|  |  |
| --- | --- |
| 3D-Skeleton-Extrapolation  3D skeleton from 2XRGB cams | ABSTRACT  A python/MATLAB based project aimed to extrapolate a 3D skeleton from two RGB cameras.  Alex Nulman, Dvir Segal and Hadas Shahar  3D Cameras |

# Background

Kinect is a line of motion sensing input devices that was produced by Microsoft for Xbox 360 and Xbox One video game consoles and Microsoft Windows PCs. Based around a webcam-style add-on peripheral, it enables users to control and interact with their console/computer without the need for a game controller, through a natural user interface using gestures and spoken commands. The Kinect for Xbox One was discontinued in October 2017, representing the end of the Kinect product line. The Computer vision research community heavily uses this device since its beginning. Both reasons pushed the research community searching for an alternatives based on other methods, among of them is the OpenPose project.

OpenPose represents the first real-time multi-person system to jointly detect human body, hand, and facial key points (in total 135 key points) on single images.

OpenPose for human pose estimation have been implemented using Tensorflow. It also provides several variants that have made some changes to the network structure for real-time processing on the CPU or low power embedded devices.

Our plan is to use OpenPose capabilities to generate a 3D skeleton extrapolated from two RGB cameras. Then, to use the Kinect Skeleton as the ground truth and compare it to our generated skeleton. In such way, we will be able to visually estimate our performance. As a side effect, we hope it will provide the research community ways to use their Kinect acquired data.

# Goal

Using 2 RGB cameras, we would like to create 3D skeleton using OpenPose’s 2D skeletons,

As well as compare the kinect skeleton to our generated 3D one.

# Proposed method

The system relies on OpenPose project – a python flavored version for OpenPose:   
<https://github.com/ildoonet/tf-pose-estimation>

Together with OpenCV capabilities:

<https://opencv.org/>

[](https://www.youtube.com/watch?v=C1Sxk6zxWLM)

Project is implemented in Python and Matlab using Image processing and computer vision toolboxes.

The source code can be found in GitHub.

<https://github.com/shtut/3D-Skeleton-Extrapolation>

# Steps

1. Setting up 2 cameras – at least 1 of which is the Kinect

2. Performing a calibration of the 2 cameras using checker-board

3. Running OpenPose skeleton generation on the 2 RGB images (2D)

4. Using the calibration parameters, triangulate the 2 OpenPose skeletons into a single 3D skeleton

5. Mapping the OpenPose 3D skeleton to the Kinect 3D skeleton by applying the appropriate translation rotation and scaling.

6. Displaying the Kinect skeleton and the extracted 3D OpenPose skeleton overlaying each other for comparison.

# Challenges

* Calibration:

At first, the data we received did not contain any calibration information.

We tried to extrapolate the calibration parameters using vanishing points but that did not work correctly.

Our solution was to create our own data with calibration, requiring calibration parameters for future runs.

* Kinect RGB output:

The RGB output of the Kinect is somewhat low resolution (640x480) and its images were flipped on the x-axis. Our other camera had a higher resolution (1920x1080).

This made the calibration process a bit more complicated since it required some pre-processing on the images- flipping and resizing, as well as ensuring both cameras orientation was the same (landscape as in Kinect).

* Frame Rate:

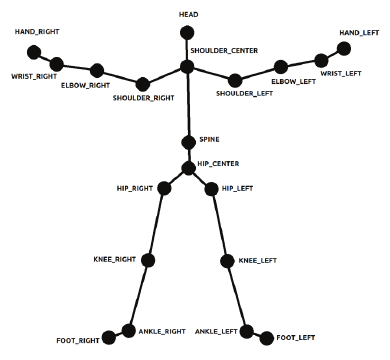
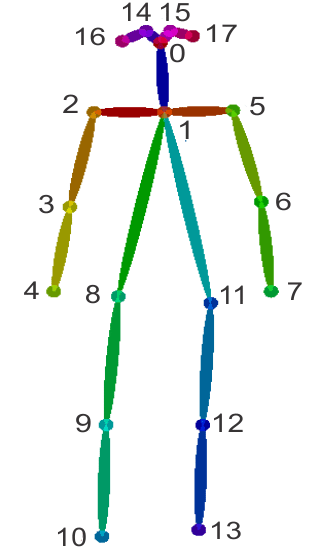
The Kinect’s frame rate is not constant and so matching it with another RGB camera was borderline impossible

For the calibration we had to find matches manually, since even when we find 1 pair of matching images, the varying framerate of the Kinect made it impossible to automate.

* Skeleton differences:

The Kinect and OpenPose skeletons look very different-

Kinect: OpenPose:

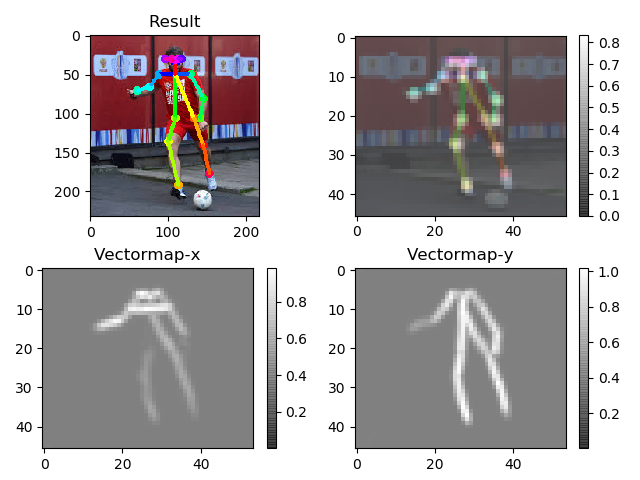


Containing some different joints and having different measurement values.

In order to compare the 2, we had to align them to the same space, i.e- find the rotation and translation between the 2 skeletons, as well as scaling.

# Results

Setting up the OpenPose environment had several setbacks but we eventually we made it happen. As part of our work, we provide instructions how to create OpenPose environment. The below image is a working example based on a demo created with OpenPose default values:



We had to generate our own data with calibration in order to compare the Kinect skeleton to our generate skeleton. We created a stereo set-up using our phone (Xiaomi Mi A1) as our left camera and the Kinect as the right camera.

Our phone camera:

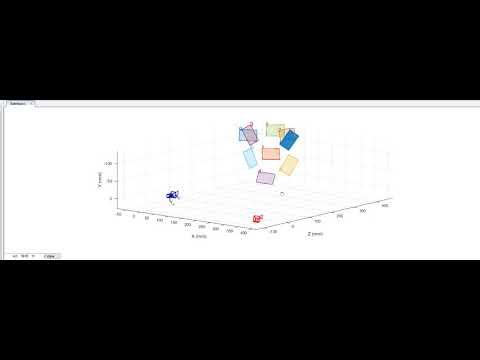


Kinect camera:

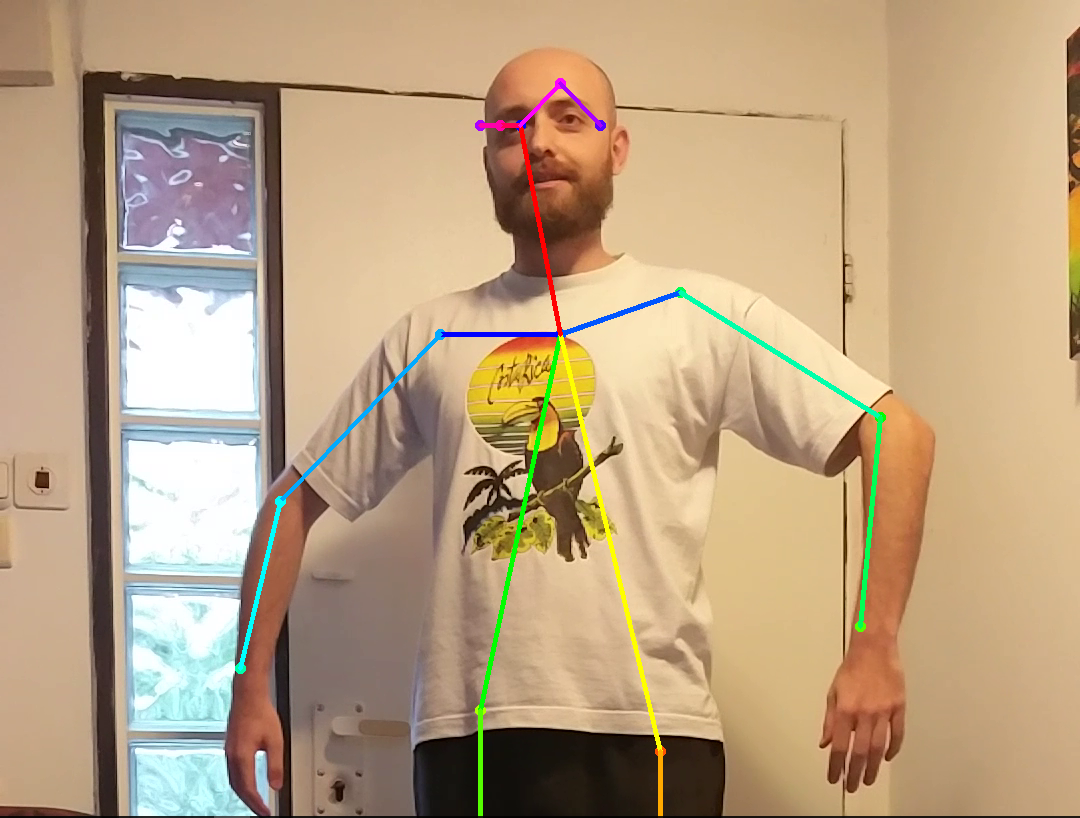


We performed the calibration using checkerboard through MATLAB’s stereo-calibrate which is part of the Image Processing toolbox.

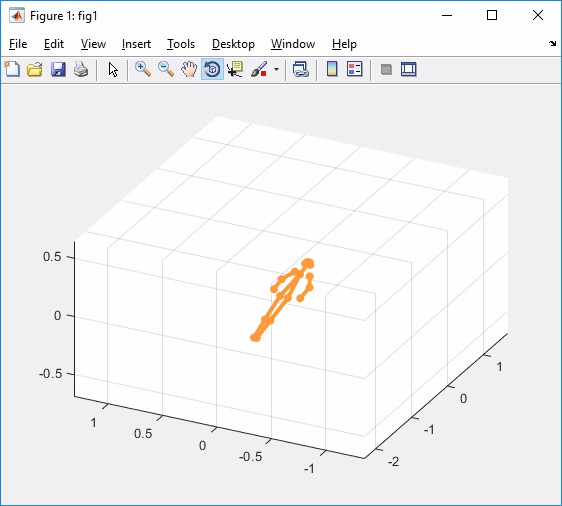


[](https://www.youtube.com/watch?v=9v9EhrFqgn0)

We ran OpenPose on our image pairs-

Finally, to create a 3D skeleton- Using the calibration data, we triangulate the OpenPose joints to find the 3D intersection of each joint.



To compare our skeleton to the Kinect one, we had to find the rotation, translation and scaling between the 2 skeletons.

**Translation:**

We moved each skeleton so that its neck will be located at [0,0,0].

We chose the neck since it was more accurate than the head in the 2 skeletons.

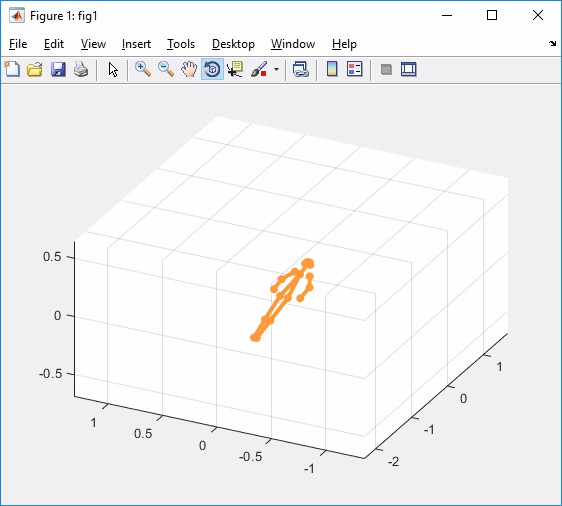
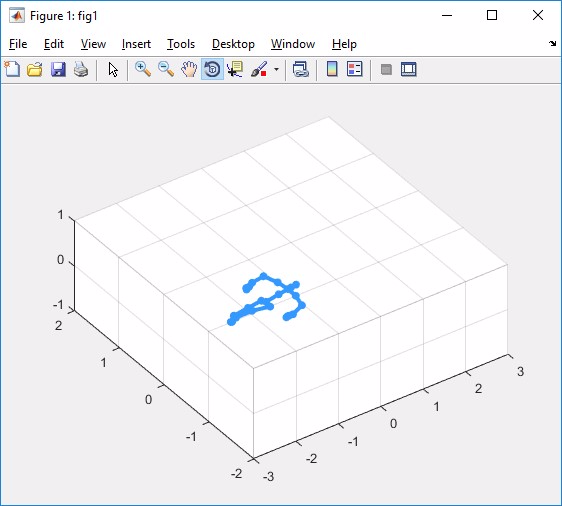
**Rotation:**

We created 2 ‘vectors’ from the neck to the left foot of each skeleton. We then calculated the rotation between the 2 vectors and applied it to the rest of the skeleton.

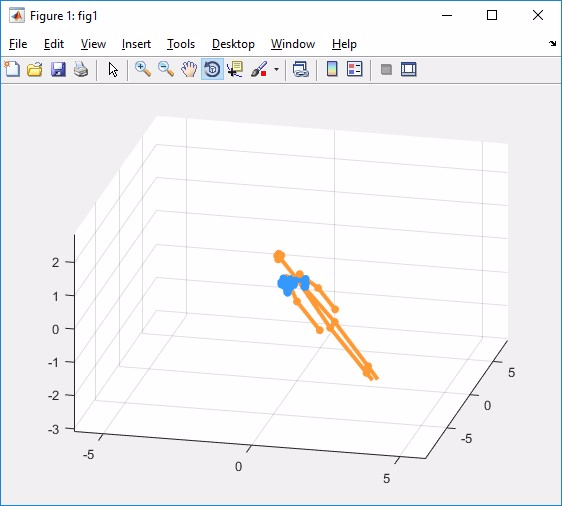
**Scaling:**

Using the same 2 vectors after the rotation, we calculate the ratio between the 2 and use it as our scaling factor so that our skeletons will have the same size.

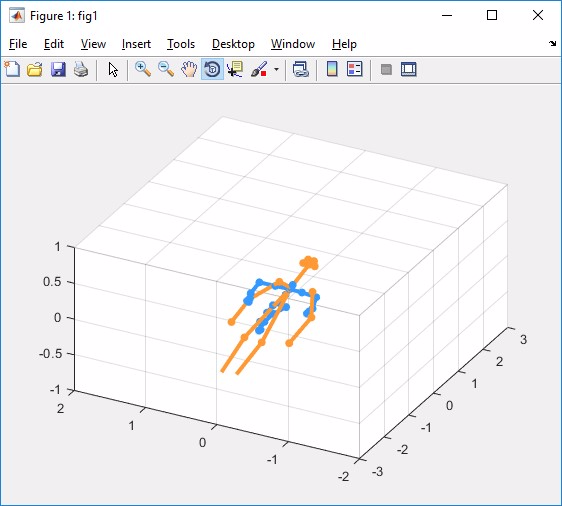
Kinect: OpenPose:



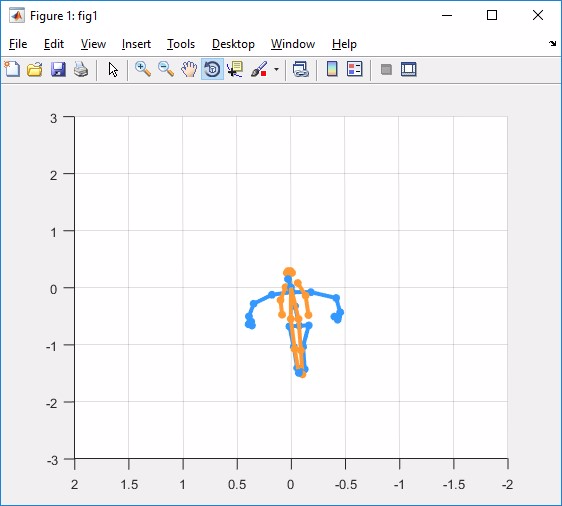
After translation to [0,0,0]:



After rotation:



After scaling:



# Summary

We deduced several outcomes from working on this project:

1. The OpenPose environment is not easy to set; hence, a novice user will struggle setting it up.
   1. We provide a simple instruction guide from future use
2. Calibrating between cameras is not straightforward; it suffers from many environmental influences (such as physical dimensions of the camera, location, hardware parameters etc.)

Several method were proposed in the literature but each has its own setbacks.

1. Matlab has an out of the box camera calibration which we recommend to use
2. OpenPose joints prediction has some failures depending on image pose; the root cause relies on the model trained by OpenPose team. For example in the image below, OpenPose fails to guess the shoulder joint properly. In can be solved by training the model with additional poses.

