A Low-Cost Motion Capture System for Body Tracking using Synchronized Azure Kinect DK

Jongoh (Andy) Jeong, Yue (Leo) Wang, Mili Shah

May 3, 2020

Abstract

In this paper we introduce a synchronized system of Microsoft Azure Kinects in order to create a low-cost motion capture system capable of body tracking. In particular, we combine data from each Kinect's body tracking software development kit (SDK) to output a single stream of human body joint positions. This single stream of data is advantageous in an environment where various factors, such as occlusion and illumination, may disrupt body joint estimation. Here, we discuss the methodology and requirements necessary to build such a system, the tools needed to synchronize and calibrate the system, and finally describe applications and future directions for this system.

Contents

1 Introduction

The objective of this project is to develop a low-cost motion capture system that can be used as a remedy to common professional, yet much more expensive motion capture (MOCAP) systems. A network of multiple Microsoft Azure Kinect DK's, each at a cost of $399, is still cheaper than a MOCAP system (at least 6 figures in cost). We find that there are numerous applications from which one can design and utilize using this synchronized system that we propose in this report.

This report is organized as follows: we discuss the general specifications of Azure Kinect DK released in mid-2019 in Section 2, review related works in Section 3, describe the environment for testing in Section 4, discuss the procedure to synchronizing multiple Kinect devices in Section 5, examine the outcomes in Section 6, and consider future directions and potential applications of this system in Section 7.

2 Azure Kinect DK

Microsoft Azure Kinect DK[1] is a developer kit for handling sophisticated tasks using sensor software development kit (SDK), body tracking SDK, speech SDK, and computer vision APIs. Each unit contains a 12-Megapixel RGB and a 1-Megapixel depth camera with various operating modes as described in Table 1, 7-microphone array for speech and sound, accelerometer and gyroscope (IMU) for sensor orientation and spatial tracking. The external pins on the rear side allows for synchronizing multiple devices together. For this integration of functionalities, this device itself provides numerous potential in the industry, as presented at the showcase in May 2019 (see Figure 1.

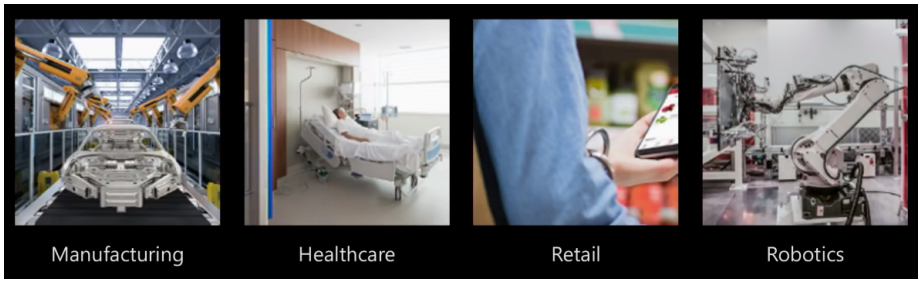

Figure 1: Industry Applications of Azure Kinect DK

[1] https://azure.microsoft.com/en-us/services/kinect-dk/

We primarily focus on utilizing the RGB and depth camer sensors and the open-sourced body tracking SDKs for this project. In particular, we leverage the improved body tracking SDK that gives 32 joint position data verses the previous version that only gave 25 (see Figure 3).

Mode	Resolution	FPS	Operating Range	Exposure Time
NFOV unbinned	640x576	0,5,15,30	0.5 – 3.86 m	12.8 ms
NFOV 2x2 binned	320x288	0,5,15,30	0.5 – 5.46 m	12.8 ms
WFOV 2x2 binned	512x512	0,5,15,30	0.25 – 2.88 m	12.8 ms
WFOV unbinned	1024x1024	0,5,15	0.25 – 2.21 m	20.3 ms
Passive IR	1024x1024	0,5,15,30	N/A	1.6 ms

Table 1: Depth Camera Modes

A sample shown at the system showcase event in 2019 is shown in Figure 2. Note that there is clearly difference between narrow and wide fields of view in the shape (hexagonal vs spherical) and range of observable objects, and between level of granularity in depth sensing capabilities in binned and unbinned modes. However, we also need to take into account the resolution of the camera in each mode (see Table 1). It seems that the best is wider field of view in binned modes, but we forgo the wider angle and resolution in compensation. For this initial step towards synchronization, we decided to work from the NFOV unbinned mode and expand further by verifying each step in a restricted environments.

3 Related Work

Islam et al. uses joint angle estimates from a single Kinect (v2) to recognize a certain Yoga posture. They base their error on a reference model developed from joint data of gymnastics. We differ in that we utilize multiple devices to minimize angular distances of joints and increase accuracy in case of an occlusion [3].

Napoli et al. compares joint angles calculated from Kinect (v2) positions and orientations with a professional motion capture system (Qualisys) at various postures and planes. Similarly, we extend this to a multi-device system for higher precision [4].

4 Environment Setup

The first step towards syncing the Kinects is the physical configuration. Since the estimated coordinates are given in the floating-point triples (x,y,z) in units of mil-

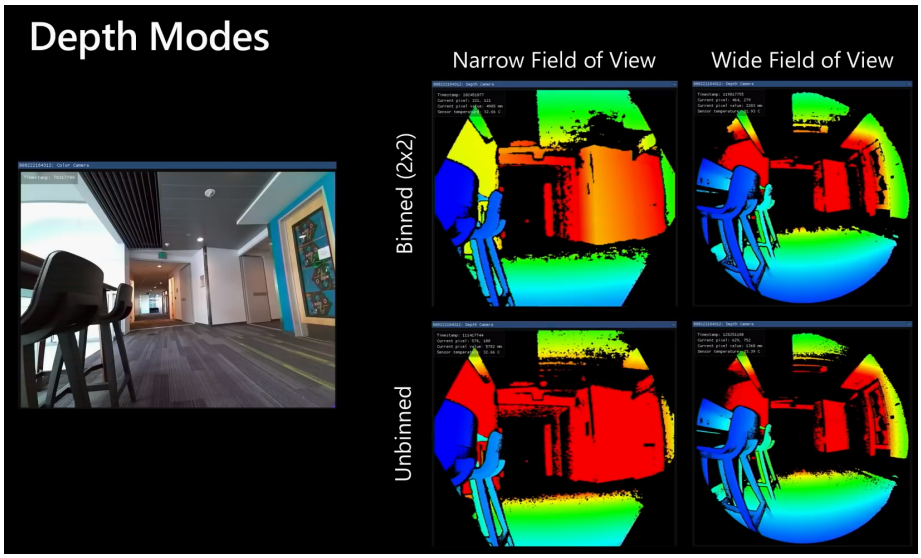

Figure 2: Depth Modes Sample

limeters, we maintain the precision level in the 32-bit single-precision throughout the process. For taking synchronized captures from all devices, we enhance the example code (green screen) from the Azure Kinect Sensor SDK Github repository[2] by setting the maximum allowed time offset between each pair of RGB cameras to be 33 milliseconds and the minimum time offset between the two depth images to be 160 microseconds. The RGB camera resolution is set to 720P, BGRA32 format at 30 FPS, and the depth camera resolution to NFOV-unbinned mode.

4.1 Configuration

Following the daisy-chain configuration, we connect the master to subordinate devices in order using audio jack (male-to-male) cables (see Figure 4 and 5). This configuration allows for one-to-many synchronization whereas the star configuration allows only up to 2 subordinate modes. The sync-out port of the master should be connected to the sync-in port of the subsequent subordinate device, and so on. For testing, we set up the three devices in the following manner shown in Figure 6[3].

[2]https://github.com/microsoft/Azure-Kinect-Sensor-SDK
[3]https://docs.microsoft.com/en-us/azure/kinect-dk/multi-camera-sync

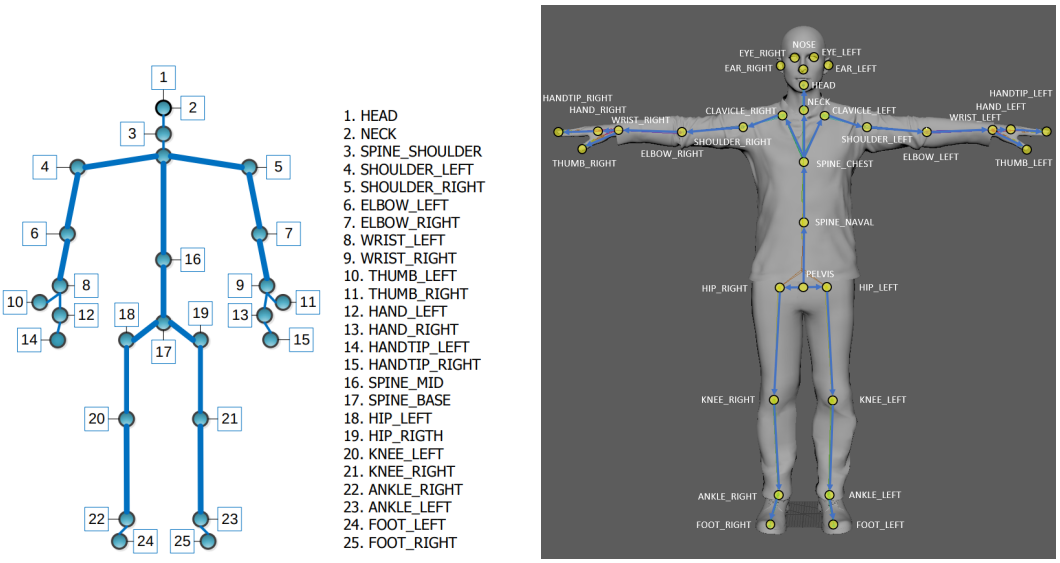

Figure 3: 25 joints for Windows Kinect v2 (2014) on left,
32 joints for Azure Kinect DK (2019) on right

4.2 System Settings

To deliver captures from both depth and RGB cameras, each unit requires a certain bandwidth for data transfer via USB. By default, most Linux-based machines allocate USB controllers 16 MB of kernel memory to handle USB. To work with more than one device, modify in `/etc/default/grub` file,

`GRUB_CMDLINE_LINUX_DEFAULT="quiet splash usbcore.usbfs_memory_mb=###"`

where '`###`' = 16 MB * (# of devices) and in bash shell, type `sudo update-grub`, or,

`sudo sh -c 'echo ### > /sys/module/usbcore/parameters/usbfs_memory_mb'`

The parameters for running the program are set as follows, after several experiments:

1. chessboard height: 9
2. chessboard width: 6
3. chessboard square length: 15
4. depth threshold (mm): 1000 (default)
5. color exposure (ms): 8000 (default)

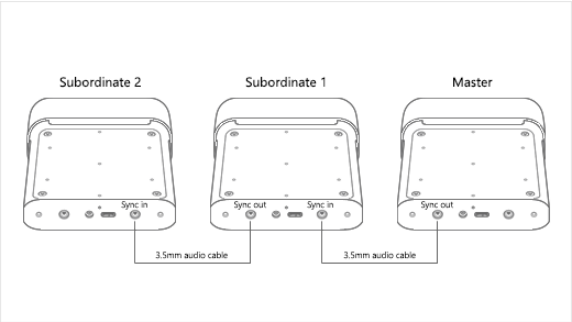

Figure 4: Daisy Chain Configuration in Physical Setup

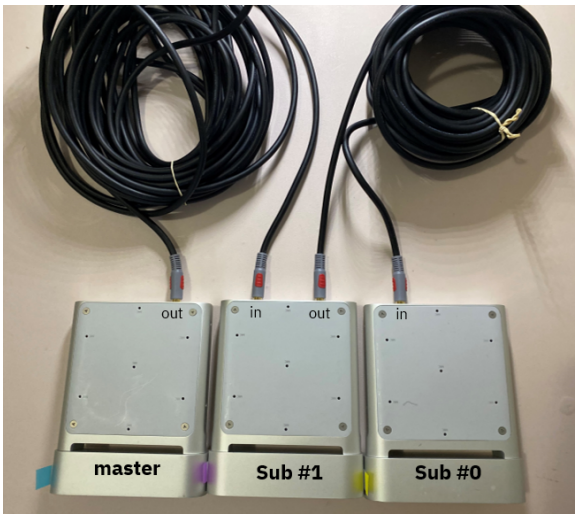

Figure 5: Daisy-chain configuration

6. powerline frequency (Hz): 60 (default)
7. calibration timeout: ∞

5 Procedure

We approach the task of calibration by synchronously taking captures from all master– and subordinate–mode cameras, transforming each estimated joint data stream onto the master RGB camera space, and perspectively projecting homogeneous 3-D coordinates onto a 2-D plane (see Figure 7).

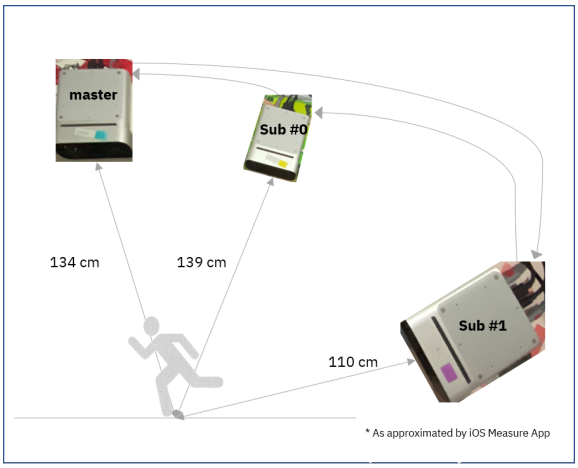

Figure 6: Testing Environment

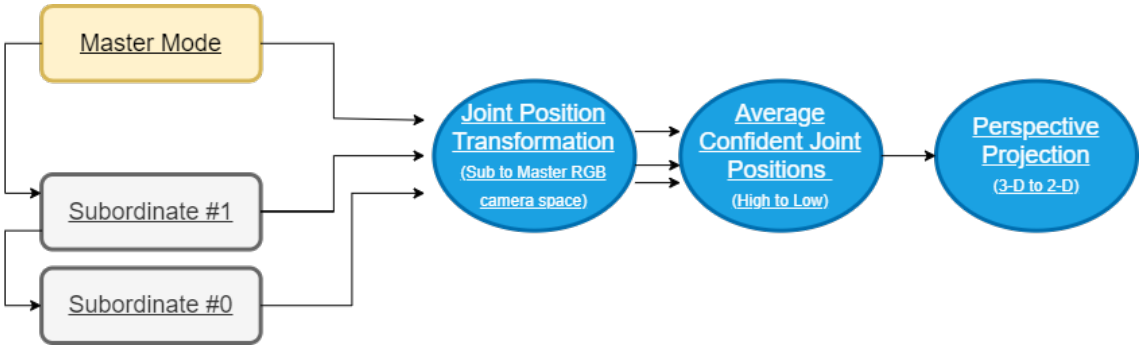

Figure 7: Flowchart

5.1 Steps

Each subordinate device takes captures continuously until one is out of sync with the master capture by the pre-defined maximum time offset. Each set of 32 joint data points from each tracker is then transformed onto the master's RGB camera space using the least-squares fitting of two 3-D point sets [1]. In this method, we determine the extrinsic components (rotation, translation) from singular value decomposition (SVD), and apply the transformation from subordinate to master camera space. In order to estimate the final joint data stream to visualize, for each skeleton we either average body joints with identical confidence level among master and subordinate streams in the order of medium to low confidence levels (medium is the highest as of today). Then we take perspective transformation on this body joint stream [2].

The formula using the intrinsic camera matrix is shown in Equation 1. Note that the current implementation assumes scaling factor s = 1, and we assume zero distortion. We also make use of Rodrigues rotation vector r, which is a more convenient and most compact representation of a rotation matrix R. The relationship between R and r is shown in Equation 2 [4].

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

$$\begin{aligned} s &= \text{scaling factor} \\ [u; v; 1] &= \text{projected homogeneous point (2-D)} \\ f_x, f_y, c_x &= \text{master camera intrinsic} \\ r_{11-33}, t_{1-3} &= \text{computed extrinsic matrix} \\ [X; Y; Z; 1] &= \text{homogeneous point (3-D)} \end{aligned}$$

$$sin(\theta) \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} = \frac{R - R^T}{2} \tag{2}$$

Below are the steps in summary:

1. Calibrate master and subrordinate devices (see Figure 8)
2. For every detected body (numbered in order of appearance) from the cameras, take synchronized captures and track skeletons from all devices
3. From joint positions, compute extrinsic calibration parameters and transform subordinate onto master RGB camera space
4. Estimate body joints by combining streams of the higher confidence level (if equal, then take the average)
5. Project homogeneous 3-D position coordinates onto a 2-D plane for display

We verify the synchronized outcomes by conducting experiments under occlusion and various lighting conditions. In Figure 9, we initially found errors without

[4] https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga61585db663d9da06b68e70cfbf6a1eac

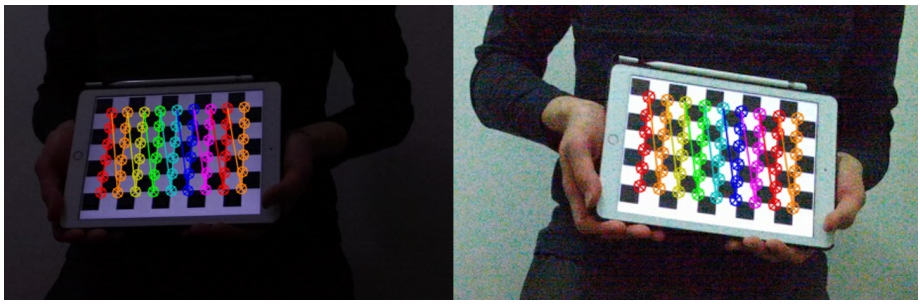

Figure 8: Calibration using Chessboard

transforming the subordinate coordinates onto a single camera space (in yellow). After uniformly transforming subordinates to master RGB camera space, we conducted cases in which we set subordinate device #0 occluded (2nd image), #1 occluded (3rd image), shined an additional light source at the master device (4th image), and then we observed final synchronized output (in green), which appears to be very closely in line with the system that uses only one. Further comparing 2- and 3-device systems in Figure 10, we noticed the synchronized system taking the more confident joint positions in an adaptive manner as desired. Figures 11,12,and 13 show consistency in position norms between normal settings and occluded or shined light conditions. The start of the plot is the normal environment, and each even prominent regions depict the altered conditions. The differences seem to fall under a few centimeters in range, which could also include human movement error due to misalignment in re-positioning after applying changes (e.g. blocking the camera or adding a light source).

Figure 9: Verification of Synchronized System under Occlusion and Illumination

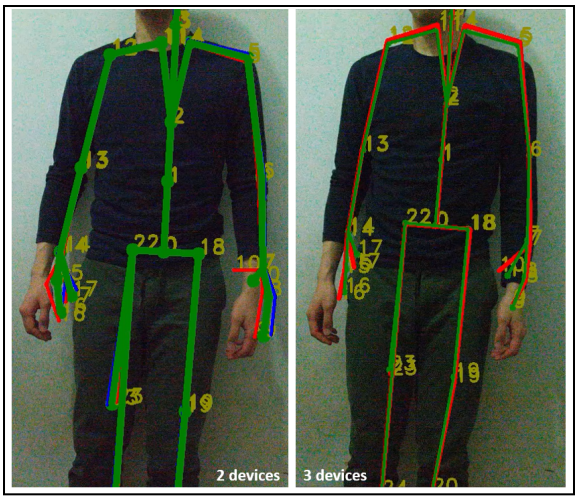

Figure 10: 2- and 3-Device System Outputs

5.2 Joint angles

Joint angles designated as A – L in Figure 14 are computed using Equation 3 from 3-D positional coordinates [3]. The vectors p_1 *and* p_2 are generated from each pair of joints listed in Table 2.

$$\theta_{pos} = arctan2\left(\frac{|\vec{p_1} \times \vec{p_2}|}{\vec{p_1} \cdot \vec{p_2}}\right) \tag{3}$$

Using 'arctan' for joint angles in $\mathbb{R}^3$ space requires an additional consideration for negative angles in certain quadrants. Given that the value of $tan(\theta)$ is positive, we cannot distinguish, whether the angle was from the first or third quadrant and if it is negative, it could come from the second or fourth quadrant. The value of 'atan' is thus an angle from the first or fourth quadrant (e.g. in range: $[-\pi/2, \pi/2]$), regardless of the original input to the tangent. In order to get back the full information, we must not use the result of the division sin/cos but we have to look at the values of each numerator and denominator separately – 'atan2' takes both, the sine and cosine, and resolves all four quadrants by adding π to the result of 'atan' whenever the cosine is negative. In short, atan(y/x) assumes that the input came from either quadrants I or IV, while atan2(y,x) resolves the correct angle in the correct quadrant.

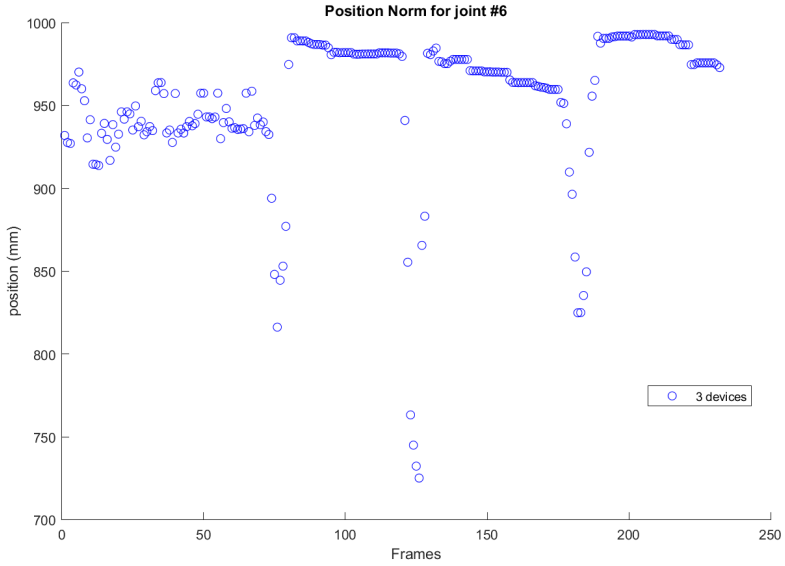

Figure 11: Subordinate #0 in Occlusion

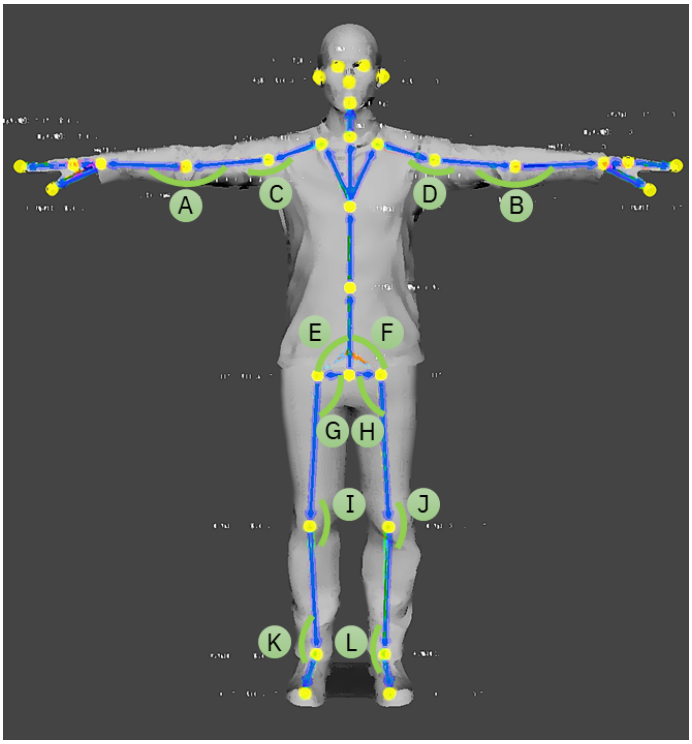

Figure 14: Observed Joint Angles

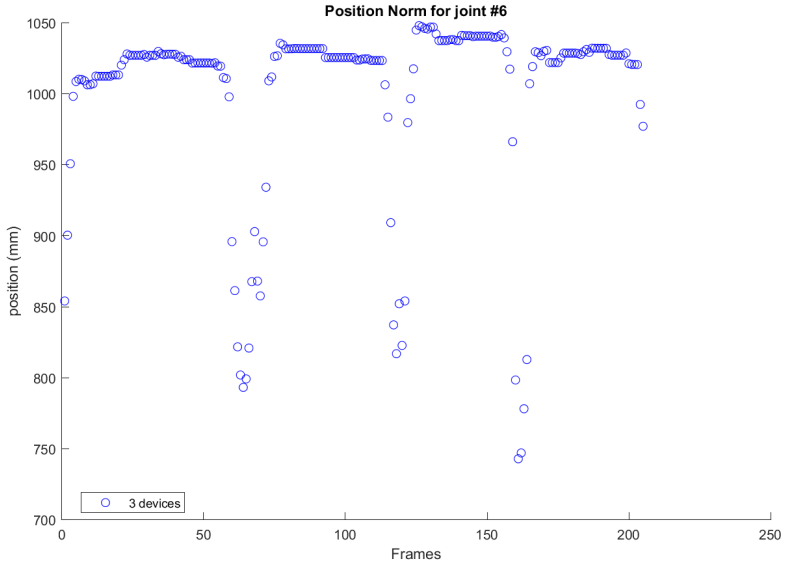

Figure 12: Subordinate #1 in Occlusion

Joint Angle	Joint 1	Joint 2	Joint 3
A	Shoulder Right (12)	Elbow Right (13)	Wrist Right (14)
B	Shoulder Left (5)	Elbow Left (6)	Wrist Left (7)
C	Clavicle Right (11)	Shoulder Right (12)	Elbow Right (13)
D	Clavicle Left (4)	Shoulder Left (5)	Elbow Left (6)
E	Spine naval (1)	Pelvis (0)	Hip Right (22)
F	Spine naval (1)	Pelvis (0)	Hip Left (18)
G	Pelvis (0)	Hip Right (22)	Knee Right (23)
H	Pelvis (0)	Hip Left (18)	Knee Left (19)
I	Hip Right (22)	Knee Right (23)	Ankle Right (24)
J	Hip Left (18)	Knee Left (19)	Ankle Left (20)
K	Knee Right (23)	Ankle Right (24)	Foot Right (25)
L	Knee Left (19)	Ankle Left (20)	Foot Left (21)

Table 2: Observed Joint Angles

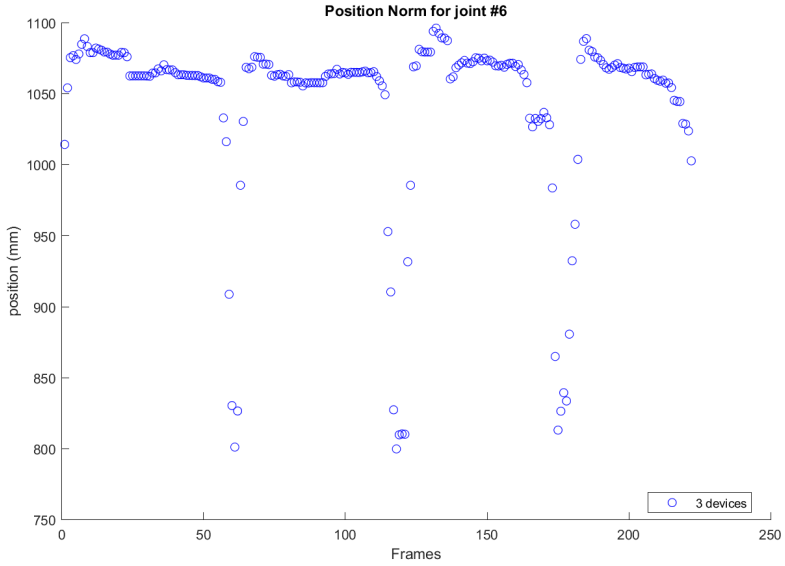

Figure 13: Master in Different Lighting Conditions

6 Results

Several experiments with these 1, 2, and 3-device systems have been run, and it is observed that some joints are estimated consistently for these systems, while some others have deviations from the single-camera system (master). The distance at which the human body is located from the system seems to affect the estimation significantly for each system. For instance, at one location the 3-device system detected the body immediately while the 2-device system had a delay or could not detect at all. In addition, there may be parallax problem in which the estimated joint positions may be offset slightly from the actual in appearance because the devices are estimating from a different height than the person in the view angle. Besides these points to further verify, the system is able to effectively selectively choose the more confident joint data and is capable of visualizing in 2-D display for user interactivity.

7 Future Directions

With the lab space open for testing soon, we hope to continue taking measurements for error analysis across our system and the Vicon MOCAP system (with 6 cameras). One approach is we assume the data from MOCAP is the ground truth, and compare

joint data streams obtained from several other systems. These systems could include (1) single camera with OpenPose algorithm, (2) single camera with Marker-based tracking, and (3) multiple synchronized camera system with Kinect's body tracking SDK.

Aside from system-level analysis, we could develop an interactive tool by adding graphical features to our visualizations. For instance, a well-known figure or character texture can be applied to our skeleton to engage the audience with real-time human pose tracking. Extending this idea, we could also work in ROS environment to control positions and orientations of drone units via Vicon MOCAP, using the open-sourced Crazyflie library.

Some feasible improvements from the programming perspective could include expanding the capability to multi-body tracking, which would require more in-depth understanding of body ID numbering and how to handle cases where devices treat the bodies with different IDs assigned. In addition, code optimization in calculating the extrinsics assuming the devices would be positioned in fixed locations could be achieved.

The project page is maintained at `https://andyj1.github.io/kinect`.

References

[1] K. S. Arun, T. S. Huang, and S. D. Blostein. "Least-Squares Fitting of Two 3-D Point Sets". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), pp. 698–700.

[2] *Camera Calibration and 3D Reconstruction*. URL: `https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html`.

[3] M. U. Islam et al. "Yoga posture recognition by detecting human joint points in real time using microsoft kinect". In: *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*. 2017, pp. 668–673.

[4] Alessandro Napoli et al. "Performance analysis of a generalized motion capture system using microsoft kinect 2.0". In: *Biomedical Signal Processing and Control* 38 (Sept. 2017), pp. 265–280. DOI: `10.1016/j.bspc.2017.06.006`.