Multi-kinect Skeleton Fusion for Enactive Games

Nikolaj Marimo Støvring, Esbern Torgard Kaspersen, Jeppe Milling Korsholm, Yousif Ali Hassan Najim, Soraya Makhlouf, Alireza Khani, and Cumhur Erkut(✉)

Department of Architecture, Design and Media Technology, Aalborg University Copenhagen, Copenhagen, Denmark
{nstavr14,ekaspe14,jkorsh14,ynajim14,smakhl14, akhani14}@student.aau.dk, cer@create.aau.dk

Abstract. We present a procedural method and an implementation of multi-Kinect skeleton fusion on Unity environment. Our method calibrates two Kinects by combining the relative coordinates of a user's torso onto a single coordinate system. The method is tested with a small number of participants in scenarios involving multiple users, results indicate that the method provides an improvement over a single camera, and it is accurate enough for games and entertainment applications. The video demonstration of the system is provided, and future directions to improve accuracy are outlined.

Keywords: Motion tracking · Depth cameras · Sensor fusion · Occlusion

1 Introduction

Tracking of one or multiple human bodies is currently feasible with computer vision [1]. Human posture and gestures are studied and tested within various applications; they can properly represent human kinematics and be used for tracking as well as activity recognition [1].

One way of tracking human is to use active depth cameras, which send out light, and upon processing the feedback, produce depth maps instead of an RGB picture. A good example of such an active camera is the Microsoft Kinect. Created as a controller for the Xbox 360 console, Kinect enables the user to play games by following her body movements. The Kinect-v1 combines a passive RGB camera with an active depth sensor, using a structured light pattern, and is capable of body recognition and skeletal tracking in real time [2].

Because of Kinect's capabilities, availability of development tools, and its consumer price range, many researchers have proposed new application areas using the sensor. For instance, its potential in education has been explored by [3]. However, when used in such multiple-user setups, Kinect suffers from the occlusion problem. A way to overcome the occlusion problem could be to set up multiple cameras, so that they can track objects in a scene from different angles, thereby reducing the possibility of occlusion. Various methods of combining multiple depth cameras have been reviewed in [4]. Most of the methods rely on point-clouds, and require the use of a checkerboard grid or other calibration objects. More recent methods rely on skeleton-based calibration of multiple

A.L. Brooks and E. Brooks (Eds.): ArtsIT/DLI 2016, LNICST 196, pp. 173–180, 2017.
DOI: 10.1007/978-3-319-55834-9_20

cameras; for instance in [5], an absolute orientation algorithm proposed by Umeyama and his colleagues [6] has been combined with multi-frame Kalman filtering to dynamically calibrate multiple depth cameras based on the skeletal data.

In this paper, we show a fast and computationally efficient implementation by imposing a symmetric initial calibration step and neglecting the possible drift. The results may not be as accurate as the needs of therapeutic applications, however they are good enough for entertainment and gaming purposes.

The structure of this paper is as follows. In the next section, we provide some background, specifically on skeleton tracking with the Kinect SDK, tracking multiple people, and occlusion problems. Then we describe the pragmatics of our implementation, followed by the tests we have conducted for evaluating our approach. We present the results of these tests, discuss the outcomes and possible improvements, and conclude the paper, indicating future work in the field.

2 Background

In this section we first review the skeleton tracking with the Kinect SDK. After this, we describe the calibration of multiple depth cameras.

2.1 Skeleton Tracking with the Kinect SDK

The Kinect depth camera, combined with the IR sensor is capable of tracking up to six people at once [7]. Of those, two people can be assigned a skeleton. Kinect SDK assigns 20 points to a tracked person. Every point is mapped to a key joint or position on the human body, effectively creating a skeleton. The Kinect is also capable of assigning a seated skeleton; that is assigning a 10 point skeleton to a seated person. It is important to point out that the Kinect SDK cannot distinguish between the front and back of a person.

The recognition and tracking of skeletons are carried out in Kinect SDK using a dedicated GPU and machine learning techniques. They increase motion tracking accuracy and can be used for advanced applications. An example of an advanced application is the posture recognition system described in [8]. This method assigns vectors to the body joints and checks the current posture against predefined ones.

The position tracking error of Kinect, averaged on the horizontal and vertical angles, is found as 29 mm (see http://chhs.gmu.edu/ccid/upload/s12-Gelman.pdf). The largest observed error was about 40.5 mm at a distance of 0.876 m from the object in question, while the smallest error was around 16.5 mm at a distance of 3.07 m. In its most accurate range (around 2–3 m) Kinect has an average tracking error of 22.4 mm.

2.2 Occlusion and Multi-camera Setups

Depth cameras work with the principle that both the emitted light is reflected from a target. If other objects are in the propagation path of the light, this assertion does not hold, and occlusion occurs. Moreover, the inhomogeneity of the medium or reflecting

surfaces may result in blind spots. In recent years, occlusion and blind spots problems of the depth cameras have been increasingly tackled [9]. To increase the recognition rate of an object from the sensors, four cameras placed 5 m apart are arranged in [9] to surround a specific tracking area. With multiple angles to study the same object, the boundary between the object and the background is cleared, making that object the cameras' focal point. With this setup, it is possible to correct the skeletal data of a self-occluding participant but not when multiple participants are within the tracking area. To fix blind spots with multiple users, additional motion sensors are placed on users to collect rotation data, which is later combined with the data the Kinects receive, to be fused into a skeleton.

Various methods that entirely rely on multiple depth cameras without additional motion sensors have been reviewed in [4], and more recent methods rely on skeleton-based calibration of multiple cameras [5], as mentioned in the Introduction. We proceed with a procedural description of our implementation.

3 Implementation

We use the position of a tracked user, in the form of one or more vectors, to calculate the position of the camera relative to it. As a user is represented by a position in each camera, and assuming that the positions calculated by each camera are fairly close to each other in the real world, we assume that the difference between these positions must be equal to the distance between the cameras. To get these positions, we use the joints and the center of gravity obtained from the Microsoft Kinect SDK, and visualize the coordinate systems of each depth cameras, as well as the reference coordinate system (main camera) of Unity.

In the current implementation, we use Unity 5.2 and its new Network code. This feature allows us to easily set up a scene and server that multiple clients, or Kinects, can connect to and thereby transfer data about the positions of the users over the network. The `Network Manager` component is only placed on one object in the scene; every other object has a Network Identity component. The manager keeps track of all these identities, manages all networking. The Network Manager HUD component handles the interface, and adds buttons to the scene, to start a server, connect to a server as a client, or start the server as a host, which also counts as a client.

The position and orientation of the tracked user are acquired and subsequently used in calibration. The information is received from the Skeleton Stream, which contains the position and orientation of every user that the Kinect tracks. We then check whether or not the movement is mirrored; the z-axis movement registered by Kinect is inherently mirrored, and therefore, we want to mirror it again to make the movement unfold in the natural direction.

Then we start with calibration, if enabled. The first check is for rotation: the script gets the horizontal and vertical rotation seen by each camera in comparison to the default coordinate system of Unity. Then we calculate the quaternion angle between the offset and a unit vector for each camera, converting the resulting angles into a quaternion matrix. The horizontal and vertical matrices are multiplied to yield the rotation matrix.

The reference position vector is then multiplied with the rotational matrix. The new position vector indicates where the cube would have been if the cameras had only been offset in position and had a zero-degree angle between them. In other words, the position vector correlates directly to how the camera is positioned in relation to the main camera of Unity. Finally, we add the positional offset, which aligns the cubes on top of each other.

However, this alignment does not yet guarantee the alignment of the orientation of the user. For this, we consider shoulder and hip center joints from the skeleton data. We first invert their z-coordinate, as discussed above. To find the horizontal orientation, the position of the left shoulder is subtracted from that of the right shoulder, the difference is projected onto a unit vector. The vertical orientation component is calculated similarly, but uses center-shoulder and center-hip joints. Both angles are applied to the aligned cube, forcing it to face the same direction as the real-life user.

The video demonstration of the system is provided at https://vimeo.com/156941392 (password: ArtsIT16-AAU-CPH). Figure 1 below presents an instant where a single camera would provide an occluded image, but two cameras within our implementation alleviates this problem.

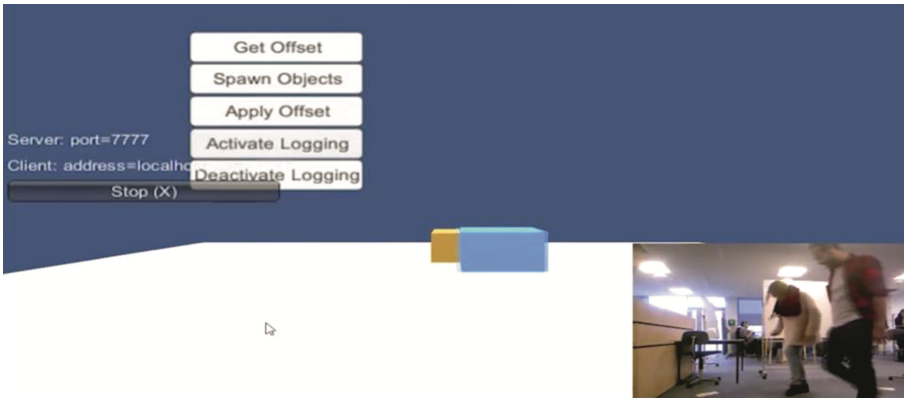

Fig. 1. An example snapshot from the video where the cameras were placed at the edges of the scene view. A single camera would provide an occluded image, but two cameras keep track of two people, represented by the blue and yellow cuboids. (Color figure online)

4 Evaluation

In order to evaluate the effectiveness of the Positional Vector Calibration Setup (PVC), we made an evaluation to compare two conditions; the PVC setup and a single camera setup. The variables of our tests were the error rate in tracking, and accuracy, measured in the offset between a participants' actual and detected position. The users were on a known position (marker) but we did not measure precisely their position. So the error of the fusion is sensitive to this position error.

We have controlled the independent variables: the number of people and their positions in the test scene. We have started with a pilot and tested different scenarios. The setups are illustrated on Fig. 2. All authors, but the last one have participated to tests. Since this is a physical and not perceptual or assessment test, the familiarity of the participants with the system is not considered important.

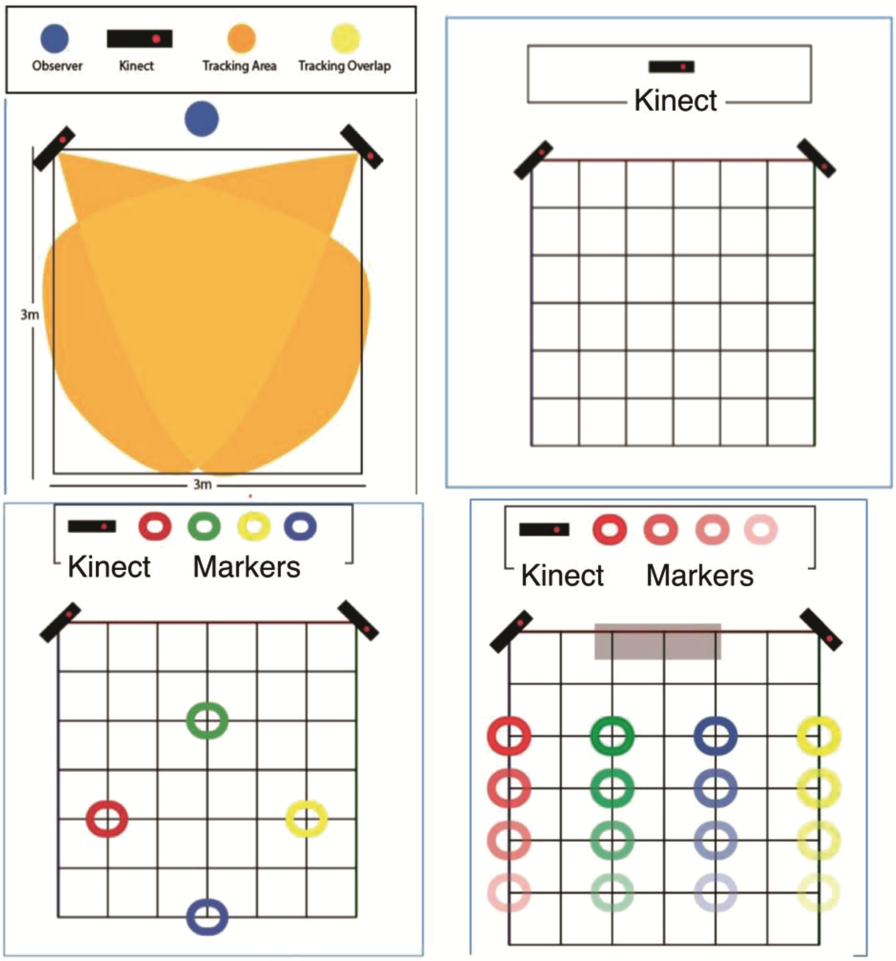

Fig. 2. From top left, clockwise: Pilot, Final, Test 1, and Test 2 setups. (Color figure online)

In the pilot, the dead spots of single and two Kinect configurations were identified (Fig. 2, top left), and the floor markers have been set accordingly (Fig. 2, top right).

4.1 Tracking Multiple Participants Without Occlusion

For this test we had four participants standing next to the marked area and the video for the test was started. Next, the test leader enabled the tracking; a stop watch was started simultaneously. After ten seconds, the first participant entered and walked across the area towards the grid point marked by the solid red circle on Fig. 2, bottom right. Every subsequent ten seconds, another test participant walked into the scene and took up a spot one meter to the right of the former, taking up the green, blue and yellow points in sequence. When all four participants where lined up, they all moved back to the marker directly behind their current, after ten seconds. This was done two more times, recording the test participants' position in a total of four places.

4.2 Tracking Multiple Participants with Occlusion

Before the test was commenced, markers where placed on grid points 28, 40, 24, 12, as shown in Fig. 2, bottom left. The test was then started like the previous one, with four participants standing and the test leader starting the test after the video. Again a stop-watch was employed, and ten seconds after the test was started, the first participant entered the scene. Ten seconds later the last three participants entered the scene, and moved around the scene at their own pace for a minute. After one minute, the participants took a position on the closest marker. Starting with the participant at the red marker, the participants moved clockwise to new markers. This was done every ten seconds for all markers.

5 Results and Discussion

The logs during the tests were saved to a text document, and were subsequently put into a spreadsheet as shown in Fig. 3. The logs are not formatted as a table with column headings; the first column shows when or why the data point was logged. Every tenth-second logs are made for all currently tracked participants under the tag 'tracking continued'. Logs are also made every time tracking of a participant starts or ends.

Tracking Lost:	0	0	UserID:	SubUser 2 5	Time:	17.11814
Tracking Begun:	0.875678	2.040108	UserID:	SubUser 2 5	Time:	22.22043
Tracking Continued:	-0.73238	1.838845	UserID:	SubUser 2 5	Time:	30.00054
Tracking Continued:	-1.40089	2.578075	UserID:	SubUser 1 3	Time:	50.03892
Tracking Begun:	0.083083	1.417884	UserID:	SubUser 1 4	Time:	51.90162
Tracking Lost:	-1.10725	3.013424	UserID:	SubUser 1 3	Time:	53.89946

Fig. 3. Sample log from the test. The columns are the tag, the position, which Kinect tracked the participant and the time.

To find the rate of errors in tracking based on this data, the 'tracking continued' logs were combined from the two Kinects and ordered according to time stamps. To account for participants being tracked by more than one Kinect amounts of overlap in the scene

where calculated. To calculate this, it was decided that if the difference in either the x or z values of two points with the same timestamp were below a threshold of 25 cm, it would be counted as an overlap.

After calculating overlaps, the total amount of detected participants were calculated by adding the number of participants detected without overlaps. The final precision of the tracking algorithm, while tracking multiple participants with possible occlusion, is found to be 81.82%.

In calculating the magnitude of the offset, the logs were sorted in time. The difference between the actually and detected positions were determined, and the magnitude of the offset vector was calculated. The mean magnitude offsets for the first test was found to be 25,55 cm for one Kinect and 15,54 cm for two Kinects.

As our system relies on local network connectivity to synchronize tracking data, delays in data transferring might have resulted in it being out of sync. The delays could have been minimized with a direct connection to a single computer. This would also have avoided the risk of potential packet loss. However, using a single computer was not an option, as the motherboards and USB busses of the computers available for our experiments were not designed to handle data streams from multiple Kinects simultaneously.

The stands which held the Kinects, as well as the markers placed on the grid, might have been slightly moved during the course of the test. This could have caused interference in terms of accuracy and could have been completely avoided by re-measuring and re-calibrating the setup before each test. Our reference grid measurements might also have causing offset; a better solution would be to benchmark our method against well-established and accurate methods.

During our tests, it appeared that the Kinects had trouble with detecting dark colors. This might be due to the IR projection from the Kinect being absorbed rather than reflected by dark materials. This could have been avoided by not wearing dark clothing during tests.

6 Conclusions and Future Work

We presented a procedural method and an implementation of multi-Kinect skeleton fusion on Unity environment. Our method calibrates two Kinects by combining the relative coordinates of a user's torso onto a single coordinate system current, by calculating the rotation, translation, and scaling in a short calibration time. The method resembles the work in [5], however the continuous tracking in successive frames by Kalman filtering in [5] is simplified here to an on-demand calibration scheme, which is accurate enough for entertainment and gaming purposes. The method is tested with a small number of participants, leaving benchmarks against accurate Motion Capture systems such as Vicon, or other methods such as [5, 10] as future work.

In our tests involving multiple users, we have obtained considerably lower accuracy, compared to the theoretical offset of a single Kinect (2.9 cm) in the optimum distance with a single user. Specifically, the average offsets we have obtained were 22.65 cm and 12.64 cm, for one and two Kinects (fused by our method), respectively. However, we

can conclude that our method significantly improves the accuracy of tracking multiple users in an optimally visible area, compared to a single Kinect. It is also worth mentioning that we have used Kinect-v1 s in our experiments, which are known to cause interference by the usage of structured IR-light. Kinect-v2 s, which rely on time-of-flight instead of structured light, may give better results in the future.

Currently we are extending our displacement-only method with velocity-based calibration. The advantage of this method is that it will not require an absolute coordinate system, and the calculations will be simpler. Moreover, we are also experimenting with calibrating the microphone arrays of multiple Kinects by sonic gestures [11].

References

1. Moeslund, T.B., Hilton, A., Krüger, V., Sigal, L.: Visual Analysis of Humans. Springer, London (2011)
2. Zhang, W.: Microsoft kinect sensor and its effect. IEEE Multimedia **19**, 4–10 (2012)
3. Hsu, H.J.: The potential of Kinect in education. Int. J. Inf. Educ. Technol. **1**(5), 365–370 (2011)
4. Berger, K.: A state of the art report on multiple RGB-D sensor research and on publicly available RGB-D datasets. In: Shao, L., Han, J., Kohli, P., Zhang, Z. (eds.) Computer Vision and Machine Learning with RGB-D Sensors, pp. 27–44. Springer, Cham (2014)
5. Li, S., Pathirana, P.N., Caelli, T.: Multi-kinect skeleton fusion for physical rehabilitation monitoring. In: Proceeding 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (2014)
6. Umeyama, S.: Least-squares estimation of transformation parameters between two point patterns. IEEE Trans. Pattern Anal. Mach. Intell. **13**, 376–380 (1991)
7. Giorio, C., Fascinari, M.: Kinect in Motion - Audio and Visual Tracking by Example. Packt Publishing, New York (2013)
8. Monir, S., Rubya, S., Ferdous, H.S.: Rotation and scale invariant posture recognition using Microsoft Kinect skeletal tracking feature. In: Proceeding 12th International Conference on Intelligent Systems Design and Applications (ISDA), pp. 404–409 (2012)
9. Jo, H., Yu, H., Kim, H., Sung, J.: A Study of Multiple Body Tracking System for Digital Signage of NUI Method. Advanced Science and Technology Letters, pp. 91–95 (2015)
10. Staranowicz, A.N., Ray, C., Mariottini, G.-L.: Easy-to-use, general, and accurate multi-Kinect calibration and its application to gait monitoring for fall prediction. In: Proceeding 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC) (2015)
11. Jylhä, A., Erkut, C.: A hand clap interface for sonic interaction with the computer. In: Proceeding Conference Human Factors in Computing Systems, Boston, April 2009