Project Overview

Andy Jeong, Leo Wang

April 16, 2020

Abstract

In this project, we aim to realize a calibrated system of multiple Microsoft Azure Kinects for a generalized motion capture system. As an initial groundwork for pose analysis using inverse kinematics, we set up multiple cameras in synchronization to leverage Kinect body tracking software development kit (SDK) to output a joint stream of human body joints in an practical environment where various factors, such as occlusion and illumination, disrupt joint estimation. We hope to expand this low-cost system to use for further kinematic analysis and visual control of robots with human pose.

Contents

1 Introduction

The objective of this project is to develop a low-cost motion capture system that can be used as an alternative to 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 (several ten-thousand dollars). With this alternative, there are a number of applications from which one can design and utilize.

Microsoft's 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.

We focus on utilizing the sensor and body tracking SDKs throughout this project. In particular, we leverage the improved body tracking SDK giving 32 joint data, whereas Kinect v2 (previous version) gives only 25. The resulting skeletons for the previous and current version of the body tracking SDK are described in Figure 1.

Mode	Resolution	FPS	Operating Range	Exposure Time
NFOV unbinned	640x576	0,5,15,30	0.5 – 3.86m	12.8 ms
NFOV 2x2 binned	320x288	0,5,15,30	0.5 – 5.46m	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

1.1 Body Tracking

2 Camera Setup

The first step towards syncing cameras should be the physical setup. We settle on the following specifications for the camera, and work in 32-bit single-precision floating-point format. For the synchronization practices, we extend an example code from

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

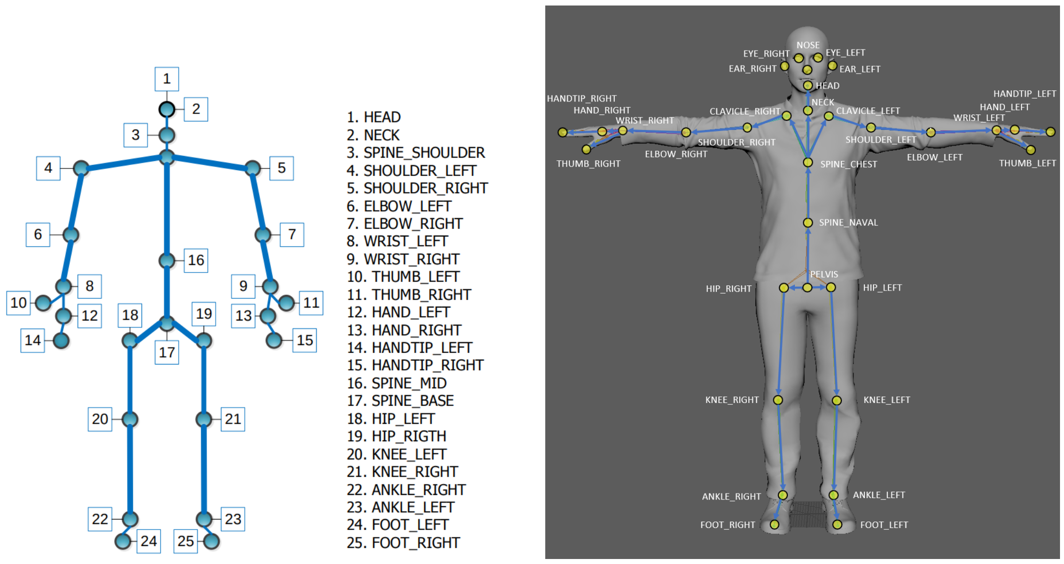

Figure 1: 25 joints for Windows Kinect v2 (2014) on left,
32 joints for Azure Kinect DK (2019) on right

Azure Kinect Sensor SDK Github repository[2], in which we set the maximum allowed time offset between each pair of cameras to be 33 000 microseconds. Also we note that the minimum time offset between two depth images are set to 160 microseconds.

- RGB camera resolution: 720P, BGRA32 format, 30 FPS
- depth camera resolution: NFOV unbinned (narrow field of view)

2.1 System Settings

To deliver captures from both depth and RGB cameras, each unit requires a certain bandwidth for USB data transfer.
Modify in `/etc/default/grub` file, '`###`' = 512 * (number of devices)

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash usbcore.usbfs_memory_mb=###"
sudo update-grub
or,
sudo sh -c 'echo ### > /sys/module/usbcore/parameters/usbfs_memory_mb'
```

[2]https://github.com/microsoft/Azure-Kinect-Sensor-SDK

2.2 Device Chain

Using the daisy-chain configuration, we connect the master to subordinate devices in order using audio jack (male-to-male) cables. The sync-out port of the master should be connected to the sync-in port of the subsequent subordinate device, and so on.

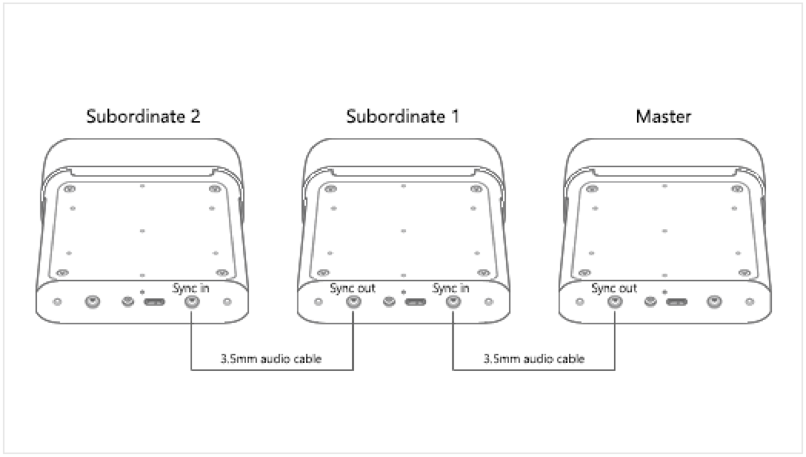

Figure 2: Daisy-chain configuration

3 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 (perspective) projecting 3-D coordinates onto a 2-D display window. Figure 3 describes the procedure.

3.1 Steps

Each subordinate-mode configured unit continuously takes captures 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 are then transformed to 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 matrix (rotation, translation) from the subordinate to master joint points and use it to apply the transformation onto the desired camera space. In order to find the final joint data stream to visualize, we choose, for each

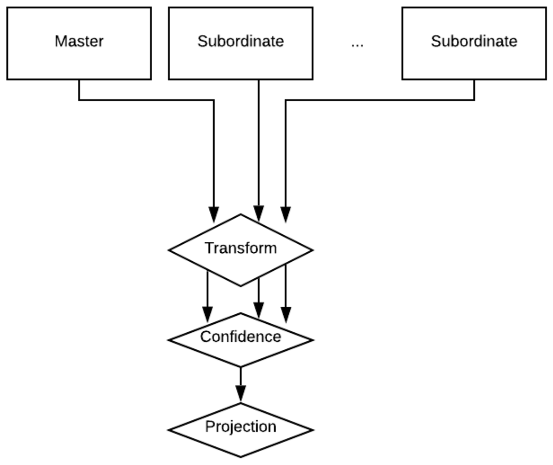

Figure 3: Block Diagram

body, the joint with higher confidence level among master and subordinate streams. Then we take perspective transformation on the final 3-D joint points [2].

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

where (in order) `s` = scaling factor
`[u;v;1]` = 2-D projected point,
$\mathtt{f}_x, f_y, f_z, c_x = master'sintrinsicmatrix$
$\mathtt{r}_{11-33}, t_{1-3} = computedextrinsicmatrix,$
and [X;Y;Z;1] = 3-D point

Below are the steps in summary:

- For every detected body (numbered in order of appearance) from the cameras, track skeletons from each synchronized captures
- From joint positions, compute extrinsics and transform subordinate onto master RGB camera space
- Take joint positions of the higher confidence level (if equal, then take the mean)
- Project 3D positions onto 2D window (no additional translations in x and y)

3.2 Joint angles

Joint angles designated as A – L in Table 2 [3] are computed using Equation 2 from 3-D positional coordinates, and using Equation 3 from orientations at the joints.

$$\theta_{pos} = arccos\left(\frac{\vec{p_1} \cdot \vec{p_2}}{[\vec{p_1}] \cdot [\vec{p_2}]}\right) \tag{2}$$

confirm:

Let Q_1 = quaternion from reference frame to joint 1,
Q_2 = quaternion from reference frame to joint 2,
Q_{12} = conjugate(Q_1) * Q_2 = quaternion from joint 1 to joint 2,
since Q_{12}.w = $cos\left(\frac{angle}{2}\right)$ and Q_{12}.xyz = $sin\left(\frac{angle}{2}\right) \cdot$ (unit axis of rotation)

$$\theta_{quat12} = 2 \cdot arctan\left(\frac{[Q_{12}.xyz]}{Q_{12}.w}\right) \tag{3}$$

To obtain an angle across three joints,

$$\theta_{quat} = \theta_{quat12} + \theta_{quat23}$$

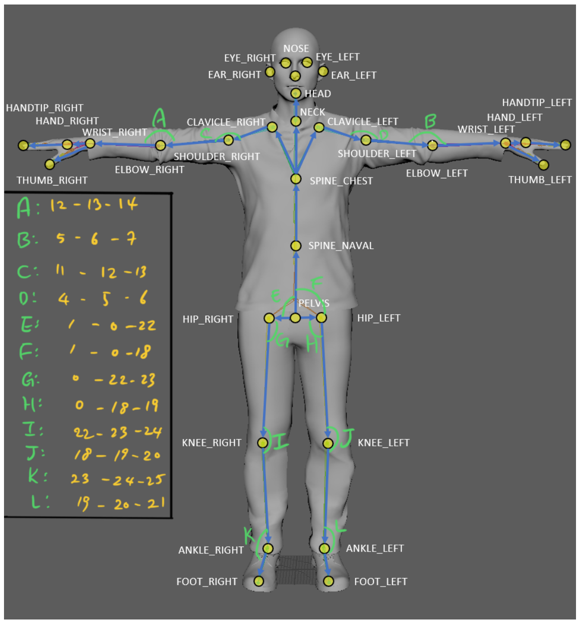

Figure 4: Observed Joint Angles *(handwritten portion will be removed later)*

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

4 Future Directions

With the lab space open for testing, we hope to continue taking measurements for error analysis across our system and the Vicon MOCAP system with 6 cameras. One approach is we take the data from MOCAP as 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, which uses the Crazyflie library and multiple PID controllers.

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.