Movement Sense

Analyzing Movement using Pose Estimation and Intel 3D Camera

Noy Boutboul

Mark Fesenko

Supervised by

Alon Zvirin

Yaron Honen

As part of a research by

Dr. Maayan Agmon

Abstract

Body posture is one of the fundamental indicators for evaluating health and quality of life. As a part of a research that aims to detect and analyze human posture, our goal is to be able to extract accurate data using photogrammetry tools. The data, collected via Intel depth cameras, consists of subjects standing in various poses and performing sport exercises, analyzed using pose estimation algorithms with heuristics we designed.

In order to validate our results, data were collected from the Vicon system which is regarded as the "gold standard" in assessing human movement, we referred to it as ground truth. The results calculated by the Realsense were compared with those obtained using the Vicon system. Correlation measurements show that there is a strong correlation between our results and the ground truth.

Our work aims to provide precise data of human posture. In addition, we provide an accessible-to-all tool to automate the analyzation and validation process of the data collected by Realsense and Vicon systems.

Environment & Tools

PyCharm version 11.0

PyCharm is an integrated development environment (IDE), specifically for the Python. PyCharm has a free version (Community version) which we use in this project, it is based on intelliJ IDEA IDE.

OpenCV 2.4.12

OpenCV is an open-source library of programming, its main focus is on computer vision field. While written mainly in C++ OpenCV offers a python interface which we used.

Tkinter 8.6

Standard Python interface to the Tk GUI toolkit (package). We used it to built the GUI tool.

Matplotlib 3.3.2

plotting library for the Python, We used it to plot calculations and results in graphs.

Cubemos – skeleton tracking SDK

Cubemos is a pose estimation algorithm that is offered by Intel, the algorithm extracts 18 joints with 3D coordinates and works in real time. The license is not free to use, though it has a free trial of use. Our main use of Cubemos algorithm was comparing the depth result of this algorithm with the depth we generate by projecting 2D pixel.

OpenPose

OpenPose is a free to use pose estimation algorithm. Introduced first in late 2017, OpenPose offers a multi-person system to jointly detect human joints key points in real time. Currently with very high popularity among pose estimation algorithms. our use of OpenPose was mainly to test the CPU version.

AlphaPose

AlphaPose is a free to use pose estimation algorithm. Introduced first in 2018, AlphaPose provides a multi-person pose estimator with high accuracy. In addition there is variety of models available to use, ranging from 18 to 126 skeleton key points, both CPU and GPU versions. While preserving accuracy rate in both CPU and GPU versions, our work was mainly done using this algorithm.

Intel Realsense Depth Camera D415 / D435

We used the Intel Realsense Depth Camera D435 \ D415. The cameras provide us high resolution (1280x720) RGB aligned with depth frame. The frame rate is around 20 fps and it works in real time. Using this cameras allowed us to improve the accuracy of our calculations by calculating in 3D space.

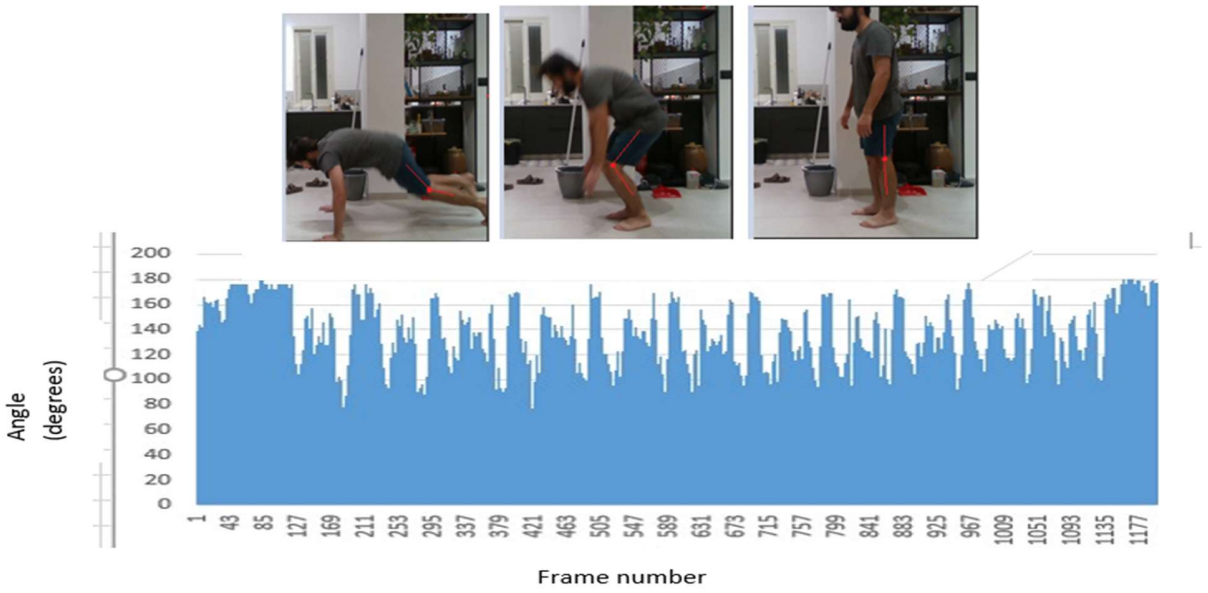
Intel Realsense SDK 2.0 (python wrapper)

Software development kit that is free to use by Intel, most of it written in C++ and it has a python wrapper which we used in this project. The SDK has a lot of features such as capturing the camera streams, aligning the depth frames to RGB frames and more. The SDK also provides many written code examples, and an organized GitHub (LibRealSense).

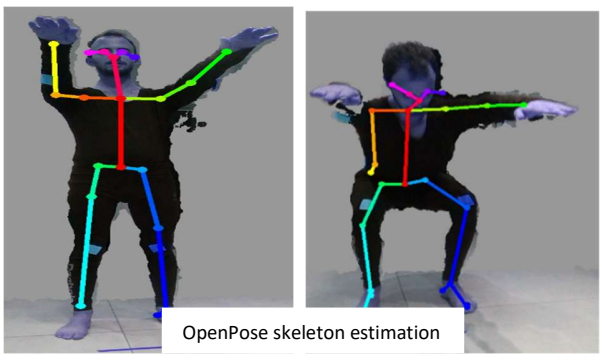
Development Process

Exploratory:

We started our development with a research about pose estimation algorithms, and Intel Realsense SDK. We found out that Intel offers a pose estimation algorithm – Cubemos using Realsense cameras, we decided to try it. We used a bag file (video) sample of a young subject performing repeated burpees exercise. We extracted RGB frames from the bag file, and used Cubemos to extract the skeleton key-points. We displayed the results as a graph where the knee angle is calculated. Note this calculation is done in 2D.



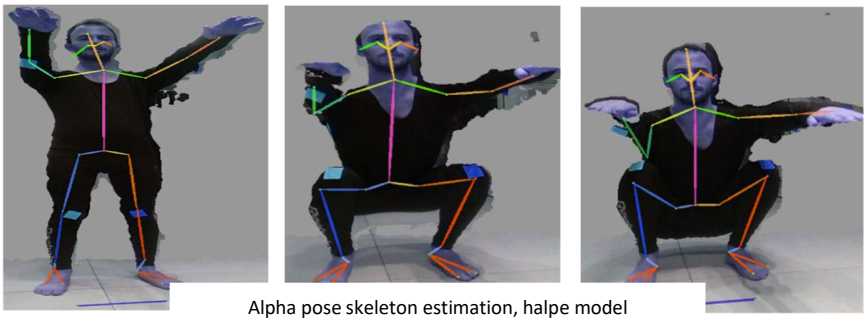
After these results we were still unsure about the accuracy of this algorithm (Cubemos) and decided to investigate other algorithms while the main objective was to find an algorithm with high accuracy of joint estimation. The research led us to OpenPose, a known pose estimation algorithm. We tested the algorithm on a sample of a young subject doing repeated squats and received inaccurate results. A large amount of frames has false predications as you can see in the pictures.
We tried to investigate what is causing the low accuracy and found out that the accurate version of OpenPose was available for GPU only, and the version we used was on CPU. Trying to switch to GPU version failed due to very high demands that we didn't have, Such as Titan X(GPU), 16GB GPU, etc.


OpenPose skeleton estimation

AlphaPose - While searching for pose estimation algorithms, we found an interesting pose estimation algorithm – AlphaPose, a pose estimation algorithm that is focused on high accuracy, AlphaPose also received high score on benchmarks, some even higher than OpenPose. It also has a CPU version that is not compromising accuracy for speed, resulting in low speed but high accuracy with CPU version. We started working with AlphaPose since then.

Alpha pose skeleton estimation, FastPose model.

FastPose model provided us with 18 skeleton key-points, but after discussing specifications with Dr.Agmon we decided we needed more points on the skeleton, especially neck point, hip points, ankle points. therefore, we switched FastPose model to Halpe model, which is 26 skeleton key-points model.



Alpha pose skeleton estimation, halpe model

3D pipeline:

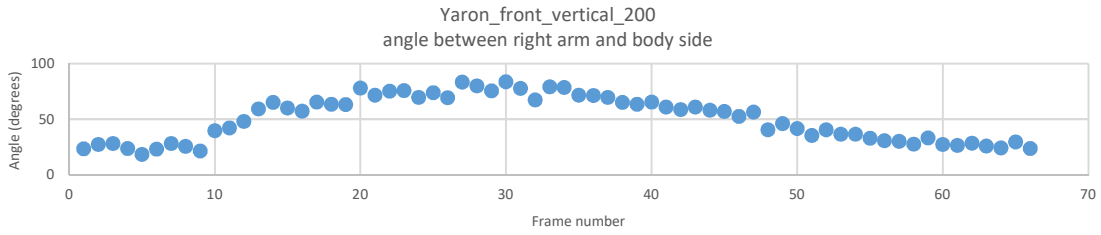
At this point we were ready to start working on preliminary proof of concept. A working pipeline of data collection, skeleton extraction, angle calculation and results display.
We started the work using Intel Realsense SDK with python wrapper and wrote a couple of scripts.

LogGenerator – script that takes a .bag file and runs it on a loop while breaking it into aligned RGB and depth frames. The scripts runs for a fixed amount of time trying to capture new frames via their timestamps. When the loop finishes we create 2 log files. The log files are basically lists of assigned RGB frame timestamp to the closest Depth frame timestamps. First log is assigning frames with unique RGB timestamps key, and the second log is assigning frames with unique depth timestamps key.

We wrote another script to run most of the calculations. The script uploads the bag file, a log file, and a skeleton-key point json file. We start by breaking the .bag file to aligned frames and immediately stopping the playback. The playback is paused to avoid frame loss when executing calculations in between frame captures. While the playback is paused we find the matching RGB frame from the log file, and matching skeleton result to that frame. We use projection function (provided by Realsense SDK) to project the pixel point at the RGB frame to the correspondent XYZ point in 3D space. From this point we calculate the desired data.

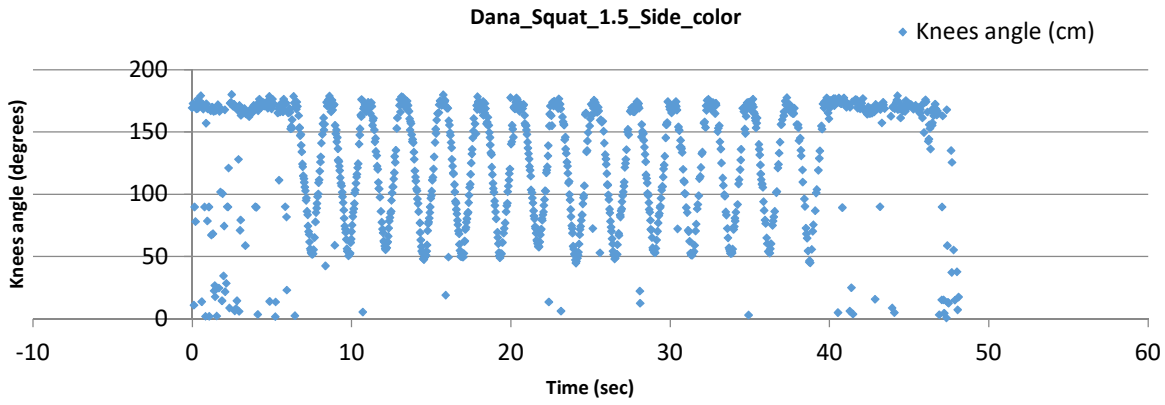
After writing and testing the scripts, we tested it on short bag files.
Displaying results of 5 second long depth video, where subject is performing 1 hand raise.
Results are calculated in 3D space.

shooting protocol: We tested samples from shooting angles: side, front, back, as well as shooting distances (camera distance from the subject): 50cm, 100cm, 150cm, 200cm. Our conclusions are that shootings are best where distance is between 100cm-150cm, 50cm captures small portion of the body, it results in poor skeleton extraction. 200cm is too far and Realsense depth is very noisy.

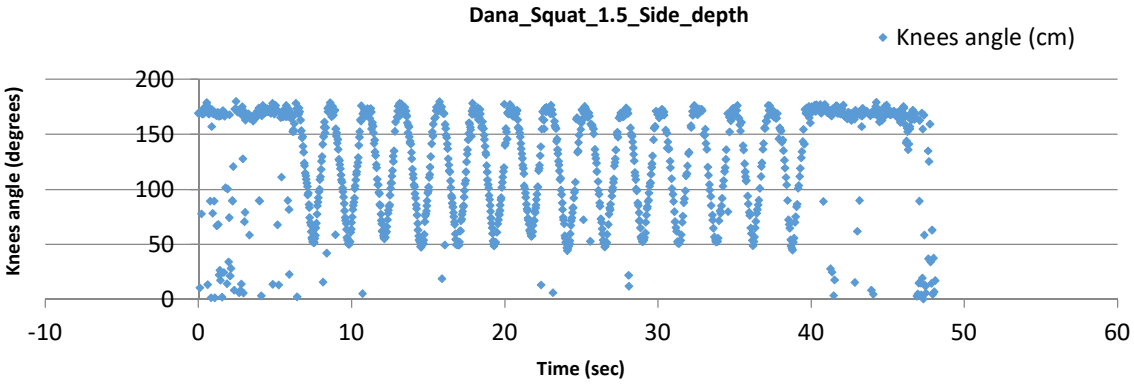


Moving to longer and more complex videos - we used samples of a young subject doing repeated burpees for 40 seconds, both from side and front camera positioning.

We calculated the subject's knee angle while doing burpees and received 2 kinds of results. One by using log which is matching RGB frames to depth frames with the closest timestamps, and the other one by using log which is matching depth frames to RGB frames with the closest timestamps.
Results with color log, from side angle at 150 cm from camera:



Results with color log, from side angle at 150 cm from camera:



We noticed that while working with depth log produced more frames, it also produced a noisier graph. We decided to keep working with log by color (RGB frames are unique and for each RGB frame we map a depth frame with closest timestamp).

Outer projection problem - When trying to extract certain data such as head tilt angle, body tilt angle, we encountered a problem at the calculations which led us to the realization that the projection of a 2D pixel into a 3D space results in a depth point which lays on the outer body. In other words, when projecting onto a person shoulder for example, the depth we get is the distance from his shirt and not from the skeleton itself. This is very problematic, working with outer projection could tilt the vectors in relate to person size.

In attempt to resolve this issue, we tested Intel's pose estimation algorithm – Cubemos, which has an integrated pose estimation algorithm with Intel's depth cameras. Cubemos offers 3D skeleton key points extraction. Our hope was to find out that Intel's algorithm handles the outer projection of depth points and returns a normalized depth estimation in relate to the skeleton.

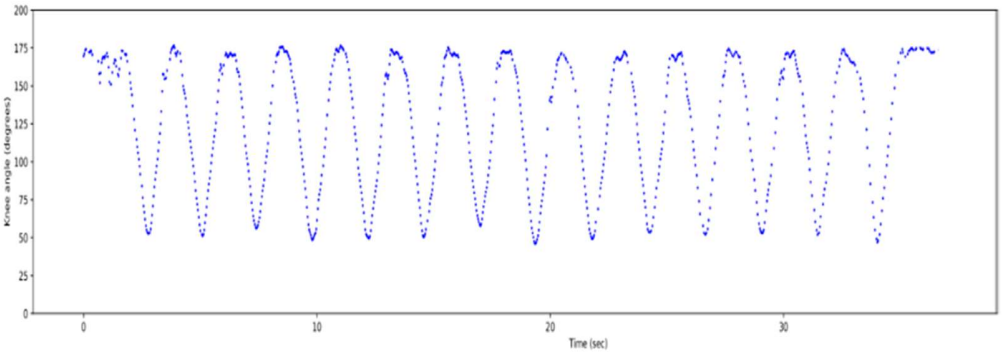
We run a test to check it, the test included extracting skeleton key points in 3D with Cubemos algorithm and comparing them with the same 2D point with projection function that is provided by Realsense SDK.

	Cubemos (intel's algorithm)	**Regular projection of the pixel**
Nose	1.11	1.1170001
Right shoulder	1.11	1.11
Left shoulder	1.14	1.149
Right hip	0.98	0.97500006
Left hip	1.10	1.07
Right elbow	1.06	1.0630001
Left elbow	1.19	1.199
Right wrist	0.98	0.98100007
Left wrist	1.10	1.108

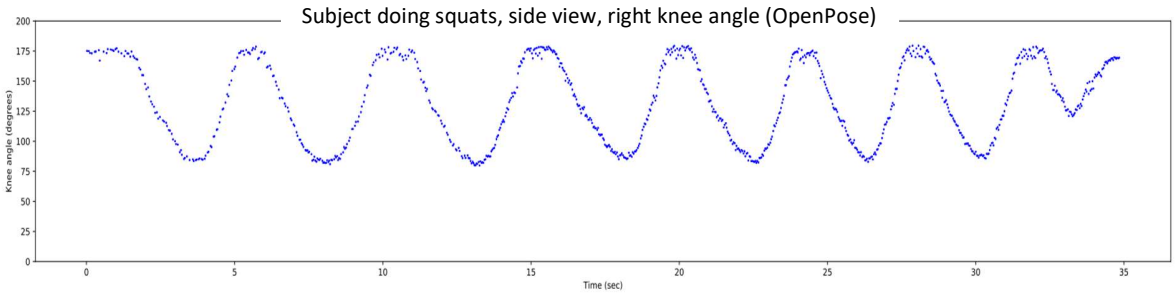
The depth was the same on both algorithms. In addition, we looked into the source code of Cubemos algorithm, and found out they use same projection function that we use on the pixel to extract the depth with no further calculations. Unfortunately, Cubemos did not solve our problem.

Outliers Elimination: One of the challenges in generating the graph was to get rid of the outliers. We came up with a heuristic way to differentiate these points. It is done by calculating the median of the last 3 points, then comparing the new point in compare to the median and checking if it's within a reasonable range from it (e.g. > 1.3X and < 0.7X). If so, we append the point to the graph, else we dump the point and expand the error range, each time a valid point is captured we reset the range to the default values. In addition, to smooth the graphs we used Gaussian filtering.

Graph after eliminating outliers and Gaussian filtering (original graph is above "Dana_Squat_1.5_side_color")



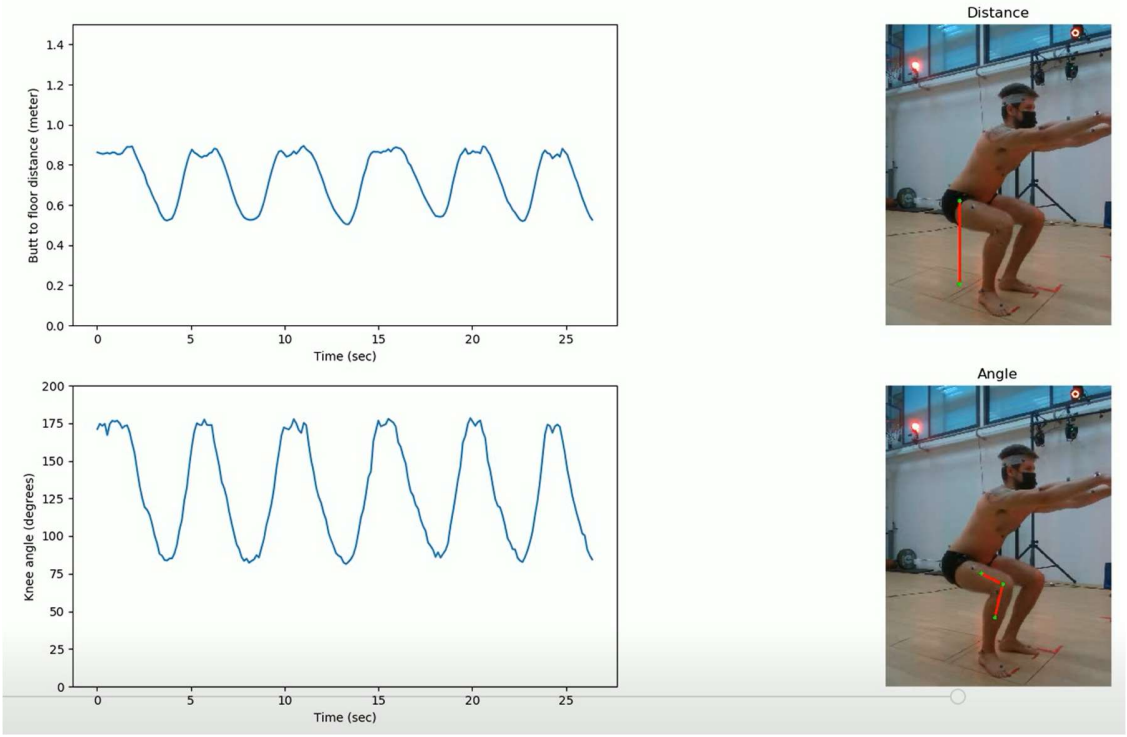
Back to Openpose - We decided to give OpenPose another try, this time we tried to rotate the input images to be horizontal before running the algorithm. The results were much better than what we got from our first attempt with OpenPose, there were much less outliers than there were with AlphaPose even without using our outlier elimination heuristic. Moreover, we found papers regarding OpenPose reliablity and even a paper about validating it with a Vicon system with squats exercise. All these reason led us to switch from working with AlphaPose to Openpose.



Visualization:

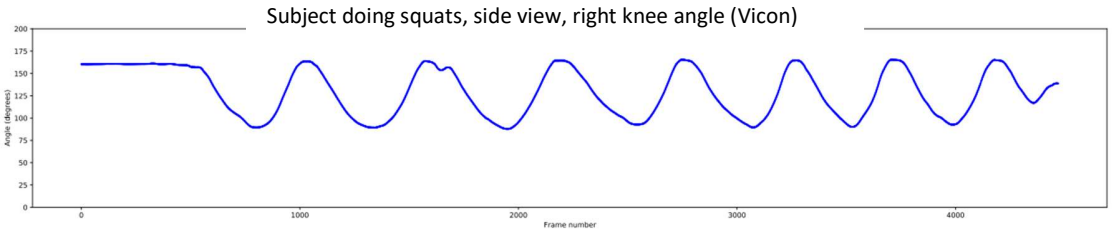
We built a code to graphically visualize the measurements calculation in both the image frame and in a graph through time. The code creates a video which consists of the original camera frames with basic drawing of the part we measure, and a graph that plots the calculated data as the video goes on.

A frame from the graphic visualization
First row shows distance from subject bottom to the estimated floor.
Second row shows the subject's right knee angle.



Vicon Validation:

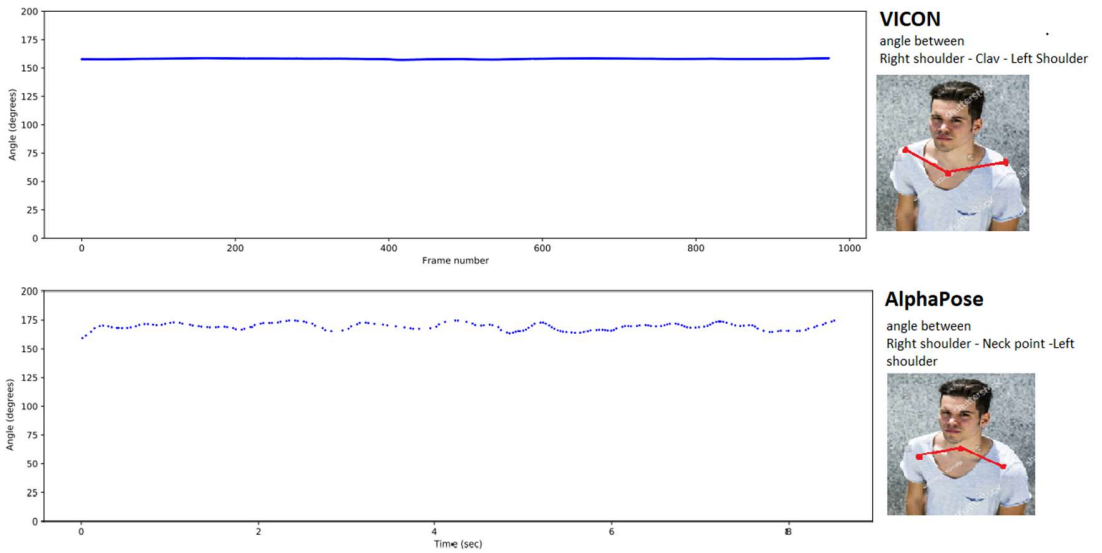
As the need to validate our results grew, our work shifted to validate our results – Realsense samples with pose estimation, with Vicon's data. Starting with building a code to read the Vicon data and calculates the measurements.



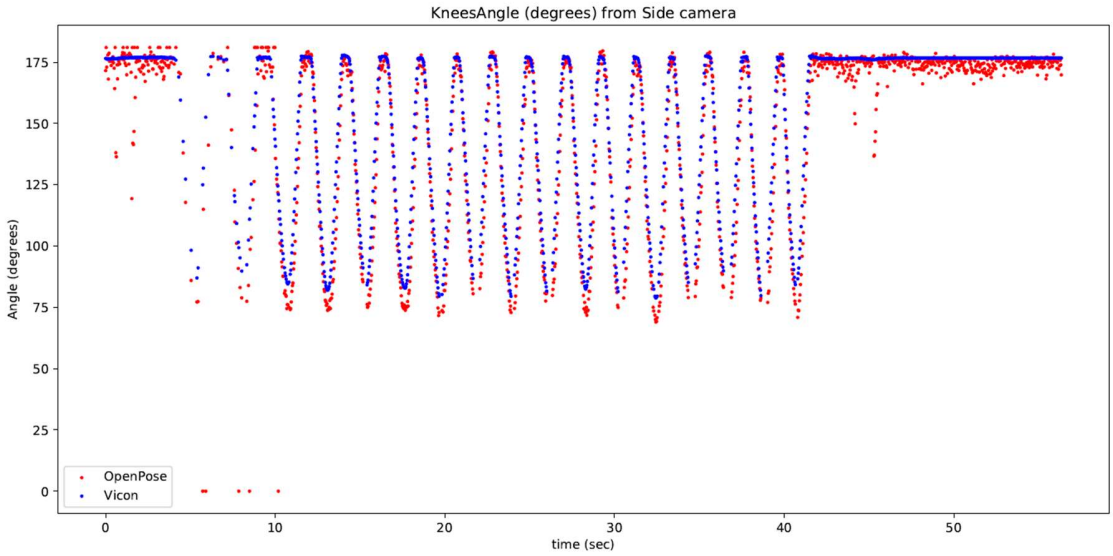
Diverse skeleton points - We encountered a challenge when trying to fit the Vicon points to the skeleton key points from the pose estimation algorithms. While the key points from the Vicon were precise at the anatomic level, the key points that we extract with pose estimation algorithms had no relation to body anatomy points. This resulted in a big difference when trying to calculate certain measurements such as shoulder's kyphosis, head tilt, etc. mainly the measurements that included points that are not joints but a specific bone or vertebra.

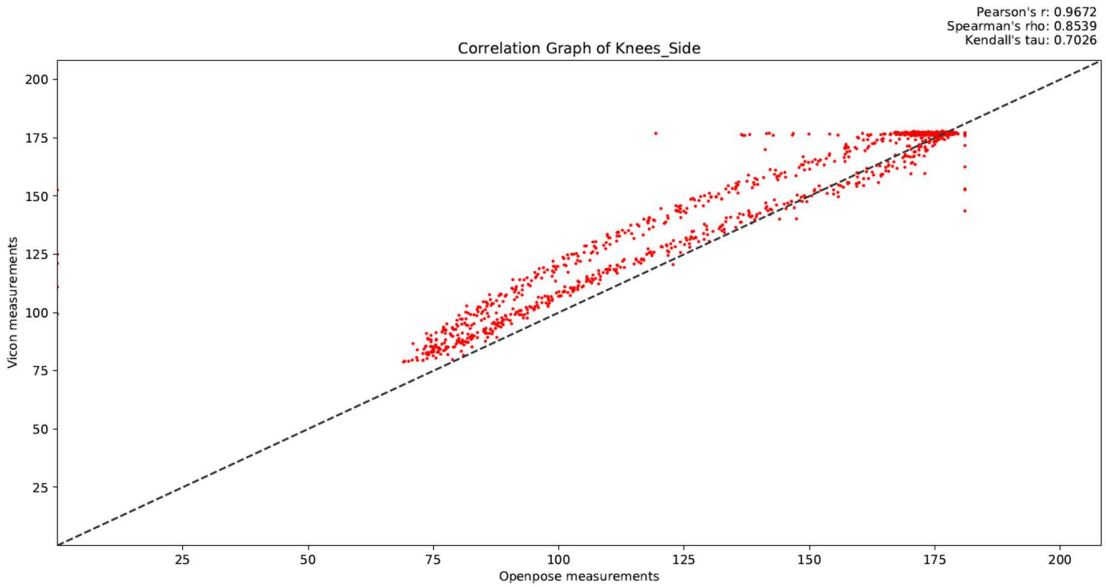
Unfortunately due to the reasons above we were not enable to calculate the shoulder's kyphosis measurement correctly and only calculate the opposing angle from the one that the Vicon calculates.

Shoulder's kyphosis example (added image to illustrate the difference of the angles):

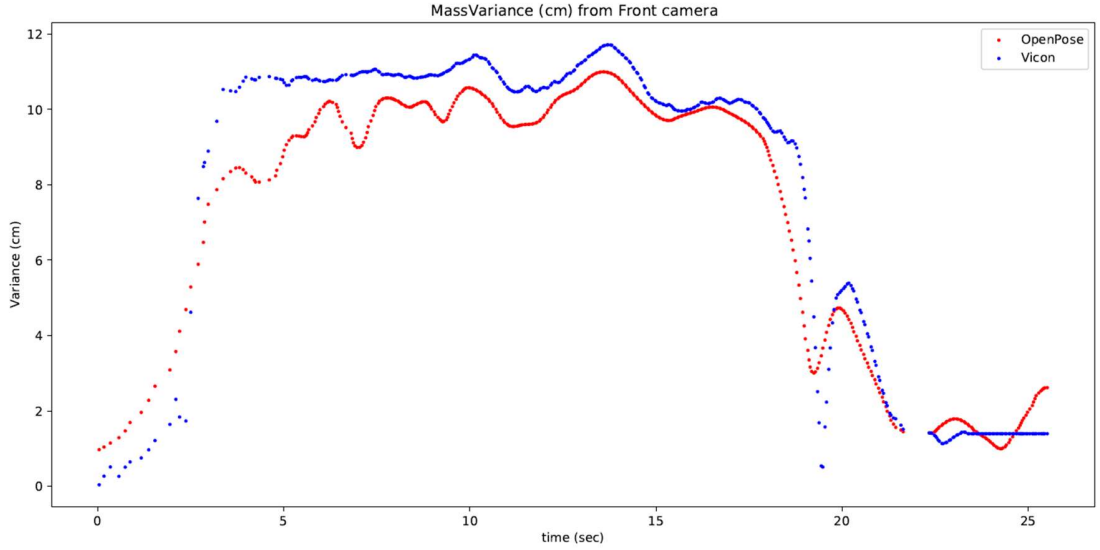


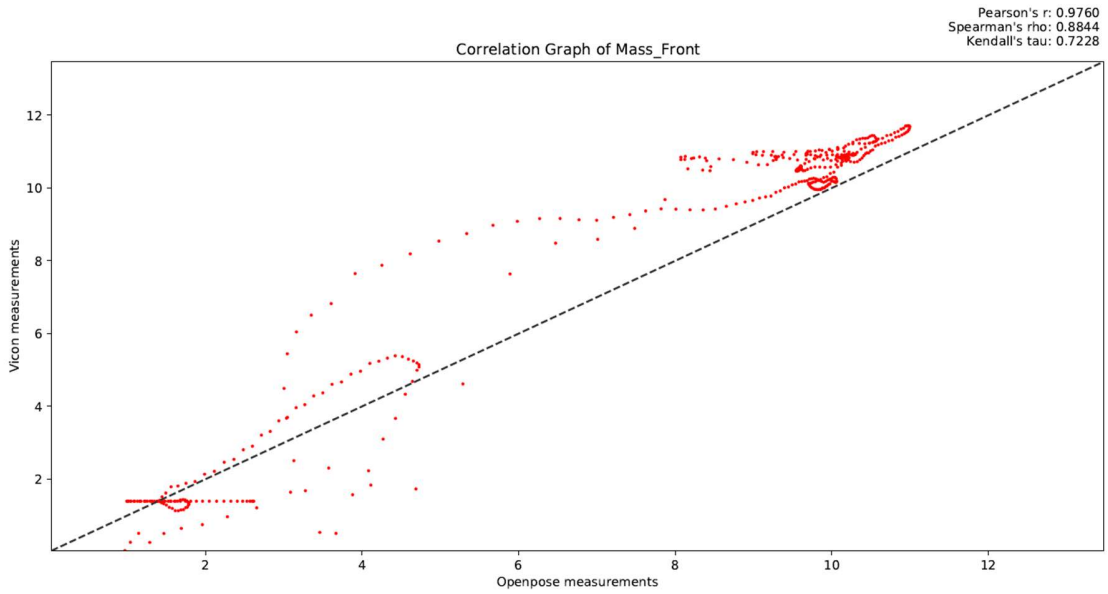
Timestamps alignment - Another challenge we faced is that the Vicon's output frames don't have timestamps on the frames, thus, we could not align the Vicon frames with the Realsense ones.
To solve this challenge we built a sync-by-movement code that syncs the Vicon frames with OpenPose frames. The code detects on both Vicon data and Realsense data where the subject raises their hand to a T-pose (all videos collected are to start with T pose), we use this point as the 0-point in time where both Vicon and Realsense frames are aligned. The Vicon's FPS rate is steady on 120 and the Realsense FPS varies from 10 to 25 but it has timestamps. We aligned them based on Realsense timestamps.
Results:



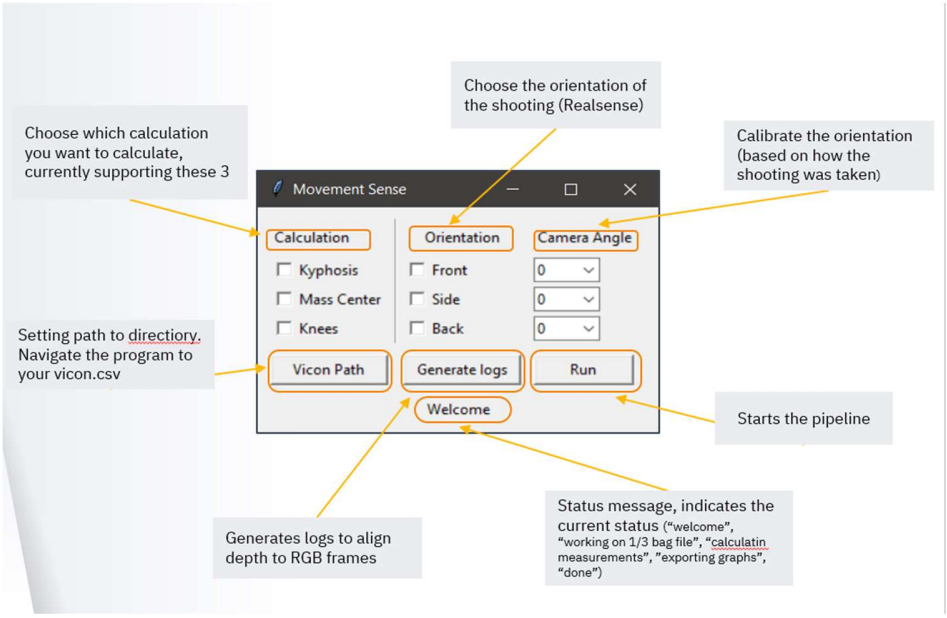


Center of Mass - The researcher requested we calculate another kind of measurement – Center of Mass. In order to calculate it, we set a new point at the middle of the waists as point_s, then the subject is requested to perform a certain exercise, and for each frame we calculate the new position for the point – point_t and then calculate the Euclidian distance between point_s and point_t.
We did not manage to calculate this measurement with sufficient precision. As the Vicon handled it with no problem, the results from Realsense + OpenPose had a lot of noise and instability.
Results:



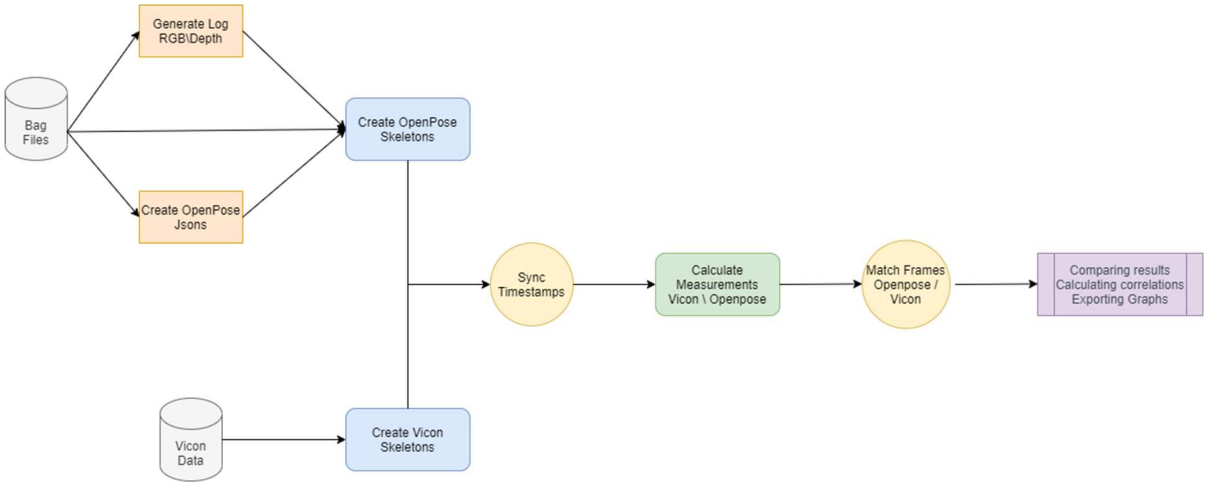


GUI Tool:



After consideration we decided to build this automation tool to provide accessibility with automating the process of graph creation for the measurements and correlation between Vicon data and Realsense BAG files.
The tool calculates the measurements from each camera orientation requested. While calculating a popup window of the current frame with OpenPose skeleton points and a frame counter (this is useful for validating the data). At the end of the run results will be generated at 'Graphs' folder containing the measurements graphs and correlation graphs.

GUI Code Flow Chart



Manual:

First, the user needs to organize the folder according to the next diagram:

```
path\
    vicon.csv
    front.bag
    front_log.json
    side.bag
    side_log.json
    back.bag
    back_log.json
    front_openpose
        1.607859213e+12_keypoints.json
        ...
    side_openpose
        1.607859213e+12_keypoints.json
        ...
    back_openpose
        1.607859213e+12_keypoints.json
        ...
```

- All rotations are optional
- xxx_log.json can be generated using this tool
- xxx_openpose is generated using the Openpose tool (separate)

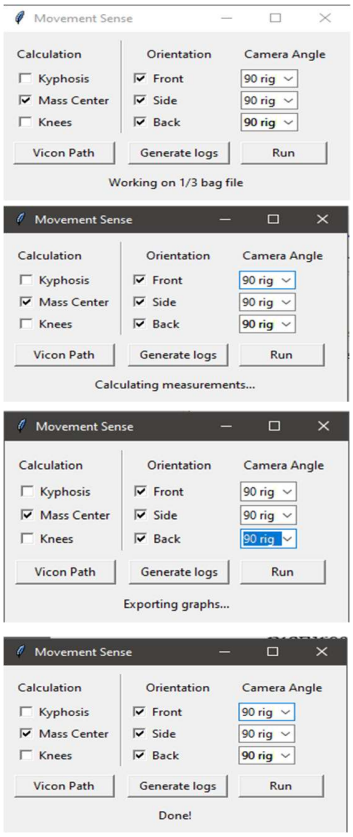
Then, click 'Vicon Path' to navigate the program to your vicon.csv.

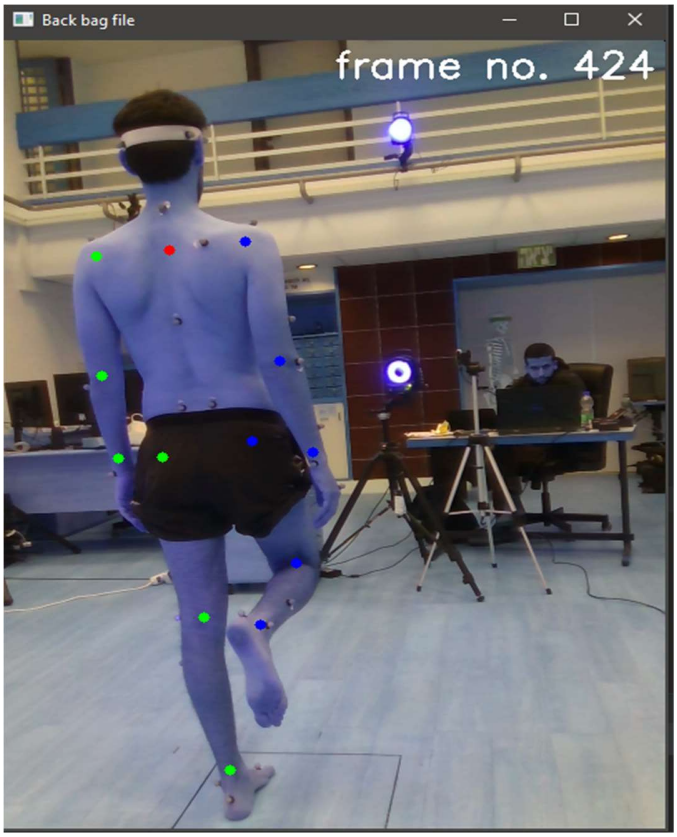
If xxx_log.json are missing, please select the requested orientations and each camera's angle, and press 'Generate logs'.

Then, to get measurements graphs, please choose the requested calculations and camera orientations used and press 'Run'.

The program will output the requested graphs to the same path originally used, inside a new folder named 'Graphs'.

Pictures from an example run:





Conclusions

The current results shows that our method (Realsense + Openpose) for measuring certain kinematic measurements is strongly correlated with the ground truth in measurement involving anatomical points available in our system.

Some measurements such as: shoulder kyphosis, head tilt, etc, relying on points unavailable in our system (due to anatomical vs estimated skeleton points issue) could still be estimated, via areal points highly correlated to those.

However, our system still is much less reliable than the Vicon system. Measurements contained more noise and were sometimes biased. This can be addressed in future studies

Future Work

Our results shows that extraction of body posture measurements using photogrammetry tools is correlated with the "ground truth" data to an extent.

In addition our method yielded results with noise and some a bias.

Perhaps, it will be more robust to try training a machine learning model that given the Vicon data as labels could train on the 3D video samples.

A well-trained model could be able to predict skeleton points up to their anatomical level and have high precision and reliability with in compare with the Vicon system. This could lead to clinically uses in medical fields and other applications.

References

Intel RealSense GitHub

Sample Code for Frame Alignment (Python Wrapper)
Pipeline Control sample code (Python Wrapper)

Best-Known-Methods for Tuning Intel® RealSense™ D400 Depth Cameras for Best Performance)

AlphaPose GitHub

OpenPose GitHub
OpenPose Maximum Accuracy Configuration

Skeleton Tracking SDK by cubemos™ - Intel RealSense Technologies
Cubemos Documentation

MPII Human Pose Benchmark (Pose Estimation) | Papers With Code
A Short Guide to Pose Estimation in Computer Vision | by Siddharth Sharma | Medium

OpenCV: OpenCV-Python Tutorials

tkinter — Python interface to Tcl/Tk — Python 3.9.5 documentation
Python - GUI Programming (Tkinter) - Tutorialspoint

Pyplot tutorial — Matplotlib 3.4.2 documentation
Matplotlib - Scatter Plot - Tutorialspoint