

Air Drumming: Applied on-device body pose estimation

Maurice Van Wassenhove
Ghent University
Ghent, Belgium
maurice.vanwassenhove@ugent.be

Abstract—Body pose estimation is a new and exciting field in Computer Vision. It allows for the estimation of the position of key body parts in images or videos. In this paper, we explore the use of MediaPipe, a popular open-source library, for Body Pose Estimation, promising real-time on-device body pose estimation. We present a demo application that uses MediaPipe to estimate the body pose of a user air drumming. The accuracy of the pose estimation is evaluated, and some issues are discussed. MediaPipe Pose is shown to achieve an average accuracy of 5-10 mm but suffers from noise and jitter. We present a post-processing method that can reduce that noise and jitter. The method is general, and can be applied to any pose estimation model.

Index terms—Body Pose Estimation, MediaPipe, Computer Vision, Motion Capture, 3D Pose Estimation, Demo Application

I. INTRODUCTION

Body pose estimation is a relatively new domain in computer vision that aims to estimate the position of key body parts in images or videos. This technology has a wide range of applications, including human-computer interaction, augmented reality, and motion capture. In recent years, there has been a growing interest in on-device body pose estimation, which refers to the ability to perform body pose estimation directly on a device, such as a smartphone or tablet, without the need for specialized hardware or an internet connection. This has been made possible by advancements in deep learning and computer vision, which have enabled the development of lightweight and efficient models that can run in real-time on mobile devices. Over the years, various models have been developed, such as Human Motion [1], OpenPose [2], AlphaPose [3], [4], MindPose [5], and MediaPipe Pose [6], each with its strengths and weaknesses. MediaPipe Pose by Google is a popular open-source library that provides real-time on-device body pose estimation, making it an attractive choice for interactive applications.

In this paper, we discuss some issues that are present in the MediaPipe Pose model, such as noise and jitter in the output. A relatively simple, yet effective, post-processing method is

proposed to reduce these issues. The method is general and can be applied to any pose estimation model. The accuracy of the pose estimation is evaluated, and some issues highlighted.

A demo application has been developed to provide a fun and interactive experience that showcases the capabilities and limitations of on-device body pose estimation. The application uses MediaPipe to estimate the body pose of a user air drumming and generates drum sounds based on the user's hand and foot movements.

II. MEDIAPIPE POSE

MediaPipe Pose is an open-source library developed by Google that provides real-time on-device body pose estimation. It is part of the MediaPipe framework [7]. The library uses a deep neural network to estimate the position of specific landmarks. Three different models are available each increasing in size. The models are LITE, FULL, and HEAVY. The LITE model is the smallest and fastest, but also the least accurate. The FULL model is the default model and provides a good balance between speed and accuracy. The HEAVY model is the largest and most accurate but also the slowest due to the large network.

MediaPipe Pose outputs the 3D coordinates of 33 key points on the human body including elements of the hands and feet, called landmarks. Two kinds of landmarks are available, Landmark and WorldLandmark. The Landmark landmarks are the 3D coordinates of the key points in the image frame coordinate system. This means that the origin of the coordinate system is a corner of the image frame and the horizontal and vertical axes are normalized to be between 0 and 1. The depth is then the depth from the camera to the landmark. The WorldLandmark landmarks are the 3D coordinates of the key points in a more traditional coordinate system. The mid-point between the hips is taken as the origin of the coordinate system where all axes originate from. The WorldLandmark landmarks estimate the real-world size of the landmarks instead of the normalized size of the Landmark landmarks. WorldLandmark landmarks decouple the coordinates from the camera at a slight loss of accuracy.

III. MEASUREMENTS

The performance of the MediaPipe Pose model was evaluated by measuring the accuracy and deviation of the model under different conditions. Using a traditional motion capture system as ground truth.¹ The measurements were conducted in a controlled environment with a single user performing various movements, such as raising hands, swinging arms, and air drumming.² The accuracy of the pose estimation was evaluated by comparing the estimated key points with the ground truth key points. The accuracy is the absolute deviation from the ground truth key points, measured in millimetres. The mean, standard deviation, minimum, maximum, and percentiles of the accuracy were calculated for each key point. The results of one of the measurements is shown in Table I. It shows that on average the deviation from the ground truth key points is around 5-10 mm, with a standard deviation of around 6-10 mm, for the Y and Z axes, the lateral axes. The deviation is a lot higher for the X axis, which is the depth axis. Due to this low accuracy in the depth axis, the 3D pose estimation is not as accurate as one might hope. However, for the application of air drumming, the depth axis is not used, so the accuracy of the lateral, 2D axes is sufficient for this use case.

TABLE I: THE TOTAL DEVIATION FROM THE MAURICE_DRUM_REGULAR MEASUREMENT. MODEL: FULL, MARKER TYPE: LANDMARK.


Deviation (mm)	X	Y	Z
mean	44.054789	5.877316	10.028784
std	52.458578	6.611694	10.963382
min	0.000578	0.000225	0.001258
25%	8.880800	1.829242	2.919620
50%	24.080457	4.135115	6.481524
75%	64.801731	7.475370	13.322674
max	423.136841	68.750352	91.792725

IV. NOISE AND JITTER

A. Jitter

One aspect that leads to a less stable signal is jitter. Jitter is the sudden, unintended variation in the position of a tracked marker. In the recordings, we see that this jitter mostly occurs when the tracked body part is either fast-moving or occluded in any way. This is mostly present when crossing arms in our recordings. As shown in Fig. 1, the jitter is clearly visible around the 20-second mark (blue, red is the motion cap-

¹A traditional Qualisys motion capture system was used as ground truth. Using infrared reflective markers, the system can track the position of the markers with sub-millimetre accuracy. These positions were used to calculate the ground truth key points for the body pose estimation.

²All measurements and their results are publicly available on GitHub: <https://github.com/Mouwrice/DrumPyAnalysis> 

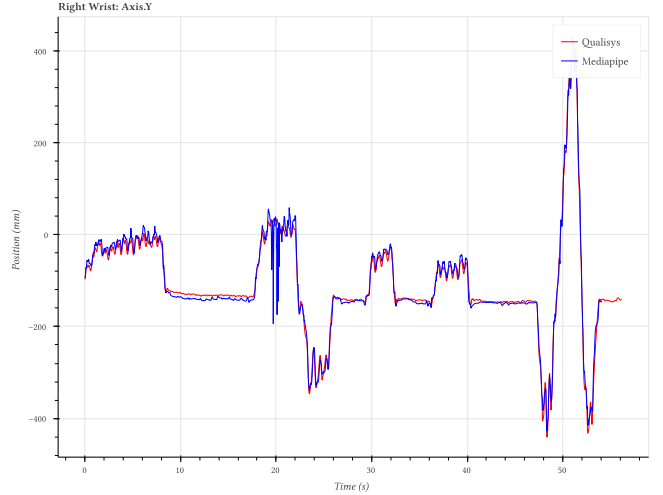


Fig. 1: A case of jitter in the maurice_drum_fast measurement around the 20 seconds mark. Model: LITE, Marker type: Landmark, Marker: Right Wrist.

tured signal). This jitter is not present in all recordings, but is a factor that can lead to a less stable signal. Jitter occurs less frequently in the larger models which make them more stable and accurate. When developing an application that relies on the stability of the signal, it is important to acknowledge that jitter can occur and that it can lead to a less stable signal.

B. Noise

Another aspect that can lead to a less stable signal is noise. Noise is the random variation in the position of a tracked marker. This noise is mostly present when the tracked body part is not moving at all. It can be seen in the trajectories that larger models produce a less noisy signal than smaller models. This is shown in Fig. 2. The noise is clearly visible in the LITE model, while the FULL and HEAVY models have a much more stable signal. It should be noted that the noise is rather small and is still in line with the accuracy values that were discussed earlier.

V. REDUCING JITTER AND NOISE

We present a post-processing method that is able to reduce the noise and jitter present in the signal. The method consists of two main steps. First, it predicts the position of the landmark based on the previous positions. Second, it compares the predicted position with the actual position and corrects the predicted position if necessary. The prediction is used to filter out noise and jitter. The method is general and can be applied to any pose estimation model.

The method principle is best explained with a figure, Fig. 3. Points *A* to *D* are the previous positions of the points. Given the previous points, the predicted position *E* is calculated. The current position *F*, the results returned from the pose estimation, is then compared to the predicted position *E*. If the dis-

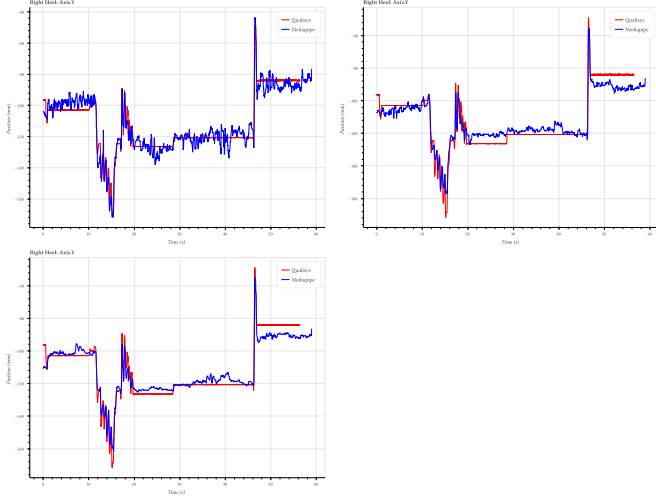


Fig. 2: A noisy signal in the `maurice_drum_regular` measurement. Models: LITE (top left), FULL (top right), HEAVY (bottom left). Marker type: Landmark. Marker: Right Heel.

tance between the predicted position E and the current position F is smaller than some threshold (e.g. the point F would lie in the striped circle around E), the current position F is considered to be caused by noise. On the other hand, if the distance is larger than some other threshold (e.g. the point F lies outside the blue circle around E), the current position F is considered to be caused by jitter. Only if the distance is between these two thresholds, the current position F is considered to be caused by the actual movement of the object. Based on this information, the current position F is corrected. F could be entirely discarded and replaced with the predicted position. However, this might lead to abrupt changes in positions. A smoother result is achieved by interposing between the predicted position E and the current position F .

A. Prediction

Since we are dealing with points that are close together in time, a simple linear prediction is sufficient. To further clarify this: we are tracking the movement of a human, of which the movements are non-cyclic and non-deterministic. This means that the movement of the person is not predictable in the long term. However, in the short term (consecutive frames) the movement is somewhat predictable. We expect the movement to be similar for consecutive frames. The higher the framerate the more accurate the prediction can be. Advanced prediction methods are therefore not necessary.

Every point has a timestamp t and a value y . For every pair of consecutive points $P_1(t_1, y_1)$ and $P_2(t_2, y_2)$ a vector \vec{v} can be computed. The vector \vec{v} is the direction from P_1 to P_2 with the length of the vector indicating the velocity of the movement: $\vec{v} = \frac{P_2 - P_1}{t_2 - t_1}$. We divide by the time difference to get the change in value per time unit, which is the velocity. This is

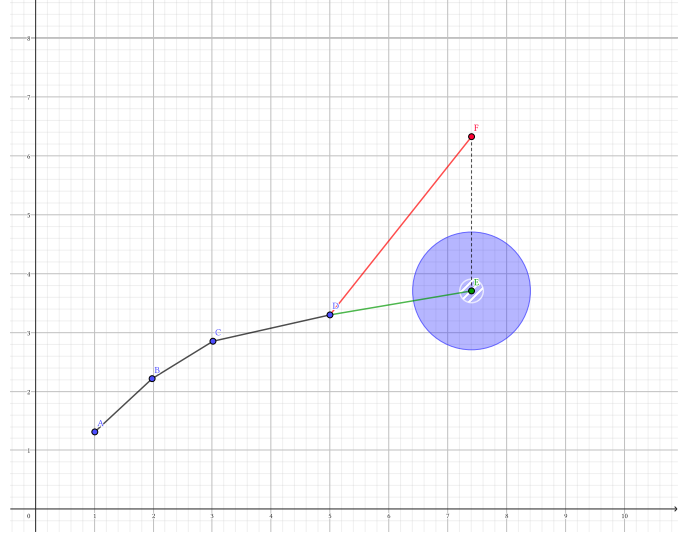


Fig. 3: Noise and jitter reduction method principle. The horizontal axis is the time axis. The vertical axis is the value of the points. Points A to D are previous positions, E is the predicted position, F is the current position from MediaPipe.

done because in a live stream setting, the time difference between frames can vary due to dropped frames, for example.

When a new position $P_3(t_3, y_3)$ is returned by MediaPipe, the predicted position P'_3 can be calculated by adding the vector \vec{v} to the last known position P_2 , multiplied by the time difference between P_2 and P_3 : $P'_3 = P_2 + \vec{v} \cdot (t_3 - t_2)$. The predicted position P'_3 is then compared to the actual position P_3 to determine if the point is caused by noise, jitter, or the movement of the object.

B. Correction

After the prediction has been made, the correction of the current position is done. The correction is based on the distance between the predicted position and the actual position. The principle idea of correction has already been discussed above. Small differences in distance indicate noise, large differences indicate jitter. But, instead of discarding these positions, we interpolate between the predicted position and the actual position.

Consider the interpolation function $f(x)$, with $x \geq 0$ the distance between the predicted position and the actual position to a value between 0 and 1: $f(x) : [0, \infty[\rightarrow [0, 1]$. A value of 0 means that the predicted position is used, a value of 1 means that the actual position is used. Take P to be the actual position, P' the predicted position and an interpolation factor b . The corrected position is then: $\hat{P} = (1 - b) \cdot P' + b \cdot P$.

The chosen interpolation function is: $f(x) = e^{-(\log_e(x))^2}$, Fig. 4. Firstly, the function is smooth, to avoid abrupt changes in the corrected position. Secondly, this function is not picked randomly. It has some nice properties. The function remains close to zero for small values of x , meaning that the predicted position is mainly used when the distance is small. This is im-

portant because small distances indicate noise. The predicted values are assumed to be less noisy than the actual values, as they are based on the previous positions. This noise reduction is why the interpolation should not reach its maximum value of 1 for small distances. The long tail of the function ensures that large distances are not entirely discarded. Because we are not entirely sure if the large distances are caused by jitter or by the movement of the object, we want to keep some of the actual position.



Fig. 4: Interpolation function. The horizontal axis is the distance between the predicted position and the actual position. The vertical axis is the interpolation factor. The blue line, $g(x) = x$, is purely a visual aid.

It is essential to understand the reasoning behind reaching a maximum value of 1. It is perfectly reasonable for the user to perform a movement that is not predicted by the method, without being an outlier. Consider a person who is standing still and suddenly starts walking. The method will predict the person to be standing still. This action by the user is not an outlier, but the method cannot predict the movement. The method should not discard the actual position in this case. Setting the maximum value of the interpolation function to 1 ensures that the actual position is not discarded in such cases.

Two parameters are used to tune the parameter function. A first parameter determines the distance at which the interpolation function reaches its maximum value. Call this parameter d . A second parameter controls the “tightness” of the interpolation around the maximum value. Call this parameter s . The interpolation function becomes: $f(x) = e^{-s \cdot (\log_e(\frac{x}{d}))^2}$ and is shown in Fig. 5.

VI. RESULTS

The `maurice_drum_fast` recording at 30 fps is used to compare the results. The method is applied to the recording and the results are compared to the original measurements.

The parameters are set to a peak $d = 0.015$, and tightness $s = 0.7$. Visually, these parameters produce the best results. The noise is reduced, and the jitter is less pronounced, while still allowing a given range of motion. To confirm that the method does, in fact, reduce these elements, we should expect to see a better signal stability. Table II shows the stability of the signal with and without processing. For the Y and Z axes,



Fig. 5: Interpolation function with $d = 0.5$ and $s = 4$. The horizontal axis is the distance between the predicted position and the actual position. The vertical axis is the interpolation factor. The blue line, $g(x) = x$, is purely a visual aid.

these are the horizontal and vertical axes, the mean is reduced by around 0.5 mm. The impact of the method on the stability is most noticeable in the percentiles. There we can observe reductions ranging from 0.5 mm to 1.8 mm. The processed signal is more stable as the outliers are reduced. The method has a positive impact on the signal stability.

TABLE II: THE SIGNAL STABILITY FROM THE MAURICE_DRUM_FAST MEASUREMENT WITHOUT PROCESSING (TOP) AND WITH PROCESSING (BOTTOM).

MODEL: LITE, MARKER TYPE: LANDMARK

Stability unprocessed (mm)	Y	Z
mean	1.814981	2.656927
std	4.700073	8.271183
min	0.000031	0.000004
25%	0.210036	0.251675
50%	0.634218	0.849195
75%	1.727039	2.558757
max	180.404176	592.605731

Stability processed (mm)	Y	Z
mean	1.419173	2.109245
std	3.663793	8.273394
min	0.000002	0.000004
25%	0.072621	0.070995
50%	0.258439	0.251171
75%	0.967466	1.169109
max	116.912891	593.312640

Lastly, we can see the reduction in jitter clearly in the Y axis, trajectory plots from the Right Wrist marker in Fig. 6. The jitter around the 20-second marker is reduced, and noise has been smoothed out. This shows that the method has a positive impact on the signal stability.

VII. THE AIR DRUMMING APPLICATION (DRUMPy)

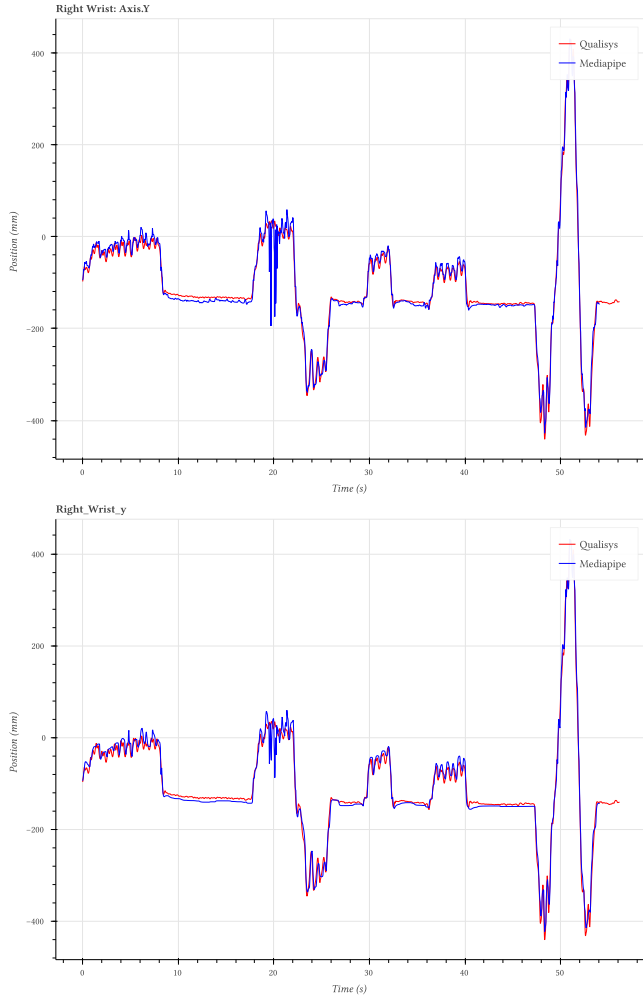


Fig. 6: The signal stability from the `maurice_drum_fast` measurement in a boxplot, without processing (top) and with processing (bottom). Model: LITE, Marker type: Landmark

The Air Drumming application is a demo application that showcases the use of on-device body pose estimation.³ The application uses the MediaPipe library to estimate the 3D pose of a person in real-time. The estimated pose is then used to detect drumming gestures and generate drum sounds. It is a fun and interactive way to explore the capabilities of body pose estimation.

The application is designed to be easy to use and understand, with a simple command-line interface and graphical user interface (Fig. 7). The user can start the application by simply launching the executable or from the command line to set some options such as which camera should be used, which model should be used etc. The application captures video frames from the camera, estimates the body pose of the person in the frame, and generates drum sounds based on the detected gestures. The user can play the drums by moving their hands

³DrumPy is the official application name, referring to the main technology, Python, with which it is made. The application and source code are publicly available on GitHub: Mouwrice/DrumPy [\[2\]](https://github.com/Mouwrice/DrumPy)

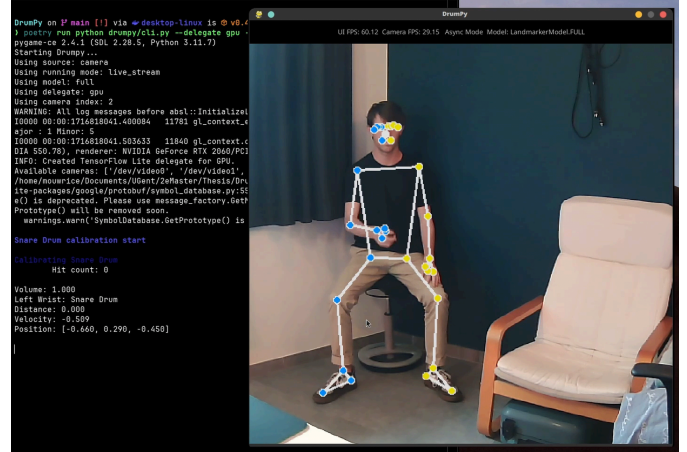


Fig. 7: The Air Drumming application in action.

and arms in the air, as if they were playing a real drum set. The feet are also tracked to detect the kick drum pedal. The application provides visual feedback by showing the estimated pose on the screen. One feature of the application is velocity-based volume control. The volume of the drum sounds is controlled by the velocity of the drumming gestures. The harder the user hits the drums, the louder the drum sounds will be. This feature adds a level of realism to the drumming experience and makes it more engaging for the user.

When launching the application, a calibration phase is initiated. During this phase, the user is asked to perform a series of drumming gestures to calibrate the position of specific drum elements such as the snare drum, hi-hat, and cymbals. This calibration step is necessary to map the detected gestures to the correct drum sounds. These steps are outputted to the user in a console that is displayed on the screen. For example, the first element to be calibrated is the Snare Drum, the user should then repeatedly hit the snare drum at the position where they want the snare drum to be. The application will then use this information to calibrate the snare drum position. After a minimum of 10 successful hits and the positions of these hits are consistent, the calibration is considered successful, and the next element is calibrated. This process is repeated for all drum elements.

For more usage information along with an installation guide, please refer to the README file in the source code repository.⁴

A. Technologies

The Air Drumming application is entirely written in Python and uses the following libraries:

- MediaPipe: For on-device body pose estimation.
- Pygame: For the audio and graphical user interface as well as capturing video frames from the camera.
- OpenCV: A library used to read video frames from a file.

⁴<https://github.com/Mouwrice/DrumPy?tab=readme-ov-file#installation> [\[2\]](https://github.com/Mouwrice/DrumPy?tab=readme-ov-file#installation)

- Click: For providing a simple and consistent command-line interface.

B. Gesture detection

The application uses the estimated 3D pose of the person to detect drumming gestures. For the demo application, the only drumming gestures detected are downward movements to hit the drums. After the hit detection, the drum element that is hit is determined by the position of the marker where a hit has been detected. The drum element closest to the marker is considered to be the drum element that has been hit. The velocity of the hit is calculated based on the speed of the downward movement. The harder the user hits the drums, the louder the drum sounds will be. The velocity controls the volume of the drum sounds, as mentioned earlier.

The hit detection is based on the observation that a downward movement is made when hitting a drum, followed by a slight upward movement. A downward movement is defined as having a consecutive series of positions where the vertical position is decreasing. An upward movement is defined as having a consecutive series of positions where the vertical position is increasing. At first, this might seem a bit counterintuitive, as one would expect the hit to be detected when the marker reaches the position of the drum element, just as in real life. In real life, the drum makes a sound when the drumstick hits the drum. However, the application is meant for air drumming, where there is no physical drum to hit. If we were to detect the hit when the marker reaches the position of the drum element, a hit might be detected when the user is still moving their hand downwards to hit the drum. This would reduce the immersion and realism. Instead, our method tries to find the point where the user expects the drum to be hit, not when the drum would actually be hit in real life. This is why the hit detection is based on the vertical trend of the marker and not the actual position of the marker.

VIII. CONCLUSION

We have shown that an average accuracy of 5-10 mm can be achieved with MediaPipe Pose for the movements relative to the camera. During the measurements, a jitter phenomenon was observed in the signal which appears to be caused by overlapping body parts such as crossed arms.

We have also shown that the signal stability can be increased by using a simple prediction model combined with an interpolation method between the predicted and the measured value. This method reduces the jitter and noise in the signal. However, the signal is still not perfect. Future work could focus on increasing the signal stability even further.

A drum application was built to demonstrate the capabilities of the system. The application uses the body pose estimation

to track the movements of the user and translate these movements to drum sounds.

The future of body pose estimation is promising. The accuracy is shown to already be sufficient to construct a real-time air drumming application. The application of body pose estimation is not limited to drumming. It can be used in various applications such as in medical rehabilitation, virtual reality, and human-computer interaction. Future work could focus on increasing the signal stability, improving the depth estimation, and finding new applications for body pose estimation.

REFERENCES

- [1] Vladimir Mandic, "Human: 3D Motion Visualization." [Online]. Available: <https://github.com/vladmandic/human-motion>
- [2] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh, "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields." 2018. [Online]. Available: <https://dl.acm.org/doi/10.1109/TPAMI.2019.2929257>
- [3] Fang, Hao-Shu and Xie, Shuqin and Tai, Yu-Wing and Lu, and Cewu, "Real-Time and Accurate Full-Body Multi-Person Pose Estimation & Tracking System." [Online]. Available: <https://github.com/MVIG-SJTU/AlphaPose>
- [4] Fang, Hao-Shu and Xie, Shuqin and Tai, Yu-Wing and Lu, and Cewu, "RMPE: Regional Multi-person Pose Estimation." 2017.
- [5] MindSpore Vision Contributors, "MindSpore Pose Toolbox and Benchmark." 2022. [Online]. Available: <https://github.com/mindspore-lab/mindpose>
- [6] Google, "MediaPipe Audio Classification." Accessed: Apr. 30, 2024. [Online]. Available: https://developers.google.com/mediapipe/solutions/vision/pose_landmarker
- [7] Google, "MediaPipe Framework." Accessed: Apr. 30, 2024. [Online]. Available: <https://developers.google.com/mediapipe/framework>