ are basic rotations about the x-axis and z-axis by an angle ϕ and θ respectively, and r is the radius of a sphere. [7] showed that the rotation between consecutive sensor poses, and the translation between consecutive tool poses should be maximized and minimized respectively; in order to improve the accuracy of the hand-eye calibration. Therefore, the translations and rotations are generated from two sets of parameters in smaller and larger ranges about the same point. This method of randomly generating tool poses for sampling within a range is flexible and generalizes well for different setups. It was tested with the floor and gantry mounted KUKA KR 16-2 robots presented in this article, and a Universal Robots UR10 robot that was mounted on a freestanding frame.

C. Robot Operating System Package

Robotic Operating System (ROS) is an open-source middleware solution for robotics. At its core it offers a communication system, which provides a message passing interface between distributed nodes in a network. One of the main contributions of the work that is presented in this article was the development of a ROS package named vive_rrcc. This package makes VR hardware such as the Vive available in a ROS environment by utilising SteamVR through the OpenVR SDK by Valve. The package is open-source under the MIT License and is freely available from https://github.com/mortaas/vive_rrcc.

The package exposes the pose of each tracked device as a coordinate frame with respect to an inertial tracking frame. These frames and their relationships are maintained in a distributed tree structure that is buffered in time with the tf2 transform library for ROS [8]. This library allows the user to transform vectors, quaternions, poses and so forth between any two frames in the tree structure. It also acts as a buffer for the poses of the tracked devices, which are available in

any frame of the transform tree, and to all nodes in the ROS environment.

Other features of the package includes controller inputs, haptic feedback (vibration), linear and angular velocities (twists) and 3D visualization of the tracked devices, and a standard interface to interact with and calibrate the node in realtime. The LibSurvive library [9] was also implemented as an alternative to SteamVR and OpenVR. Unlike OpenVR, this library allows for access to the low-level components of the lighthouse tracking and supports the use of different community implemented tracking algorithms.

The generated sample poses can be automatically realized on the robot system. Robot trajectories are planned from the generated tool poses with the MoveIt library [10], a motion planning framework that is integrated with ROS. Tool and sensor poses are then sampled from the transform tree between each executed trajectory, and the program waits a predefined time before sampling, in order for the robot and tracking dynamics to settle. This hardware-agnostic approach can be used on any ROS-Industrial supported robot with a MoveIt package.

The sampled poses of a tracked device are subject to noise in the sub-millimetric and sub-degree range, which may introduce a small error in the calibrated system. This noise was reduced by almost two orders of magnitude by using the quaternion averaging method in [11] with 120 samples.

D. Calibrating the Tracking System

One-to-one mapping from a robot cell to its virtual representation is established by finding a spatial relationship between the inertial frames of the robot cell and tracking system. This relationship was found by employing hand-eye calibration, in order to estimate the rigid transformation $\hat{\mathbf{X}}$ between a tracked device that is firmly attached to the robot and an arbitrary tool frame of the robot.

A transformation between the inertial frames of robot cell $\{rc\}$ and tracking system $\{vr\}$ is computed for each of the measured sample poses, with the estimated solution $\hat{\mathbf{X}}$ from solving the hand-eye problem:

$$\mathbf{T}_{rc}^{vr} = (\mathbf{T}_{t}^{rc})^{-1} \hat{\mathbf{X}}\, \mathbf{T}_{s}^{vr}, \qquad \hat{\mathbf{X}} = \hat{\mathbf{T}}_{t}^{s} \tag{5}$$

where $\mathbf{T}_{rc}^{vr}$ is the transformation from the inertial frame of the robot cell to the Vive's inertial frame according to the calibration, $\mathbf{T}_{vr}^{s}$ is the transformation to the tracked device frame relative to the Vive's inertial frame, and $(\mathbf{T}_{t}^{rc})^{-1}$ is the tool frame relative to the robot cell.

The computed transformations are then averaged in the same way as the sampled poses, in order to reduce a small nonlinear and spatially dependent error of the tracking system. The resulting average is used to calibrate the system by automatically updating the corresponding relationship in the transform tree. Figure 5 shows the virtual representation of the robot cell after performing this calibration.

E. Rapid Obstacle Placement

A simple framework was created in order to define collidable objects with geometric primitives, such as boxes, spheres, cylinders, and cones in the coordinates of the robot cell using a tracked device. This section presents the box case.

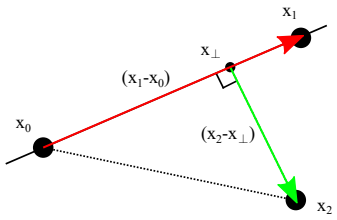

Fig. 3. Defining a unique plane from three 3D points.

A box can be uniquely defined in 3D space from four points x_0, x_1, x_2, $x_3 \in \mathbb{R}^3$. The first three points are used to define the orientation of the box from basis vectors:

$$\mathbf{b}_x = \frac{x_1 - x_0}{\|x_1 - x_0\|}, \tag{6a}$$

$$\mathbf{b}_y = \frac{x_2 - x_\perp}{\|x_2 - x_\perp\|}, \tag{6b}$$

$$\mathbf{b}_z = \mathbf{b}_x \times \mathbf{b}_y \tag{6c}$$

where $x_\perp$ is defined such that the basis vectors are orthogonal, as shown in Fig. 3:

$$x_\perp = x_0 - \left[(x_0 - x_2)^T \mathbf{b}_x\right] \mathbf{b}_x. \tag{7}$$

These basis vectors can be used to form the rotation matrix:

$$\mathbf{R} = \begin{bmatrix} \mathbf{b}_x & \mathbf{b}_y & \mathbf{b}_z \end{bmatrix} \in \mathrm{SO}(3). \tag{8}$$

It is now possible to define the length L, width W and height H of the box by introducing the fourth point x_3:

$$L = \|x_1 - x_0\|, \quad W = \|x_1 - x_\perp\|, \quad H = (x_3 - x_\perp)^T \mathbf{b}_z. \tag{9}$$

The translation to the center of the box is then given by:

$$\mathbf{t} = x_0 + {}^1\!/_2 \left(L\, \mathbf{b}_x + W\, \mathbf{b}_y + H\, \mathbf{b}_z\right) \in \mathbb{R}^3 \tag{10}$$

This method of defining a box is intuitive, and the framework visualizes a point, line, plane and box in that order for each point that is defined by the user. The recorded collidable objects are saved as Simulation Description Format (SDF) files, a human readable XML format that describes objects and environments for robot simulators, visualization, and control.

The planar part of this method could be used as an alternative to the automated alignment and correction method in [12]. Where three Vive Trackers affixed to a frame is utilised in order to align the virtual space with the physical ground, and fix the tilt that was reported in [3]. Similarly, it is possible to define the ground plane with the method that is described here, and fix the issue using only one tracked device and defining three points instead.

III. EXPERIMENTAL SETUP

A. Vive-Robot Cell Setup

The system was tested on the robot cell with an approximate size of $6\,m \times 4\,m \times 4\,m$, shown in Fig. 4. The robot

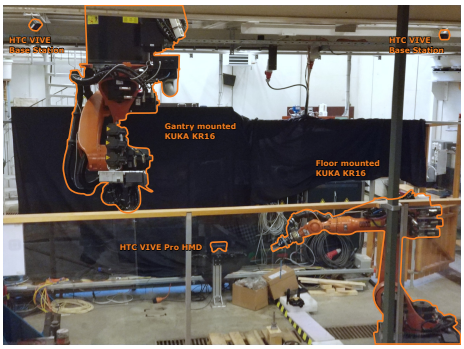

Fig. 4. Setup of the Vive's tracking system in a robot cell.

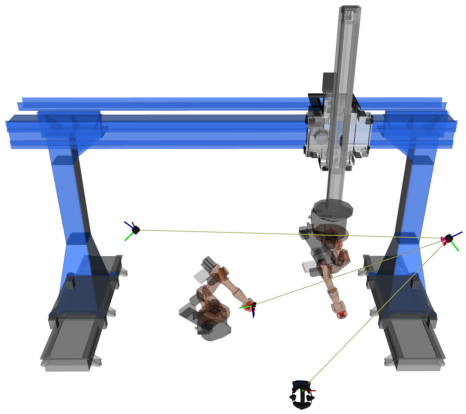

Fig. 5. The virtual representation of the robot cell after calibration, as visualized in RViz (ROS 3D Robot Visualizer).

cell consists of two KUKA KR 16-2 industrial robots, one of which is mounted on the floor, and the other is mounted on a gantry system from Güdel. Both base stations are mounted below H-beams in the roof structure, following the recommendations from HTC. The base stations have a field of view of 120°, leaving 30-45° for adjustments. The base stations should be placed such that their view of each other and the robot cell is unobstructed. It is also important that their field of view overlaps as much as possible within the intended tracking volume. A Vive Pro Starter Kit with first-generation base stations was used, which provides updates at a rate of 220-370 Hz depending on the type of tracked device [13]. The measurements are sub-sampled in the ROS package at a rate of 120 Hz by default.

B. Calibration of the Mapping

A tracked device was firmly attached to the gripper of the floor robot, and 51 tool poses was generated in the range $(r \in [1.4,\, 1.6]\,,\, \theta \in [-5\pi/16,\, -3\pi/16]\,,\, \phi \in [\pi/8,\, 3\pi/16])$ for positions and range $(r \in [1.4,\, 1.6]\,,\, \theta \in [0,\, -\pi/2]\,,\, \phi \in [5\pi/16,\, 11\pi/16])$ for orientations, with origin at the robot base. This range corresponds to sampling sensor poses in close proximity to the tool pose of the floor mounted robot in Fig. 5. The synthetic tests in [6] suggests that this number of samples should result in a solution that is close to convergence, which is further refined by solving the minimization problem in (3). A wait time of 20 seconds was used in order for the tracking dynamics to settle within a reasonable range of a few millimeters. A visualization of the robot cell after calibration is given in Fig. 5.

C. Testing the Calibrated System

A volume of $1.0\,m \times 3.0\,m \times 1.0\,m$ in the center of the robot cell was sampled with the calibrated system at $4 \times 7 \times 3$ distributed points, in order to show an indication of its accuracy. The sampling was performed with the same setup as the calibration, with a tracked device firmly attached to the robot. Each sample was compared with an ideal sensor pose, which was computed with the forward kinematics of the robot and the estimated solution $\hat{\mathbf{X}}$ from solving the hand-eye problem:

$$\tilde{\mathbf{T}}^{s}_{rc} = \mathbf{T}^{vr}_{rc}\,\mathbf{T}^{s}_{vr} - (\mathbf{T}^{rc}_{t})^{-1}\hat{\mathbf{X}} \tag{11}$$

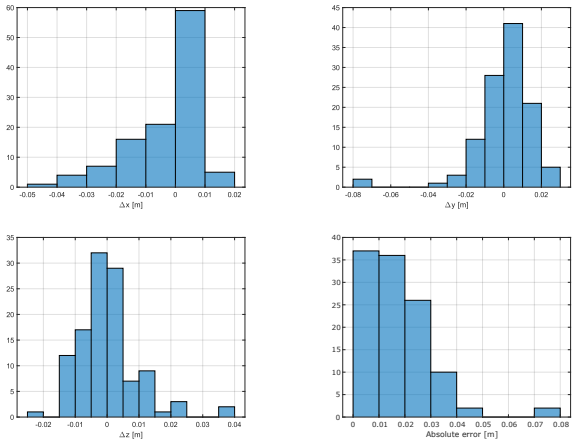

Fig. 6. Frequency distribution of the measurement error along the x-, y-, z-axis and their absolute values with the robot's forward kinematics as a ground truth. The y-axis shows the number of samples in each bin.

with transformations defined as in 5.

The sampling procedure was run twice in order to validate the runs against each other, and resulted in the frequency distribution of Fig. 6. This distribution has a small negative bias along the x-axis, where the resulting mean along the axes was found to be $[-0.003363,\, 0.0002183,\, 0.0003022]$ meters with standard deviation $[0.0119,\, 0.01468,\, 0.009321]$ meters. The cause of this bias is not well-understood, but the tracking dynamics seems to imply that it is caused by a larger drift along the x-axis. These results indicates a measurement error in the centimetric range, and the maximum absolute error was $8\,cm$.

D. Assembly Use-Case

The calibrated system was tested on a simple assembly scenario that is shown in Fig. 7, where a mock-up for inserting a rotor into a motor housing was mapped with a tracked device. A simple tool was made from a tracked device with a spike probe attached to it, as shown in Fig. 8. The tool was used to register the necessary points to define collisions in the assembly scenario, by pointing the spike at points and pressing a button. Figure 9 shows the virtual representation

Fig. 7. Assembly scenario.

Fig. 8. Simple tool based on a Vive Tracker with a 15 cm long and 1 cm thick spike probe screwed into its quarter inch UNC threaded camera mount.

of the scenario, which was meticulously defined in about five minutes. The dimensions of the Euro-pallet in this figure was defined with centimetric accuracy and a millimetric deviation between similar parts.

IV. DISCUSSION

The calibrated system has a few problems that are related to specific issues of the lighthouse tracking. These issues are mentioned in literature about the Vive and its tracking system, but the documentation about troubleshooting and correcting them is sparse. This article hopes to rectify some of this sparsity by outlining the issues, their cause, and potential fixes in a concise manner.

1) Prioritizing Inertial Measurements: [14] showed that the Vive's tracking algorithm gives greater weight to its inertial measurements in order generate smooth trajectories for VR applications. This weighting can clearly be seen in Fig. 10, where a tracked device was moved quickly between two points. The error is converging a lot faster when the tracked device is moving, which then slowly approaches its final value with an overdamped (second order) impulse

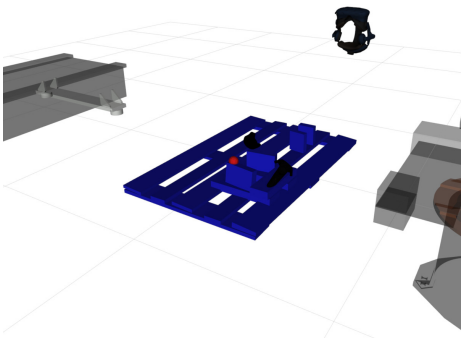

Fig. 9. Virtual representation of the assembly scenario, as visualized in RViz. The red sphere represents the tip of the spike probe.

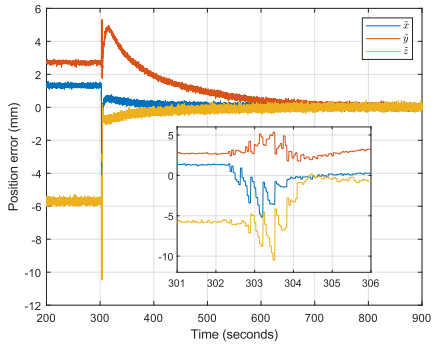

Fig. 10. Error response from moving a tracked device quickly between two points. The move starts at 302, 4 seconds, and it lasts approximately 2 seconds.

response. Although the wait time that was used for calibration is low relative to the tracking dynamics and causes a millimetric error, the hand-eye calibration is robust.

Convergence can take as long as 500 seconds, and causes an error in the millimetric range when measuring the position of a device before it has converged. The only known way of avoiding this error is the use of a third party tracking algorithm, which is exactly what Borges et al. introduced in their article [14]. An open-source back-end such as LibSurvive has to be used in place of SteamVR, in order to use a third-party tracking algorithm.

2) Tilted Reference Frame: [3] reported that poses measured with the Vive are provided in a reference frame that is tilted with respect to the physical ground plane. This issue is caused by the fact that the reference frame is aligned with the gravity vector, which is estimated with an IMU in the tracked device. The tilted reference frame is a symptom of sensor bias in the IMU [15], and the solution is to either return the device or recalibrate the IMU [16]. Access to the calibration tools requires a SteamVR tracking license. The tilted floor is not an issue for the calibration procedure that is presented in this article, as it relies on an external calibration that does not depend on the choice of coordinates.

3) Switching Bias: [3] also observed a large systematic error that switched its value whenever tracking was completely lost and regained. According to the inventor of lighthouse tracking, Alan Yates (Reddit username: vk2zay), the error occurs whenever the base stations disagree with each other by a large amount [17]. The error is caused by a recalibration of the base stations, in order to reduce the discrepancy between them. This recalibration shows up as a bootstrapping of one of the base stations in the web console of SteamVR. The resulting error is nonlinear in Euclidean space, as the pose of the base stations is changed internally in the tracking system. It was noted that this change occurs instantaneously for all devices, and a monitor was added to the ROS node in order to warn the user if a recalibration has occurred.

This recalibration can be avoided for the most part, by always keeping a tracked device in a location that is visible to both base stations without risk of concealment. The head-mounted display (HMD) in Fig. 4 was used for this purpose.

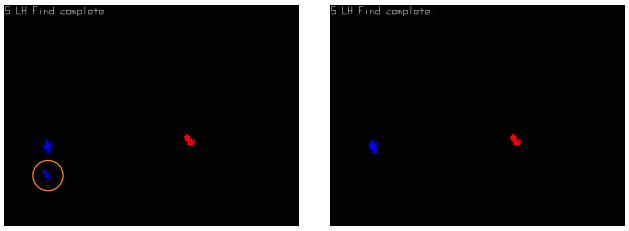

Fig. 11. The LibSurvive calibration tool before (left) and after (right) the reflections in the orange ring was removed with a black piece of fabric, where the points should be clustered together.

4) Tracking Jitter: The final and perhaps most common issue is tracking wobble and jitter, which is caused by reflections from the environment. Robot cells, for instance, are often enclosed by a fencing system with clear polycarbonate for safety reasons. This enclosure causes reflections that may have a negative impact on the robustness of the tracking.

The LibSurvive library is able to visualize the reflections in a 2D map through its calibration tool, as shown in Fig. 11. This figure shows the situation before and after the black piece of fabric in Fig. 4 was added to the robot cell. Removing the reflections resulted in more robust tracking for SteamVR, and the LibSurvive tracking would not work properly without this change.

V. CONCLUSION

In this article a set of ROS packages are presented that were developed for calibration of an HTC Vive with respect to a robot cell, rapid placement of collidable objects and identifying relevant points in the robot cell. The procedure is hardware-agnostic and can run on any system with ROS-Industrial and a MoveIt plugin. The calibration was tested using a KUKA KR 16-2 and an assembly use-case was presented. The calibration showed a centimetric positioning error, which suggests that the system can be used for crude positioning of objects, such as for collision avoidance or high-level planning, or if the underlying control algorithm exhibits sufficient robustness to positioning errors. The article outlines some of the most common tracking issues, and gives a description of how to resolve them.

ACKNOWLEDGMENT

The authors would like to thank the LibSurvive community for their troubleshooting and advice, and Rune Sandøy of SINTEF Manufacturing and Mjøs Metallvarefabrikk for the assembly usecase.

REFERENCES

[1] A. Perzylo et al. “SMErobotics: Smart Robots for Flexible Manufacturing.” In: *IEEE Robotics Automation Magazine* 26.1 (Mar. 2019), pp. 78–90.

[2] M. A. Fischler and R. C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography.” In: *Commun. ACM* 24.6 (June 1981), pp. 381–395.

[3] D. C. Niehorster, L. Li, and M. Lappe. “The Accuracy and Precision of Position and Orientation Tracking in the HTC Vive Virtual Reality System for Scientific Research.” In: *i-Perception* 8.3 (2017).

[4] M. A. Astad. “Vive for Robotics.” Master Thesis. Department of Engineering Cybernetics, NTNU, 2019.

[5] Y. C. Shiu and S. Ahmad. “Calibration of wrist-mounted robotic sensors by solving homogeneous transform equations of the form AX=XB.” In: *IEEE Transactions on Robotics and Automation* 5.1 (Feb. 1989), pp. 16–29.

[6] F. C. Park and B. J. Martin. “Robot sensor calibration: solving AX=XB on the Euclidean group.” In: *IEEE Transactions on Robotics and Automation* 10.5 (Oct. 1994), pp. 717–721.

[7] R. Y. Tsai and R. K. Lenz. “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration.” In: *IEEE Transactions on Robotics and Automation* 5.3 (June 1989), pp. 345–358.

[8] T. Foote. “tf: The transform library.” In: *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*. Apr. 2013, pp. 1–6.

[9] C. Lohr et al. *Lightweight HTC Vive Library*. [Online; accessed 27-May-2019]. 2019. URL: https://github.com/cnlohr/libsurvive.

[10] S. Chitta, I. Sucan, and S. Cousins. “MoveIt! [ROS Topics].” In: *IEEE Robotics Automation Magazine* 19.1 (Mar. 2012), pp. 18–19.

[11] F. L. Markley, Y. Cheng, J. L. Crassidis, and Y. Oshman. “Averaging Quaternions.” In: *Journal of Guidance, Control, and Dynamics* 30.4 (2007), pp. 1193–1197.

[12] A. Peer, P. Ullich, and K. Ponto. “Vive Tracking Alignment and Correction Made Easy.” In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. Mar. 2018, pp. 653–654.

[13] O. Kreylos. “Lighthouse tracking examined.” In: *URL: http://doc-ok.org* (2016).

[14] M. Borges, A. Symington, B. Coltin, T. Smith, and R. Ventura. “HTC Vive: Analysis and Accuracy Improvement.” In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Oct. 2018, pp. 2610–2615.

[15] Steam. *Floor Tilt a Hardware or Software Issue?* [Online; accessed 15-May-2019]. 2016. URL: https://steamcommunity.com/app/358720/discussions/0/353916981477560813/?tscn=1490201536.

[16] Reddit. *Fix for Slanted Floor Issue - IMU Recalibration*. [Online; accessed 15-May-2019]. 2017. URL: https://www.reddit.com/r/Vive/comments/6tzthx/fix_for_slanted_floor_issue_imu_recalibration/.

[17] Reddit. *Controllers jump when changing base station line of sight*. [Online; accessed 13-May-2019]. 2017. URL: https://www.reddit.com/r/Vive/comments/5tafa5/controllers_jump_when_changing_base_station_line/.