The author gives permission to make this master dissertation available for consultation and to copy parts of this master dissertation for personal use. In all cases of other use, the copyright terms have to be respected, in particular with regard to the obligation to state explicitly the source when quoting results from this master dissertation.

This master's dissertation is part of an exam. Any comments formulated by the assessment committee during the oral presentation of the master's dissertation are not included in this text.

***Abstract*—Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut.**

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

Contents

1 Introduction 2
1.1 On-device body pose estimation 2
1.2 Traditional motion capture systems 2
1.3 Motivation 3
1.4 Demo Application 3
1.5 Goals and research questions 4
1.6 Structure of the dissertation 4
2 A brief overview of the State Of The Art 5
2.1 Human Motion 5
2.2 Volumetric Capture 5
2.3 MMPose: RTMPose 6
2.4 AlphaPose 7
2.5 MindPose 8
3 MediaPipe Pose Landmarker 9
3.1.1 Features 9
3.1.2 Inference models 10
4 Measuring accuracy and deviation 14
4.1 Methods 14
4.1.1 Measurement Setup 14
4.1.2 Deviation from one time series to another 16
4.1.3 Metrics 19
4.1.4 Aligning MediaPipe to Qualisys 19
4.2 Results 24
4.2.1 Results Example 25
4.2.2 Effect of the models 28
4.2.3 Achievable framerate 34
4.2.4 Jitter 35
4.2.5 Noise 36
4.2.6 Resolution 36
4.2.7 World Landmarks 38
4.2.8 Depth Issues 40
5 Reducing Jitter and Noise 42
5.1 Method 42
5.1.1 Prediction 42
5.1.2 Correction 44
5.2 Results 45
5.2.1 30 fps 46
5.2.2 60 fps 50
5.2.3 Conclusion 52
6 The Air Drumming Application (DrumPy) 53
6.1 Technologies 54
6.2 Gesture detection 55
6.2.1 Hit detection 56
6.2.2 Finding the nearest drum element 58
6.2.3 Latency 58

6.3 Notion of Origin and Coordinate System 59
7 Future work 61
7.1 Increasing signal stability 61
7.2 Depth estimation 61
7.3 Real-time application 62
8 Conclusion 63
References 64

18047

1 Introduction

This introductory chapter provides the necessary context for this master's dissertation. It starts by presenting the concept of body pose estimation and motion capture. Next is an introduction to the demo application that will be developed as part of this project and the motivation behind it. Next,it discusses the research questions that will be addressed and the goals to be achieved. Finally, it outlines the structure of the dissertation and provides an overview of the chapters that follow.

Note that this master's dissertation is not a pure computer vision or machine learning research project. It is a project that aims to uncover some practical issues when using body pose estimation for interactive applications and finding ways to mitigate these issues. The project is a combination of research and development, with a focus on the practical aspects of using body pose estimation for interactive applications.

1.1 On-device body pose estimation

Before diving into the goals and research questions of this project, it is important to provide some context on what is meant by on-device body pose estimation. Body pose estimation is the task of inferring the pose of a person from an image or video. The pose typically consists of the 2D or 3D locations of key body parts, such as the head, shoulders, elbows, wrists, hips, knees, and ankles. In recent developments, more and more key points are commonly found in these estimation tools, sometimes with the ability to achieve complete hand and finger tracking. Body pose estimation has a wide range of applications, including human-computer interaction, augmented reality, and motion capture [1]. It can be considered a new form of motion capture based on computer vision.

On-device body pose estimation 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 is made possible by recent 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.

1.2 Traditional motion capture systems

Traditional motion capture systems are used to track the movements of actors or performers either in real-time or offline. These systems typically consist of multiple cameras that capture the movements of reflective markers placed on the actor's body. The captured data is then processed to reconstruct the actor's movements in 3D space. Motion capture systems are widely used in the entertainment industry for creating realistic animations for movies, video games, and virtual reality experiences.

1.3 Motivation

Currently traditional motion capture systems are still the most accurate and reliable way to capture human movements. However, they are expensive, require specialized equipment and expertise to set up and operate. On the other hand, on-device body pose estimation offers a more accessible and affordable alternative that can run in real-time on consumer devices. By developing a demo application that uses on-device body pose estimation for air drumming, we can explore the capabilities and limitations of this technology and its potential for interactive applications. This can help inform future research and development efforts in the field of computer vision and human-computer interaction as well as inspire new applications and use cases.

1.4 Demo Application

The project aims to develop a demo application that uses on-device body pose estimation to enable air drumming. The application allows users to play virtual drums by moving their hands in the air, as well as use their feet to press down on virtual pedals. The goal is to provide a fun and interactive experience that showcases the capabilities and limitations of on-device body pose estimation.

The main inspiration came from an older sketch performed by Rowan Atkinson as part of his Rowan Atkinson stand up tours during the years 1981 to 1986. In the clip, Rowan bumps into, what appears to be, an invisible drum kit.[1] There are no actual attributes on stage, the only thing standing on the stage is a drum stool. Various drum sounds can be heard which seem to perfectly match the movements performed by Rowan Atkinson. After the character played by Rowan Atkinson understands that he has stumbled upon an invisible drum kit, he starts playing the drums with his hands and feet. What follows is a neat trick of coordination and timing, as the sounds that we are hearing are obviously either prerecorded or performed by someone off-stage. The demo application aims to capture some of that magic by allowing users to actually play drums without the need for physical drumsticks or a drum kit.

The demo application will be developed using the MediaPipe framework, which provides a sufficiently accurate implementation of body pose estimation. The application will leverage the body pose estimation provided by MediaPipe to track the user's body movements in real-time. It will then use this information to generate drum sounds based on the user's hand and foot movements. The application will also include a graphical user interface that provides visual feedback to the user.

[1] The clip is available on YouTube from the official "Rowan Atkinson Live" channel: *https://www.youtube.com/watch?v=A_kloG2Z7tU*

1.5 Goals and research questions

As mentioned, one part of this project is to develop a demo application that uses on-device body pose estimation to enable air drumming. But that is not all. One aspect of this research is to evaluate the performance of the body pose estimation model provided by MediaPipe and identify its limitations. This will involve conducting experiments to measure the accuracy and robustness of the model under different conditions. Another goal, on top of the performance evaluation, is to provide a more pragmatic comparison when it comes to using body pose estimation versus traditional motion capture systems. During the development of the demo application, some properties of the body pose estimation have been identified that need to be considered when developing interactive applications. All of this addresses the lengthy research question: "What are the capabilities and limitations of on-device body pose estimation, specifically MediaPipe Pose, for interactive applications compared to traditional motion capture systems?"

During the measurements some signal stability issues were identified. These issues are caused by jitter and noise in the body pose estimation output. So another goal is to come up with a method that can reduce these issues. This leads to the second research question: "How can jitter and noise in the body pose estimation output be reduced or mitigated to improve the stability of interactive applications?"

1.6 Structure of the dissertation

Following this introduction, the dissertation is structured as follows:

- Chapter 2 provides an overview of the state-of-the-art in body pose estimation and motion capture, focusing on recent developments and advancements in the field.
- Chapter 3 introduces the MediaPipe framework and its body pose estimation model, highlighting its key features and capabilities.
- Chapter 4 presents the measurements that were conducted to evaluate the performance of the MediaPipe Pose model and identify its limitations.
- Chapter 5 discusses the issues of jitter and noise in the body pose estimation output and proposes a method to reduce these issues.
- Chapter 6 describes the development of the demo application for air drumming, including the design and implementation of the application.
- Chapter 7 provides some insights into future work that could be done to improve the demo application and address the limitations of on-device body pose estimation.
- Finally, Chapter 8 concludes the dissertation by summarizing the key findings and contributions of this research.

2 A brief overview of the State Of The Art

In this section, we will provide a brief overview of some recent body pose estimation tools and papers that have been published. Moreover, for every tool or paper, we will provide a short summary of the key features and limitations.

Some rather strict requirements for this specific demo application were set at the beginning of the project. The tool should be able to run on-device in real-time, and should be able to provide 3D pose estimation. By on-device, we mean that the tool should be able to run on consumer-grade hardware, such as a smartphone or a laptop. Another important requirement is the amount of detail that can be tracked. For a drumming demo application, it is essential that the hands and feet are properly detected and tracked. In many of the papers that we will discuss in this section, the tools often lack one of these requirements.

2.1 Human Motion

The Human Library is an open-source tool, based on web technologies, that provides a wide range of detection tasks. One of which is body pose tracking. In full the name is “Human: AI-powered 3D Face Detection & Rotation Tracking, Face Description & Recognition, Body Pose Tracking, 3D Hand & Finger Tracking, Iris Analysis, Age & Gender & Emotion Prediction, Gaze Tracking, Gesture Recognition” [2]. It also comes with a library, human-motion, which is focused on the 3D motion visualisation of the face, body, and hands [3].

Many of the requirements for this project are met by the Human Library. It has an excellent amount of detail that can be tracked, and it can run on-device. However, during testing, we could not get the performance to be high enough for real-time applications. The body pose was only updated every second or so. Even if some setup optimizations could be made on our side, we do not believe that the performance could be improved enough to be used in a real-time application.

2.2 Volumetric Capture

This tool is quite different from the other tools in this section. It is not really a body pose estimation tool, as it does not explicitly provide any precise markers or skeleton. Instead, it provides a volumetric representation of the human body [4]. This means that it provides a 3D model of the human body. It does so with a set of calibrated depth cameras, such as the Intel RealSense camera. It requires quite a lot of specific hardware and multiple cameras, so it clearly does not conform with the set requirements. This tool is also not really suitable for our application, as we need precise tracking of the hands and feet using traditional markers. However, it is an interesting tool that could be used in other applications. For example, it has potential in the use of Mixed Reality applications. An overview of the Volumetric Capture tool can be seen in Figure 1.

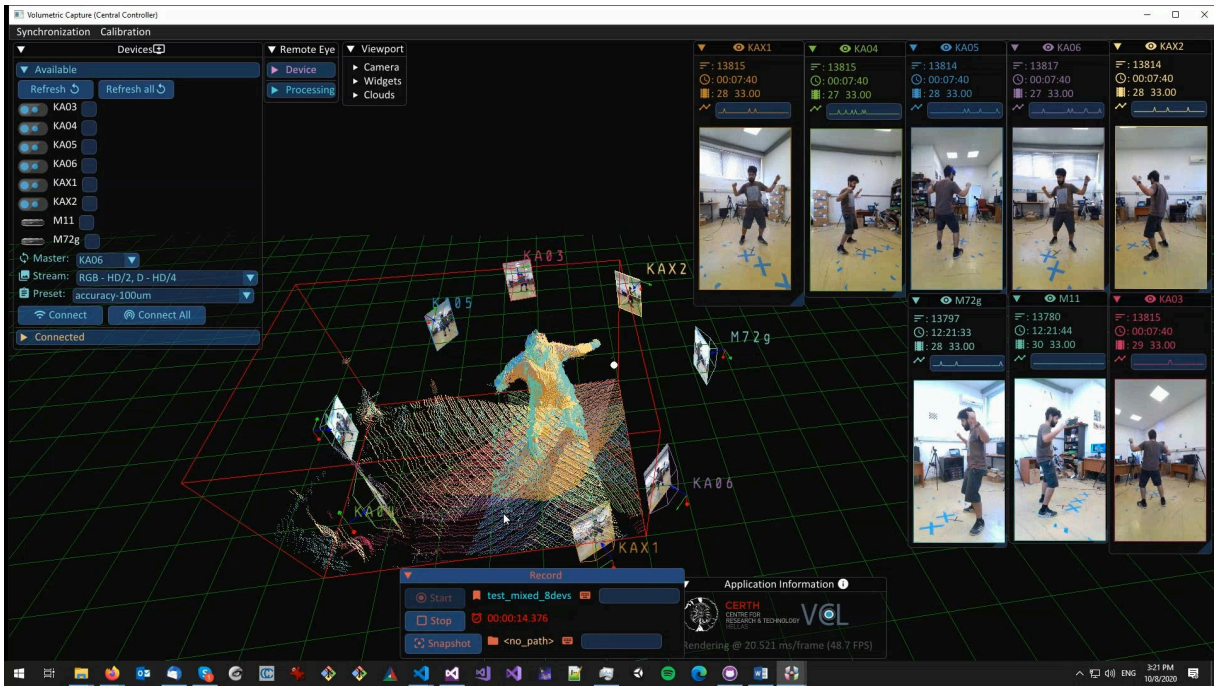


Figure 1: "The Volumetric Capture tool overview taken from the official documentation [4]."

Volumetric Capture was developed by the 3D vision team of the Visual Computing Lab.[2] Unfortunately, according to the documentation and releases on GitHub, the project seems to be abandoned.[3] The last release was in 2020 and is only available on Windows 10. This limited compatibility and support appears to be common for many of the tools in this section. Many tools go along with some research paper, but once the research is done, the tool is abandoned. This lack of updates and maintenance is less than ideal for use in an actual application.

2.3 MMPose: RTMPose

Figure 2: RTMPose official logo.

[2] *https://vcl.iti.gr/*

[3] *https://github.com/VCL3D/VolumetricCapture/releases*

To quote from the official GitHub page: “MMPose is an open-source toolbox for pose estimation based on PyTorch. It is a part of the OpenMMLab project.”[4] [5]

The model of interest in the collection of models is RTMPose [6]. RTMPose is a pose estimation toolkit that works in real-time and with multiple persons in the frame. The model is quite fast and can run on consumer-grade hardware. They boast with achieved frame rates of 90+ FPS on an Intel i7-11700 CPU and 430+ FPS on an NVIDIA GTX 1660 Ti GPU. However, it does not provide 3D pose estimation, it is limited to two dimensions. Despite this, it is available on many different platforms and devices such as Windows, Linux, and ARM-based processors. This tool ticks many of the boxes for our project, but the lack of 3D pose estimation was a dealbreaker. But, as shown in the measurements of MediaPipe Pose in the next section, the 3D pose estimation is not always as accurate as one might hope. It is even so that the depth, the third dimension, is not used in the final application. Given the uncovered limitations of the MediaPipe Pose tool, RTMPose might be a better alternative for our usecase. Especially since it has a way higher achievable frame rate than MediaPipe Pose.

2.4 AlphaPose

AlphaPose is yet another open-source, multi-person pose estimator [7], based on the research paper “RMPE: Regional Multi-person Pose Estimation” [8]. It is one of the earlier body pose estimation tools originating from 2017. Being an older tool, its accuracy and performance are not as high as some of the newer tools, such as RTMPose. Besides, it is one of those tools that is not maintained once the research paper has been published, making it unfit for actual usage.

Figure 3: The AlphaPose logo.

[4]The OpenMMLab project is a collection of “open source projects for academic research and industrial applications. OpenMMLab covers a wide range of research topics of computer vision, e.g., classification, detection, segmentation and super-resolution.” *https://openmmlab.com/*

2.5 MindPose

MindPose is the last tool that we will discuss in this section. It is the result of an open-source project jointly developed by the MindSpore team.[5]

[5]MindSpore is an open-source AI framework developed by Huawei. It is a deep learning and machine learning framework that is used for training and inference of AI models. *https://www.mindspore.cn/*

3 MediaPipe Pose Landmarker

The following section describes the MediaPipe Pose Landmarker solution [9]. It first provides some context about the broader MediaPipe Framework before going into more detail on the MediaPipe Pose Landmarker solution. The features of the solutoin are discussed as well as the motivation behind the choice of this solution for the air drumming application. The inference models used in the solution are also briefly discussed, including BlazePose and GHUM.

MediaPipe is a collection of on-device machine learning tools and solutions by Google [10]. It consists of two main categories. There are the MediaPipe solutions, which have predefined "Tasks" that are ready to be used in your application [11]. There are tasks on vision such as the detection and categorisation of objects or the detection of hand gestures [12], [13]. One of these vision solutions is the MediaPipe Pose Landmarker. Other solutions such as text classification and audio classification are also present [14], [15]. On the other hand, exists the MediaPipe Framework. It is the low-level component used to build efficient on-device machine learning pipelines, similar to the premade MediaPipe Solutions [16]. The remainder of this thesis solely addresses the MediaPipe Pose Landmarker Solution from this point forward, and will frequently be referred to as simply "MediaPipe" or "MediaPipe Pose" for the sake of conciseness.

MediaPipe Pose is available on three different platforms. One can use it in Python, on Android and on the web. However, these are only just API's to interact with the actual detection task. The application presented in this thesis is completely written in Python but all the concepts that are discussed are applicable to any platform.[6]

3.1.1 Features

The main feature of MediaPipe Pose is of course, just as all body pose estimation tools, to extract the body pose from a given image or video frame. Unlike many other body pose estimation tools, MediaPipe delivers a 3D estimation instead of the more common 2D estimation, by introducing depth. However, in the measurement results, this added depth dimensionality is shown to be less than ideal.

The MediaPipe Pose Landmarker solution matches all the requirements set at the beginning of the project. It can run on-device in real-time and provides 3D pose estimation. It only requires a single camera to operate and can run on consumer-grade hardware. A laptop with a webcam is sufficient to run the application. The hands and feet are properly detected and tracked, which is essential for the drumming application. The MediaPipe Pose Landmarker solution is also quite fast and can run in real-time. The MediaPipe Pose

[6] Following the announcement at Google I/O 2024, MediaPipe is now part of the larger Google AI Edge collection of tools and solutions. *https://ai.google.dev/edge*

Landmarker solution is a perfect fit for the air drumming application. Not only does it meet all the requirements, it is one of the few tools currently available that is ready to be used in a production environment. MediaPipe is not just a research tool; it is a tool that is actively maintained and updated by Google. This is a big advantage over many other tools that are often abandoned once the research paper has been published.[7] All of these reasons are why MediaPipe has been chosen for this project.

MediaPipe has three modes of operation, called the `RunningMode`. MediaPipe can work on still images (`IMAGE`), decoded video frames (`VIDEO`) and a live video feed (`LIVE_STREAM`) [9]. Using the live video feed mode has some implications. When running MediaPipe in a real-time setting, the inference time of the model is constraint by this real-time application. When frame inference takes too long to be in the time window the frame gets dropped. Another major aspect of real-time applications is that the inference should not block the main thread and halt the program. This is why the inference in the `LIVE_STREAM` mode is performed asynchronously and results are propagated back using a callback function.

One other feature other than returning the body pose is the creation of an image segmentation mask.[8] MediaPipe has the ability to output a segmentation mask of the detected body pose. This mask could be used for e.g. applying some visual effects and post-processing, but it is not of much use in this implementation.

3.1.2 Inference models

MediaPipe Pose is based on two computer vision models for the inference of the body pose. The first one is BlazePose which is only designed to return two-dimensional data in the given frame [17]. A second is the GHUM model, a model that captures 3D meshes given human body scans. A synthetic depth is obtained via the GHUM model fitted to the 2D points [18].

The Pose task consists of 3 `.task` files that contain the actual detection task and models. Three models are available: Lite (3 MB size), Full (6 MB size) and Heavy (26 MB size). The measurement results will prove that the larger models can provide a more accurate and correct result but at reduced inference speeds. There is a trade-off to be made between accuracy and real-time processing. Although the Heavy model is almost 9 times larger than the Lite model, the improvement in accuracy is a lot smaller. The Lite model is not considerably worse and produces fine results for the drumming application.

[7] Following the announcement at Google I/O 2024, MediaPipe is now part of the larger Google AI Edge collection of tools and solutions. This once again confirms that MediaPipe is ready to be used in a professional and commercial setting. *https://ai.google.dev/edge#mediapipe*

[8] A segmentation mask is a grayscale image, sometimes just pure black and white, with the goal of partitioning the image into segments. For example, the MediaPipe segmentation mask colours all pixels white where the human silhouette is visible.

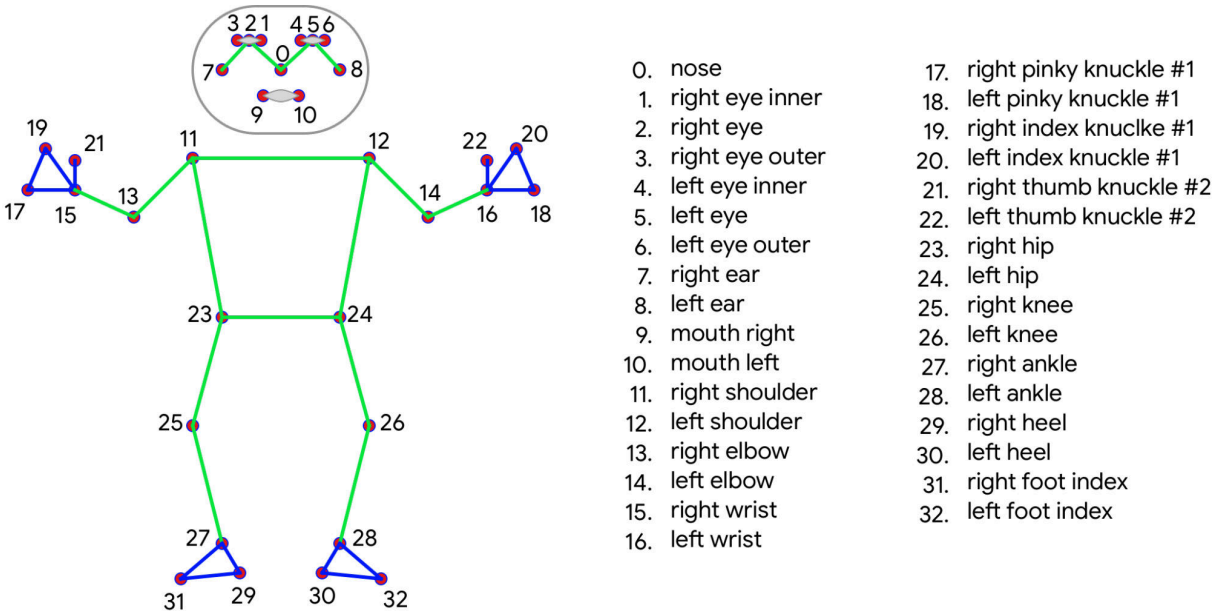


Figure 4: BlazePose 33 landmark topology with the original COCO landmarks in green.

3.1.2.1 BlazePose

BlazePose provides human pose tracking by employing machine learning (ML) to infer 33, 2D landmarks of a body from a single frame [17]. A standard for body pose originating from 2020 is the COCO topology [19]. The COCO 2020 Keypoint Detection Task is a challenge to develop a solution to accurately detect and locate the keypoints from the COCO dataset. The original COCO topology only consists of 17 landmarks. With little to no landmarks on the hands and feet. BlazePose extends this topology to a total of 33 landmarks by providing landmarks for the hands and feet as well. These added landmarks are crucial for the drumming application and is part of the reason MediaPipe was chosen.

Next some parts of the BlazePose design and tracking model are discussed. It is important to understand some underlying methods discussed here as it helps to interpret the measurement results. A complete and detailed overview is provided in the original paper [17].

The pipeline of MediaPipe is displayed in Figure 5. Before predicting the exact location of all keypoints the pose region-of-interest (ROI) is located within the frame using the *Pose detector*. The *Pose tracker* then subsequently infers all 33 landmarks from this ROI. When in `VIDEO` or `LIVE_STREAM` mode, the pose detection step is only run on the first frame to reduce the inference time. For any subsequent frames, the ROI is then derived from the previous points.

As this solution is intended to be used in a real-time setting, the inference needs to be within milliseconds. The MediaPipe team have observed that the position of the torso is most easily and efficiently found by detecting the position of the face. The face has the most high-contrast features of the entire body and thus results in a fast and lightweight

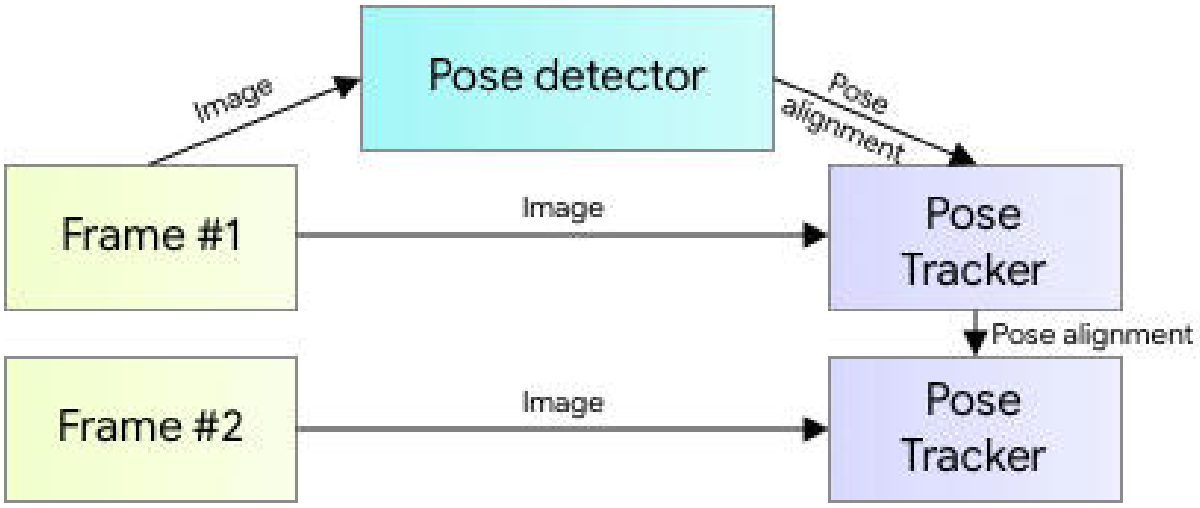


Figure 5: MediaPipe Pose estimation pipeline

inference compared to the rest of the body [20]. This, of course, has the logical consequence that the face should be visible within the frame. This means that the "drummer" has to face the camera head on for the best result.

After detecting the face the construction of the pose region-of-interest begins. The human body centre, scale, and rotation are described using a circle. This circle is constructed by predicting two virtual points. One point being the centre of the hips, the other lies on the circle as the extension of the hip midpoint to face bounding box vector. These two points should now describe a circle as shown in Figure 6.

Lastly, we very briefly describe the actual network architecture for the tracking of keypoints. The model takes an image as input with a fixed size of 256 by 256 pixels and 3 values for each pixel providing RGB values, Figure 7. The model uses "a regression approach that is supervised by a combined heat map/offset prediction of all keypoints" [17], [21].

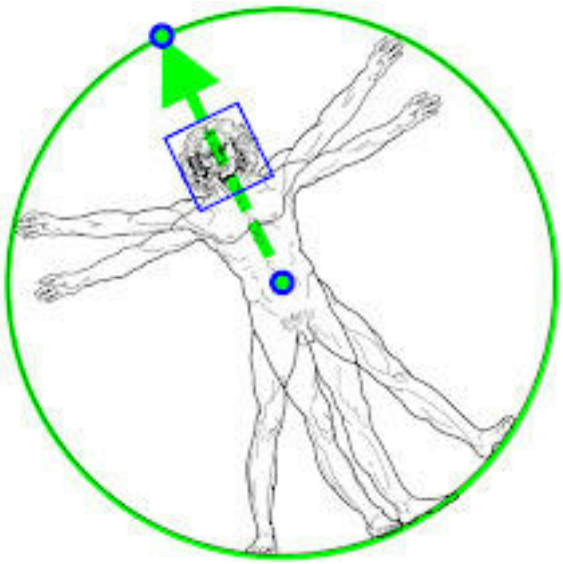


Figure 6: MediaPipe Pose detection and alignment using virtual keypoints.

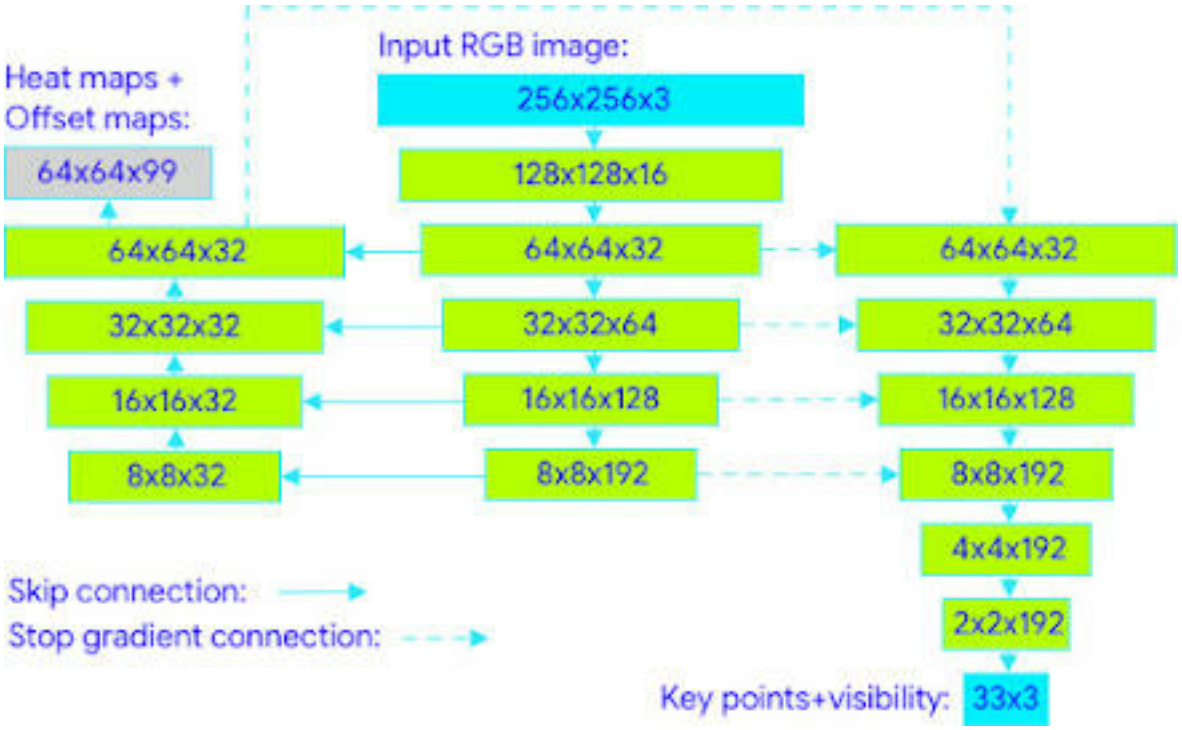


Figure 7: MediaPipe Pose network architecture.

The left-hand side outputs the heat maps and offset maps. Both the centre and left-hand side of the network are trained using the provided heatmap and offset loss. Afterwards, the heatmaps and output maps output layers are removed and the regression encoder on the right-hand side is trained. The fixed input layer size indicates that there is no use in increasing the resolution to get an increase in the prediction accuracy. The measurements also confirm this hypothesis.

3.1.2.2 GHUM (Generative 3D Human Shape and Articulated Pose Models)

As mentioned, MediaPipe Pose offers an extra dimension unlike many other purely 2D pose estimation tools. This third dimension is, of course, depth. It does so by utilising an entirely different model, being the GHUM model [18]. The GHUM model can construct a full 3D body mesh given image scans of a person. The outputs form an actual 3D mesh of 10,168 vertices for the regular model and 3,194 vertices for the lite model. MediaPipe is quite vague on the exact usage of this model in the pose solution. The only mention of GHUM is in the following sentence: *"Keypoint Z-value estimate is provided using synthetic data, obtained via the GHUM model (articulated 3D human shape model) fitted to 2D point projections."* [22]. Nonetheless, in the measurements it is shown that there is some notion of depth, but it is far from accurate. Especially when compared to the other 2D values provided by the BlazePose model.

4 Measuring accuracy and deviation

The following chapter discusses the accuracy and deviation of a body pose estimation tool. More specifically, the outputs of the MediaPipe Pose landmark detection solution are compared to that of a traditional optical tracking system. All base measurements are taken from a Qualisys Oqus MRI[9] system setup at the Art and Science Interaction Lab at De Krook in Ghent, provided by IDLab-Media[10].

4.1 Methods

To be able to get indications of the accuracy of MediaPipe Pose, a baseline, or base "truth", needed to be established. Using the aforementioned Qualisys MoCap setup at ASIL we can track marker locations at sub-millimetre accuracy, which are then taken as ground truth.

4.1.1 Measurement Setup

The setup is configured to be in line with the expected environment the drum application will be used in. Being, a single person sat on a chair facing a webcam, performing drumming motions. The person performing the movements has infrared reflective trackers positioned on the body as close as possible to where the MediaPipe Pose landmarks are located. This setup is shown in Figure 8.

The Qualisys Motion Capture setup first requires a calibration step, after which markers can be tracked with a precision of up to 0.3 mm. In later measurements, the accuracy has dropped to 0.8 mm, which can be attributed to calibrating a larger volume of space. For our use case, this level of accuracy is still sufficient. As we will see, the MediaPipe accuracy will not get close to an accuracy of 0.8 mm, so the Qualisys accuracy is more than sufficient. After applying the reflective markers, motion capture can easily be performed using the Qualisys Track Manager (QTM) program[11]. After having labeled each marker and trajectory, these can be exported to a TSV (tab separated values) file [23]. One such a file looks like the snippet below, Listing 1. First, some context is provided on the measurement. After the listing of the marker names and their trajectory types, follows the actual trajectory. Every line in the file then describes the 3D position of every marker per frame. Every three floating point numbers are the x, y and z coordinates of a given marker, in the same order as they are listed.

[9] The Oqus MRI is one of Qualisys' traditional optical motion capture cameras, requiring physical trackers on the body that reflect incoming infrared light from the camera. *https://www.qualisys.com/cameras/oqus-mri/*

[10] The Art and Science Lab is a *" highly modular research infrastructure aimed at interaction research"*, provided by IDLab-Media, one of the research teams within the research group IDLab from Ghent University and imec. *https://media.idlab.ugent.be/about/*

[11] *https://www.qualisys.com/software/qualisys-track-manager/*

Figure 8: A frame from the camera recording showing the measurement setup.

The video recordings are captured with a regular webcam with a maximum resolution of 1080p and a frame rate of 30 frames per second. The camera placement needs to be in line with the forward facing axis of the Qualisys captures to reduce any deviations introduced by being off-axis. Afterward, the videos get processed frame by frame with MediaPipe, using the *VIDEO* running mode[12]. The video mode is an offline processing mode, meaning that no frames will be dropped to satisfy a live stream constraint. This mode allows us to

```tsv
NO_OF_FRAMES  6916
NO_OF_CAMERAS 13
NO_OF_MARKERS 10
FREQUENCY 120
NO_OF_ANALOG  0
ANALOG_FREQUENCY  0
DESCRIPTION --
TIME_STAMP  2024-04-15, 10:39:38.777  406646.66137317
DATA_INCLUDED 3D
MARKER_NAMES  Wrist_L Hip_L Foot_Index_L
TRAJECTORY_TYPES  Measured  Measured  Measured
-473.521  191.980 613.382 -677.077  164.814 615.497 -138.493  187.190 53.579
-473.564  192.348 613.040 -677.061  164.822 615.494 -138.494  187.229 53.555
-473.646  192.696 612.779 -677.045  164.794 615.445 -138.465  187.193 53.464
-473.683  192.890 612.513 -676.988  164.782 615.436 -138.448  187.252 53.482
```

Listing 1: A snippet of what a TSV output from a Qualisys tracked measurement looks like.

[12] *https://developers.google.com/mediapipe/solutions/vision/pose_landmarker/python#video*

```csv
frame,time,index,x,y,z,visibility,presence,landmark_type
2,53,0,0.4807698130607605,0.22160261869430542,-0.360921174287796,0.99999964237213
2,53,1,0.4906315207481384,0.20818790793418884,-0.33244213461875916,0.999999165534
2,53,2,0.4961097836494446,0.2083108127117157,-0.33247220516204834,0.9999989271163
2,53,3,0.49966344237327576,0.20876441895961761,-0.33247825503349304,0.99999880790
2,53,4,0.4742940366268158,0.2077968716621399,-0.3331023156642914,0.99999892711639
2,53,5,0.4689987897872925,0.2078893780708313,-0.3331734538078308,0.99999892711639
2,53,6,0.4639938175678253,0.20820003747940063,-0.3331492245197296,0.9999990463256
2,53,7,0.5079091787338257,0.21490448713302612,-0.17037953436374664,0.999997854232
2,53,8,0.46058356761932373,0.21451136469841003,-0.17510229349136353,0.99999868869
2,53,9,0.49152788519859314,0.23765860497951508,-0.30100592970848083,0.99999976158
```

Listing 2: A snippet of the CSV output taken from MediaPipe processed frames. Lines are truncated.

measure models that we otherwise would not be able to run at a constant 30 frames per second. The result of every processed frame is then written to a CSV file. Every line in the file contains all the known information of one marker at the given frame. Resulting in the following format, displayed in Listing 2: *frame number*, *time* (in milliseconds), *index* (of the marker), *x*, *y*, *z*, *visibility*, *presence*, *landmark_type* (either 0 to indicate a `Landmark`, or a 1 to indicate a `WorldLandmark`).

4.1.2 Deviation from one time series to another

Before describing how the outputs from the previous section are used to come up with accuracy measurements, a brief overview of how deviation from one time series to another is derived, is needed. We want to compare the trajectories from the Qualisys capture with that of the MediaPipe captures. These trajectories consist of discrete-time points. As the frame rate of both systems differ, the discrete-time points are unaligned.

Figure 9 provides a visual representation of the problem. We have two time series to compare. Series f in red and series g in blue. Due to these points having a different frequency and unaligned time stamps, it is not possible to compare these just by iterating over both series at the same time. Where an iteration corresponds to jumping to the next point in time in both series.

One option would be to "walk" over the series based on the time difference between points. The walk starts by selecting the first point of each series, A and K in this example. Then there are three options:

1. Only jump to the next point in the first series
2. Only jump to the next point in the second series
3. Jump to the next point in both series

The option that minimizes the time difference between the selected points is chosen. This method ensures that every point in both series is taken into account. The pairs of points

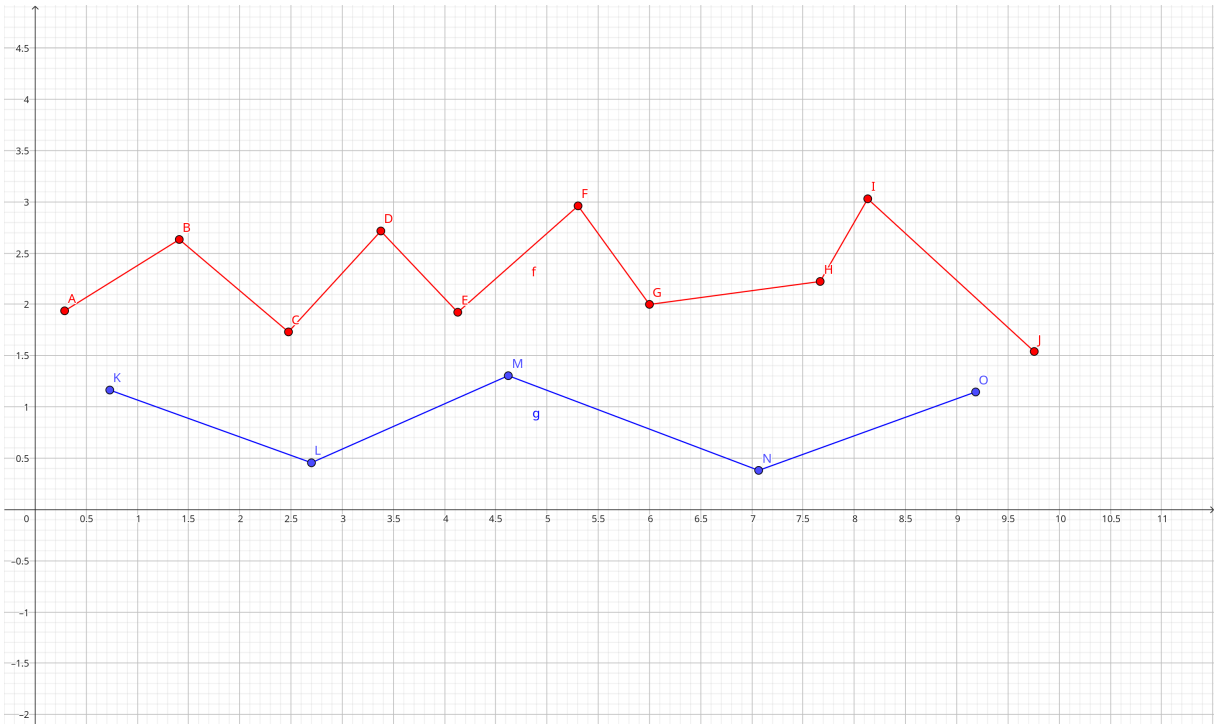


Figure 9: Example of unaligned time series, the horizontal axis being time, the vertical axis would be some value

that would be selected by this technique is displayed in Figure 10 by the dotted lines. The measured difference between both signals is then the sum of the length of all dotted lines (the difference between point pairs) divided by the amount of dotted lines (pairs).

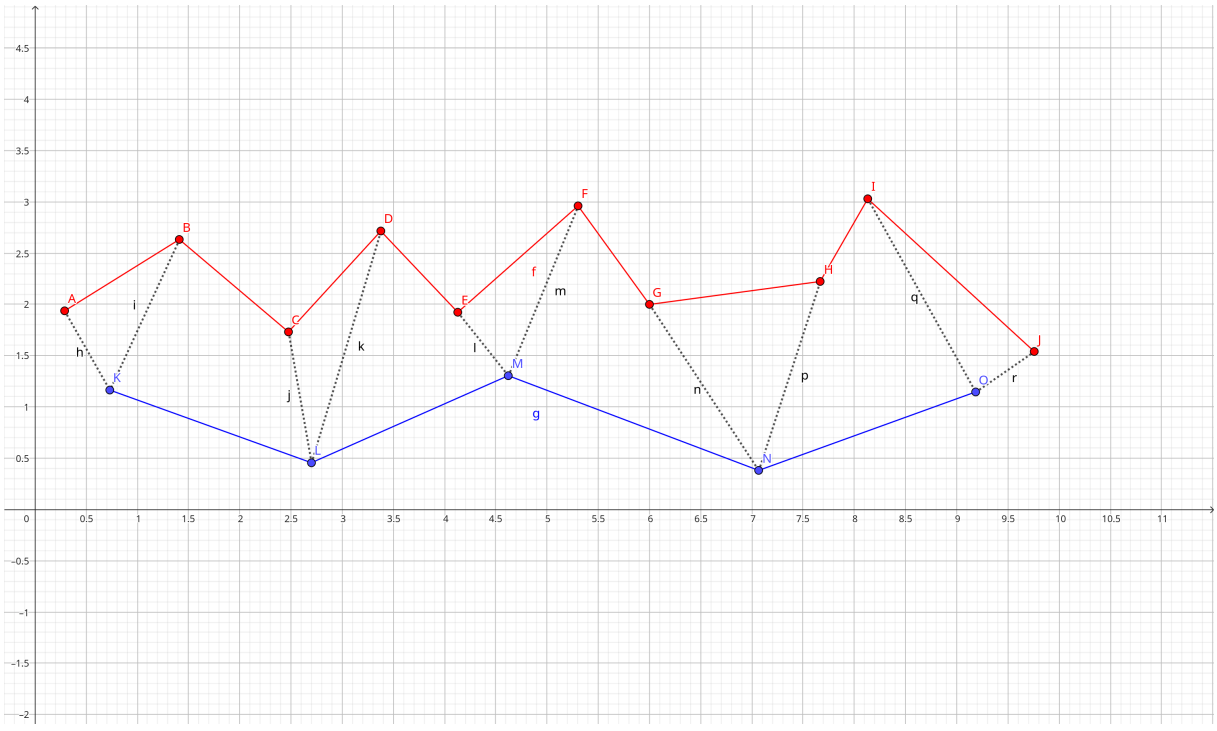


Figure 10: Point pairs selected by walking over both series indicated by the dotted lines.

However, it is clear to see that this method will result in a higher deviation when comparing a high frequency signal against a lower frequency signal. Which is the case in our measurements. A high frequency Qualisys measurement (100Hz or more) is compared against a lower frequency MediaPipe measurement (30Hz). Because we know these differences in frequency, we are not interested in a difference method that takes these into account. The goal of the measurements is to find the average deviation of a point generated by MediaPipe with that of the corresponding point generated by the Qualisys capture. The following new method achieves that goal.

Instead of comparing every point in both series, a dominant series is selected. This simple method iterates over the dominant series and for each point in the dominant series finds the corresponding point in the other series. The corresponding point being the point with the timestamp closest to that of the dominant point. Adapting the behavior of the "walk" to accommodate this idea of having a dominant time series, would result in the following procedure:
For each point in the dominant time series, walk over the points of the other series until the difference in time stamps with the dominant point no longer decreases. Then we have found a new pair. Results of this procedure are displayed in Figure 11.

With this method, we have that the irrelevant points are no longer included in the measurements. Yet this does not mean that the points that are now taken into account are perfectly aligned with one another. However, due to the high frequency of the Qualisys

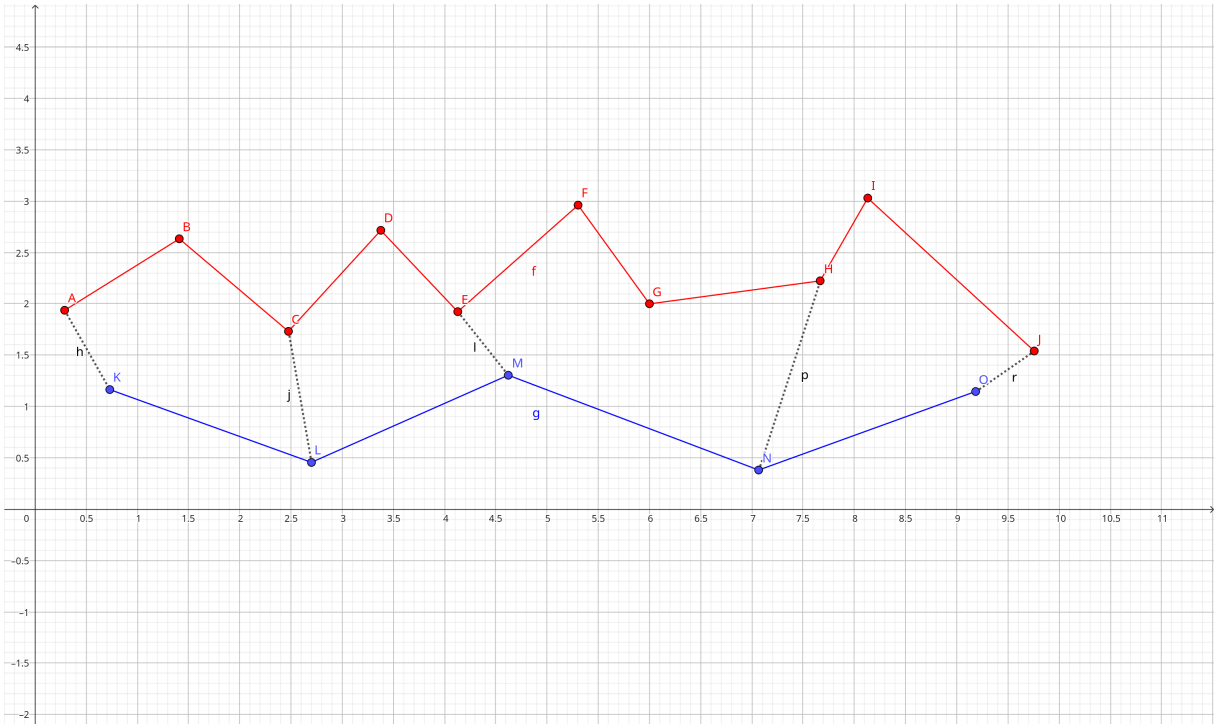


Figure 11: Point pairs selected by iterating over the dominant series (g, blue) and walking over the other (f, red), indicated by the dotted lines.

captures, we note that the time difference will be relatively small, and thus the difference induced by this nonalignment will be quite small. For a Qualisys capture with a frame rate of 120Hz, there is a frame every $\frac{1000\text{ ms}}{120\text{ Hz}} = 8.33...$ milliseconds. In the worst case, the MediaPipe point lies exactly between two Qualisys frames, resulting in a maximum time difference of only 4.166... milliseconds.

4.1.3 Metrics

Two simple metrics can now be computed having found a way to get point pairs. A first one is the average offset of one signal to another. By simply taking the average difference of each point pair. A second and very similar metric is the average deviation. It denotes how much a signal deviates from another one. Instead of taking the regular difference of a point pair, the absolute difference is taken. This is an important distinction. The average offset is simply a metric that tells us how far a signal is from another. The average deviation, however, can be interpreted as the accuracy with which a signal approximates another signal. So when the Qualisys signal is seen as the base truth and the MediaPipe signal is seen as an approximation of that base truth, we have an accuracy measure for the MediaPipe signal!

Another important metric is the signal stability. For instance, if the MediaPipe signal is very stable, but has a high average deviation, it is still a good approximation of the Qualisys signal. This is because the MediaPipe signal is consistently off by the same amount. This is an essential metric for interactive applications. If the signal is not stable, the application will not be able to provide a consistent experience to the user. The stability of a signal can be measured by the standard deviation of the deviation. A low standard deviation means that the signal is stable, a high standard deviation means that the signal is not stable. Another way to measure the stability of a signal is by taking the differences in deviation between consecutive points. A stable signal should have a low difference in deviation between consecutive points. It is important to note that the stability of a signal is not the same as the accuracy of a signal. A signal can be very stable but not accurate, and vice versa.

After this slight detour, follow the methods used to align the MediaPipe results to the Qualisys captures.

4.1.4 Aligning MediaPipe to Qualisys

Using the setup described in Section 4.1.1 we have two time series that can be compared using the method from the previous section. However, before that is possible there is still one major issue that needs to be solved. The points from the MediaPipe result originate from a totally different axis and origin point. The Qualisys captures have their origin point calibrated on the floor, with the x-axis being the forward facing axis, the y-axis the

horizontal axis and the z-axis the vertical axis. The MediaPipe Landmarks, on the other hand, do not actually correspond with a point in space but rather with a point in the video frame and an associated depth. For the `Landmark` markers the origin point in this MediaPipe case is the left corner of the video frame. The x-axis is the horizontal axis along the frame, the y-axis is the vertical axis along the frame, and the z-axis is the depth from the camera. MediaPipe also has a different kind of Landmarks (`Landmark`), namely the World Landmarks (`WorldLandmark`). These try to map the regular Landmarks, which are a point in the video frame, to a point in space. With the center of the hips taken as origin. The axis remain the same but are scaled so that the `WorldLandmarks` are in line with the actual size of movement in the space.

This section gives a complete overview on how the MediaPipe signal has been aligned to match the Qualisys signal and provide a proper measurement on accuracy. The entire section uses a measurement of the vertical position of the left wrist marker, taken from MediaPipe using regular `Landmarks` and the `FULL` model.

The output from the recordings are time series that can be plotted. The output from the measurements is read and without any processing plotted to a line plot in Figure 12. On the horizontal axis is time in seconds. On the vertical axis is the value of the point in time in millimeter. Figure 12 shows a clear mismatch in axis. Plotted is the z-axis from both capture systems. But as mentioned, in MediaPipe the z-axis is the depth and not the vertical axis.

The problem of mismatched axes is a very simple one to solve. Before plotting the MediaPipe signal we switch the axis so they match with the actual direction of axis in the Qualisys recording. The MediaPipe axes are thus mapped as follows:

- $x \rightarrow y$
- $y \rightarrow z$
- $z \rightarrow x$

As can be seen in Figure 13, we now have plotted the proper vertical axis. One might have noted the pretty nonsensical values of the MediaPipe signal. For one, they are negative. Whereas moving up corresponds to an increase in value with the Qualisys captures, the inverse is true for the MediaPipe results. This requires us to re-invert the vertical values during analysis, resulting in negative values. Normally `Landmark` values would be in the range [0, 1], 0 being one side of the video frame, 1 the other side. But these values are inverted by multiplying by -1 resulting in a new range of [0, -1]. Because this inverting method should also work for `WorldLandmarks` which have no predefined range of values it is not possible to invert the values using the following method: $x \Rightarrow 1 - x$.

Secondly, all MediaPipe values have been multiplied by 1000. As the Qualisys output is in millimeter we already prepare the MediaPipe signal by interpreting the incoming signal

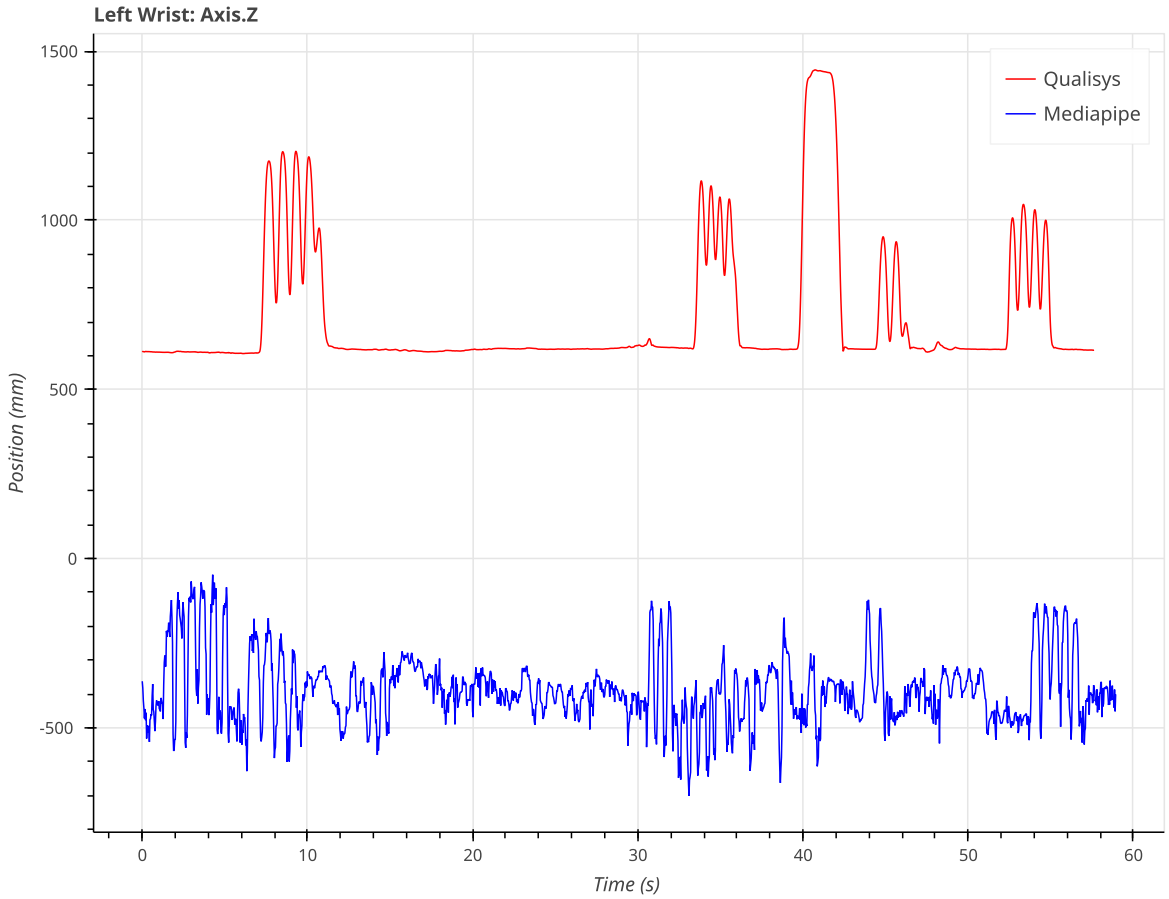


Figure 12: Plot of the MediaPipe (Blue) and Qualisys (Red) left wrist z-axis without processing.

as meter and converting it to millimeter. The MediaPipe `Landmark` signal has no real unit of course but interpreting it as meters allows for a simpler interpretation when it comes to scaling, explained later in this section.

Now that the basics are out of the way we can start aligning the signal. A first step is removing the average offset the MediaPipe signal has to the Qualisys signal. The method of walking over the dominant series (MediaPipe in this case) and gathering pairs of points from both series as discussed in the previous section, Section 4.1.2, is used for this. For every pair of points we can simply take the difference between those points. The average of these differences is then the offset of the MediaPipe signal. The result of removing this offset from the MediaPipe signal is displayed in Figure 14.

With the two signals close together another problem becomes apparent. They are offset in time. This makes sense as both measurements cannot easily be started at exactly the same time. We need to introduce a starting offset. This starting offset should minimize the deviation between both signals. This is achieved by iteratively increasing a starting offset and capturing the offset that resulted in the least deviation. In Figure 15 it is shown that this method finds the most perfect offset. Both signals are perfectly aligned in time.

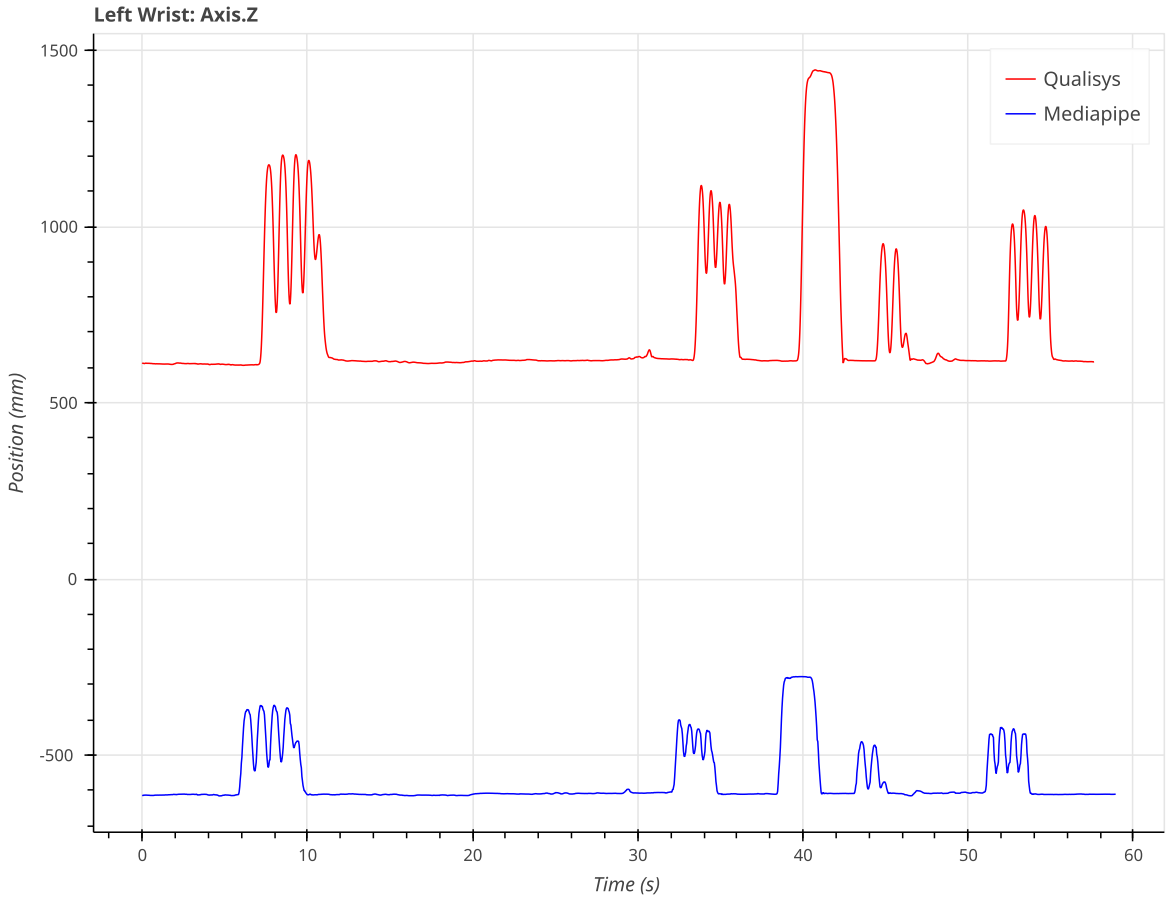


Figure 13: Plot of MediaPipe (Blue) and Qualisys (Red) left wrist z-axis after re-arranging the MediaPipe axes.

After this offset operation the average vertical offset is computed again and subtracted from the signal.

The final and most intricate part of the alignment is getting the scaling right. As we can see in the previous plots the scale of the signal is not at all correct. Here a scaling factor needs to be found that minimizes the deviation. There is one caveat, we cannot simply scale the signal by multiplying it with a given factor. This would scale the signal away from the origin point. One can see that, in fact, we should "stretch" the signal vertically to make it align. In other words, the signal needs to be scaled around the center point of the signal that it is being aligned to.

The center point of the signal is easily calculated as the average of the signal's values. The stretching of the signal then goes as follows: For every original point of the signal, take the difference between that point and the center point of the other signal. Scale the difference by the scaling factor. The new stretched point is now the center point plus the scaled difference.

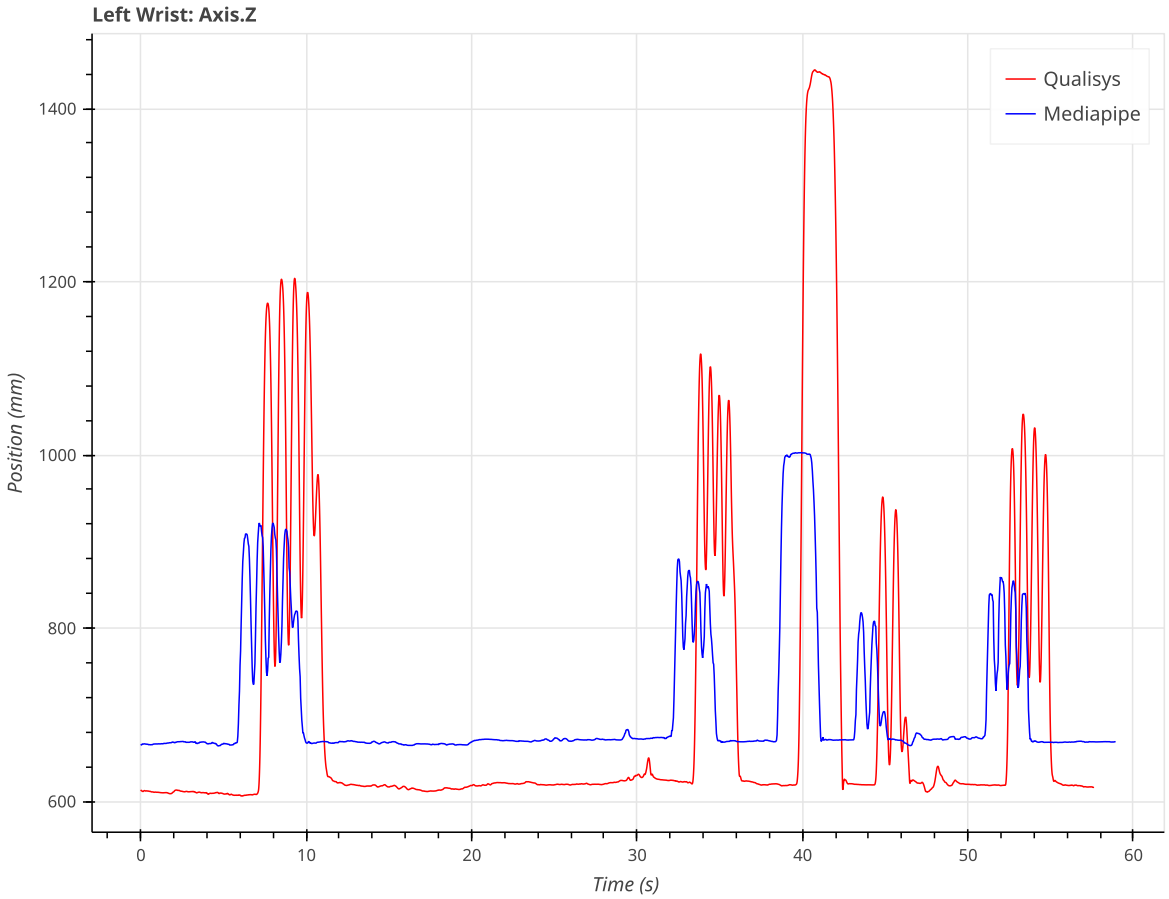


Figure 14: Plot of the MediaPipe (Blue) and Qualisys (Red) left wrist z-axis with the average offset removed.

The previous method of finding the optimal time offset is a simple one. Since it is a discrete problem, the optimal value can easily be found by testing all possible values. The optimal scale, however, is not a discrete value. To find the optimal scaling factor we need an optimization algorithm.

The optimization problem at hand can be solved using Golden-section search [24].[13] It is a technique for finding an extremum (minimum or maximum) of a function inside a specified interval [25]. Which in our case the function is the deviation function that takes in the scale factor and outputs the deviation after applying the scale. The algorithm converges to one extremum by narrowing down an interval of possible values. Without going into too much detail on the algorithm and its implementation, the algorithm is initialized to search within a range of [0, 10] as possible scale factors and stops when the improvements in deviation fall below 0.01 mm. Applying the Golden-section search method on our running example returns a scale factor of around 2 and results in a nice alignment

[13]During the measurement we found that the depth axis, the X axis, is so inaccurate that it is not possible to use the described method to align the signals. The golden section search would converge to a scaling factor of 0, which minimizes the deviation. This is not a useful result. This is why the depth axis always has a fixed scale of 0.5 applied to it. It is a value that was found to produce the best alignment in the measurements.

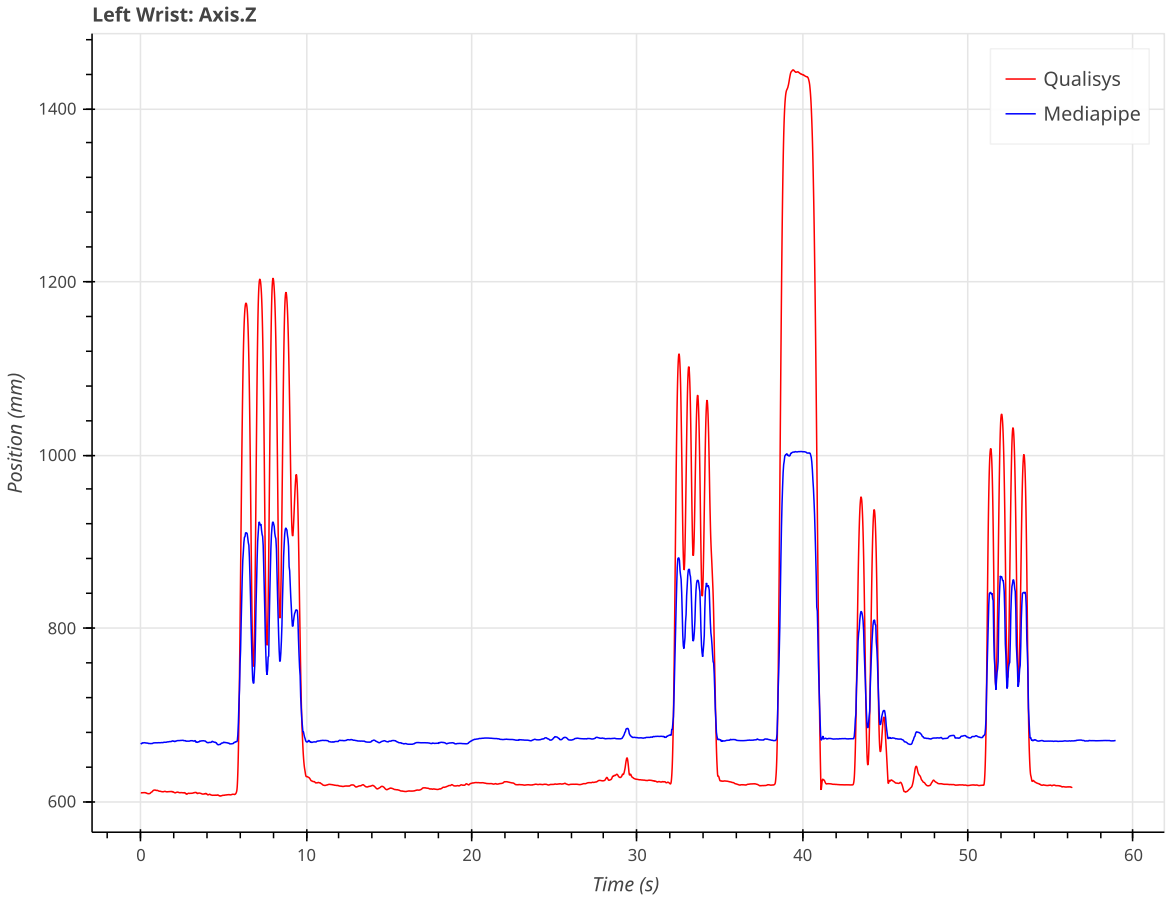


Figure 15: Plot of the MediaPipe (Blue) and Qualisys (Red) left wrist z-axis with the time offset removed.

between both signals (Figure 16). The factor of 2 also makes sense. In the `Landmark` mode, the range of values lie between 0 and 1, reaching these outer values at the edges of the frame. As mentioned, the `Landmark` signal is interpreted to be in meter. As a consequence, the scaling factor is not only a scaling factor, it has become a measurement of the dimensions of what is visible in the frame. This means that the visible height in the video frame is 2 meters, at the location of the test subject, of course.

4.2 Results

In this section all the measurement results are listed and discussed. The results go over the effect of the different MediaPipe Pose models (`LITE`, `FULL`, and `HEAVY`) and the accuracy and frame rate that was achieved. All measurements are performed with `GL version: 3.2 (OpenGL ES 3.2 NVIDIA 550.78)` on a `NVIDIA GeForce RTX 2060` GPU and an `Intel(R) Core(TM) i7-9750H (12) @ 2,60 GHz` CPU, unless specified otherwise. The actual inference is executed on the GPU while the rest of the application just runs on the CPU with minimal overhead. It is the inference that is the most time-consuming element.

The results discuss the accuracy and signal stability of the MediaPipe result, as well as some aspects that affect the signal. The accuracy is the deviation from the MediaPipe

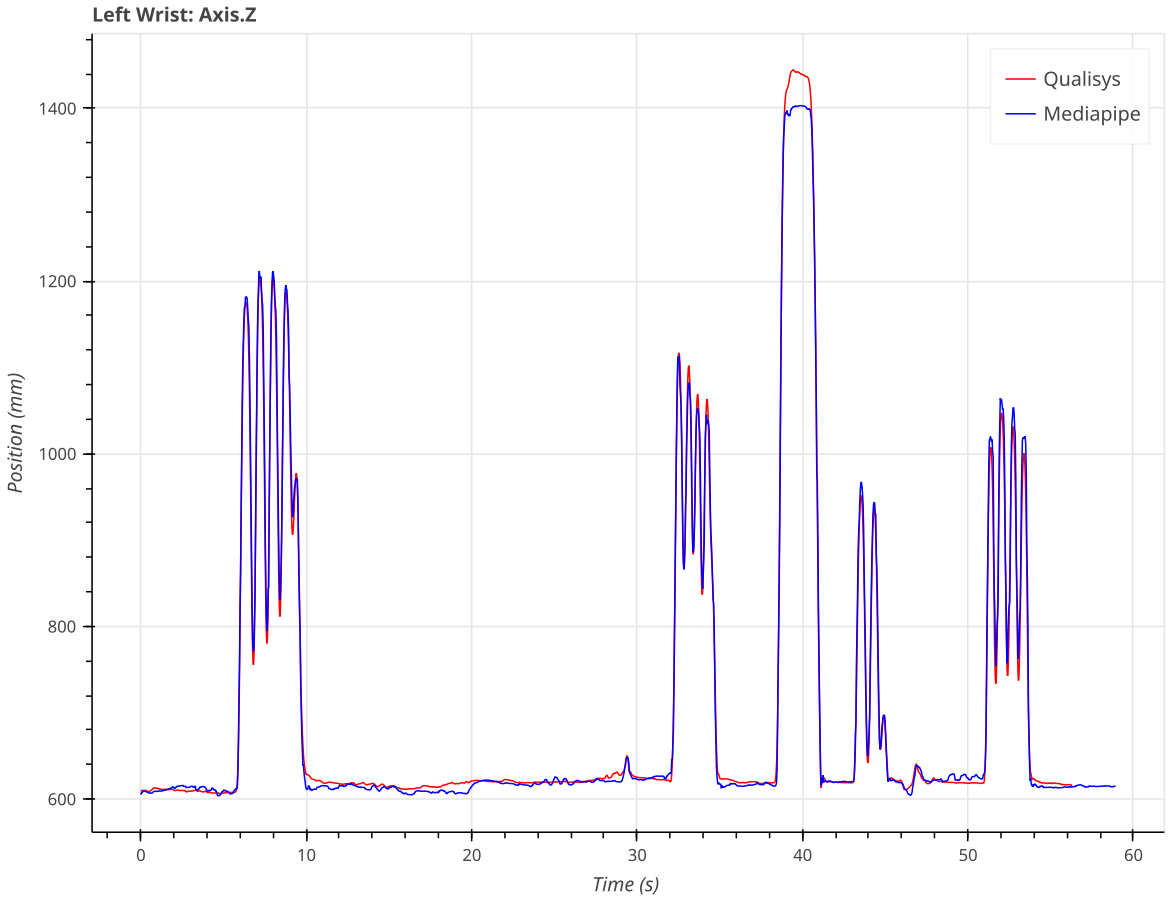


Figure 16: Plot of the MediaPipe (Blue) and Qualisys (Red) left wrist z-axis with the MediaPipe signal scaled to match.

recordings using the techniques from the previous chapter. The signal stability is the absolute value of the derivates of these deviations. The derivative is numerically computed by taking the difference of two succeeding deviation values. By taking the absolute value, we get a measure of how much the deviation differs per frame and thus how stable the signal is. The lower this value, the better the signal stability. All of these values are taken per frame and per tracked marker. They are then described in box plots with a corresponding table. Indicating the mean, standard deviation, min, max, and some percentiles. Since there is too much data to be included in this thesis text, all measurement results are made publicly available on a GitHub repository.[14]

4.2.1 Results Example

To further clarify the results in the coming sections, consider the following example. It is a measurement of an air drumming recording which has been tracked by both Qual-

[14]Follow this link to the GitHub repository: *Mouwrice/DrumPyAnalysis* All measurement results are located in the `measurements` folder. Every measurement is in a separate folder, with the folder name being a very brief description of the measurement. Inside every measurement folder are the actual results, grouped by the MediaPipe model that was used.

isys and MediaPipe. The MediaPipe `FULL` model has been used, together with the regular `Landmark` marker type.[15]

The trajectory of the left wrist marker can be plotted, e.g. the trajectory along the z-axis (vertical axis) in Figure 17.

The deviations are plotted in a box plot (Figure 18), as well as the stability of the signal (Figure 19).

Lastly, the deviation and stability values are also made available by describing them in a tabular format as shown in Table 1 and Table 2.

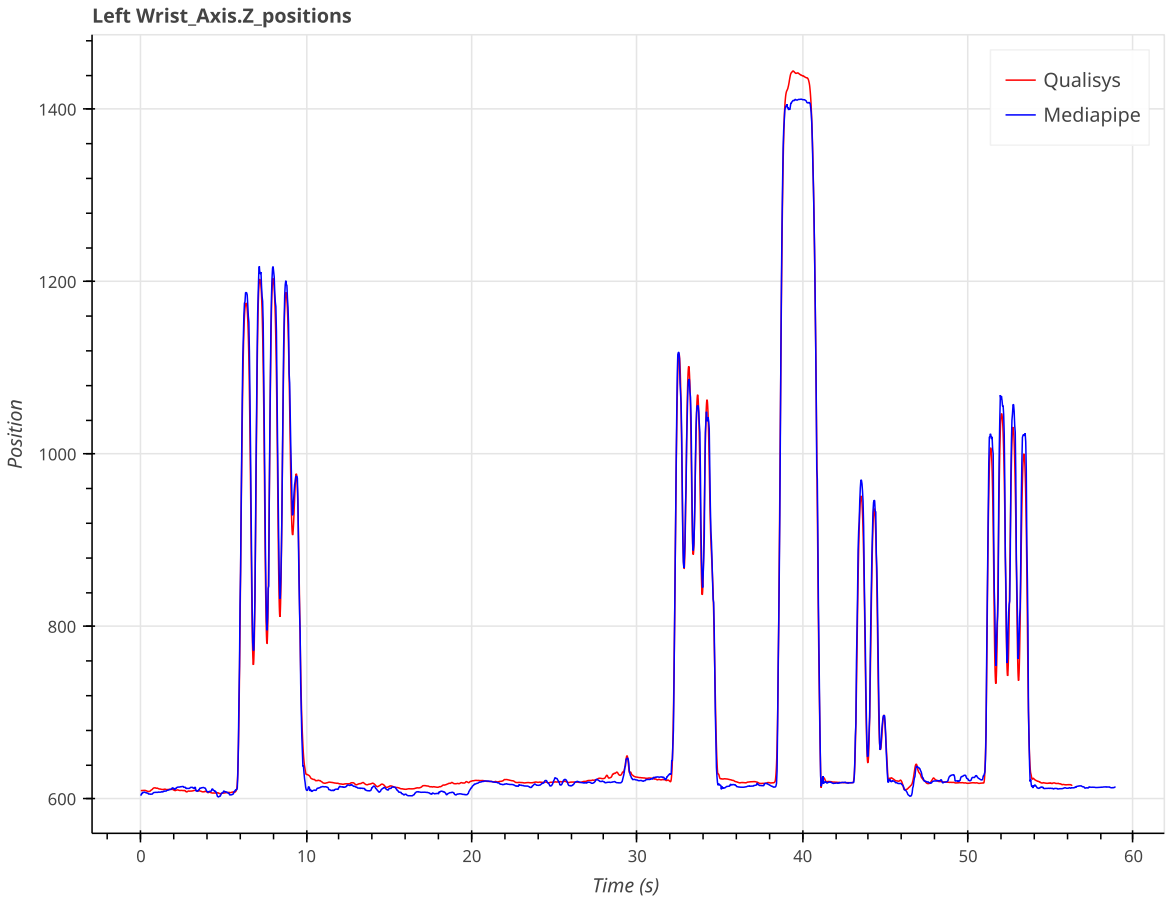


Figure 17: The trajectory along the z-axis (vertical axis) of the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

[15]The complete results of the example here are from the following measurement: *https://github.com/Mouwrice/DrumPyAnalysis/tree/main/data/measurements/maurice_drum_regular/FULL*

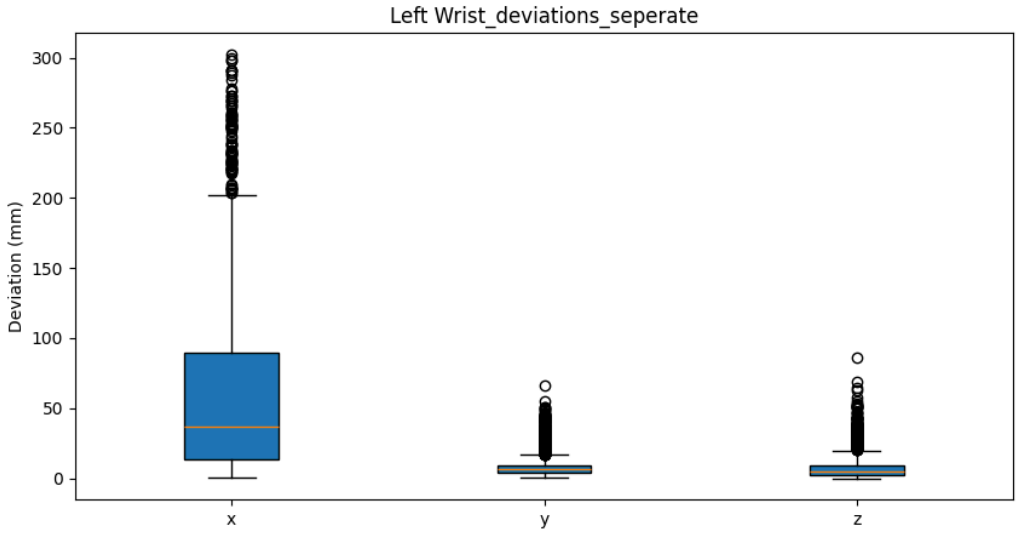


Figure 18: Per axis deviation of the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

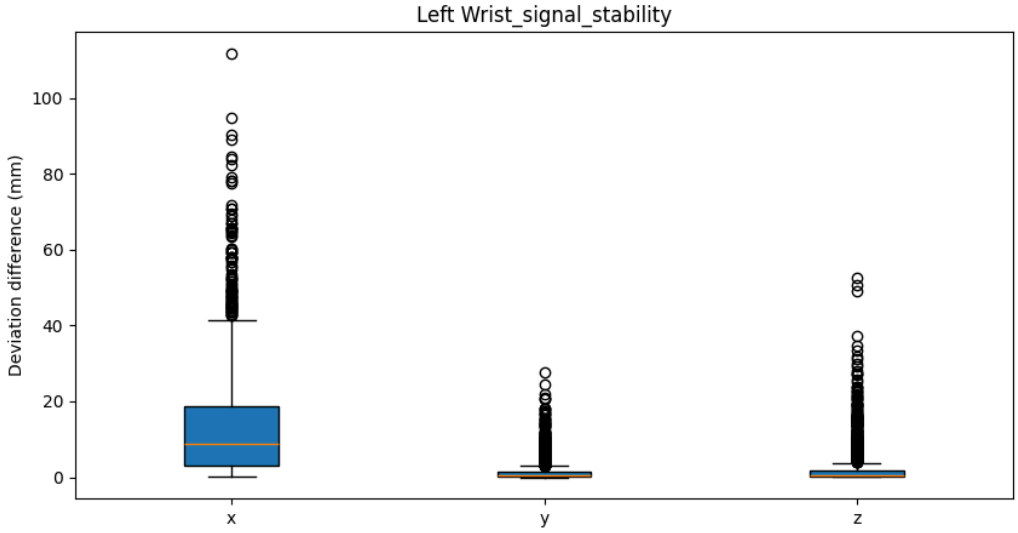


Figure 19: Signal stability of the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

Deviation (mm)	X	Y	Z
mean	50.453455	8.921537	16.330614
std	59.822373	9.993484	15.278144
min	0.048005	0.002286	0.001623
25%	16.249777	2.189822	7.440695
50%	34.482323	5.407212	11.327267
75%	62.916892	11.654837	18.541468

Deviation (mm)	X	Y	Z
max	376.630114	63.694605	87.138306

Table 1: The deviation of the signal from the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

Stability (mm)	X	Y	Z
mean	13.498988	1.621383	2.256355
std	14.644994	2.919033	4.950195
min	0.000851	0.000011	0.000156
25%	3.090311	0.238420	0.163133
50%	8.675376	0.552805	0.459906
75%	18.577672	1.355008	1.632371
max	111.856248	27.770244	52.482742

Table 2: The signal stability from the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

4.2.2 Effect of the models

MediaPipe has three different models available, each with a different size. The larger the model, the more accurate it results should be since the inference model has a larger network that does the inference. Using a larger model negatively impacts the real-time performance, of course. Comparing these different models, we found that the average accuracy does not drastically increase given a large model. The achievable frame rate, however, can drop significantly given these large models, dependent on the hardware used.

For each model, 3 results are provided, each result being from one specific recording. The recordings are all drumming motions but with increasing levels of intensity. The first recording is of small movements (`maurice_drum_small`). The second one features normal movements (`maurice_drum_regular`) and the last one contains fast and big movements (`maurice_drum_fast`). The accuracy is taken from a total of 10 tracked markers, being the `Left Elbow`, `Right Elbow`, `Left Wrist`, `Right Wrist`, `Left Hip`, `Right Hip`, `Left Heel`, `Right Heel`, `Left Foot Index`, `Right Foot Index`.

4.2.2.1 LITE

The `LITE` model is 3 MB in size and is the smallest model available.

The total deviation of all three recordings are displayed in Table 3, Table 4, and Table 5, respectively. We can immediately draw some conclusions from these results. The Y and Z axis (horizontal and vertical) have a relatively good accuracy compared to the x-axis (depth). The depth is actually very imprecise and unstable, having a high mean deviation

and a high standard deviation makes this depth not very usable. It also indicates that the wider the range of movements and the speed at which they are performed results in a slight drop in accuracy. Almost all deviation values are slightly higher than their corresponding values from a recording featuring less and slower movement.

Despite the x-axis being very imprecise, the Y and Z axis consistently achieve an accuracy of minimum 1 centimetre. For an air drumming application, this level of accuracy is already pretty usable.

Deviation (mm)	X	Y	Z
mean	46.216412	6.504117	8.755020
std	52.435068	8.534992	9.435842
min	0.000424	0.001266	0.000092
25%	7.747909	1.523800	2.533034
50%	26.830197	3.518769	5.717354
75%	65.330929	7.591232	10.964000
max	304.913545	91.105333	69.909789

Table 3: The total deviation from the `maurice_drum_small` measurement. Model: `LITE`, Marker type: `Landmark`.

Deviation (mm)	X	Y	Z
mean	44.065230	6.498654	10.911481
std	55.829339	7.804492	12.150179
min	0.000830	0.000079	0.000673
25%	8.900595	2.055750	3.500705
50%	24.601205	4.378533	7.580231
75%	57.471564	8.009390	13.549875
max	460.941039	164.103709	140.990013

Table 4: The total deviation from the `maurice_drum_regular` measurement. Model: `LITE`, Marker type: `Landmark`.

Deviation (mm)	X	Y	Z
mean	64.872857	7.181918	10.046277
std	70.935324	9.058099	14.284985
min	0.006952	0.000276	0.000796
25%	9.690526	2.168622	2.386345
50%	31.462373	4.908393	5.571245
75%	113.099430	8.903545	12.760656
max	680.599720	226.170262	592.765323

Table 5: The total deviation from the `maurice_drum_fast` measurement. Model: `LITE`, Marker type: `Landmark`.

4.2.2.2 FULL

The `FULL` models is double the size of the `LITE` model, at 6 MB. But that does not mean the deviation is halved. Compared to the `LITE` model, the average accuracy is increased by 1 and at most 2 mm. The standard deviation is reduced a bit using this heavier model. Some outlier values are also reduced. This indicates that the `FULL` model provides a cleaner result but only with a marginal increase in accuracy.

Deviation (mm)	X	Y	Z
mean	40.883051	5.702290	7.874736
std	50.636690	6.940303	8.912165
min	0.000027	0.000366	0.000008
25%	5.576486	1.458272	2.017663
50%	19.581533	3.285286	4.814794
75%	56.780453	7.264643	9.913902
max	309.870406	75.601338	82.893488

Table 6: The total deviation from the `maurice_drum_small` 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

Table 7: The total deviation from the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`.

Deviation (mm)	X	Y	Z
mean	57.356438	6.444404	9.623760
std	71.275673	8.949972	15.462130
min	0.002776	0.001176	0.000118
25%	5.237184	1.277760	1.798388
50%	17.238973	3.093675	4.298769
75%	102.966444	7.503870	11.247950

Deviation (mm)	X	Y	Z
max	679.421116	117.911612	586.803848

Table 8: The total deviation from the `maurice_drum_fast` measurement. Model: `FULL`, Marker type: `Landmark`.

4.2.2.3 HEAVY

The `HEAVY` model is the largest of them all at a size of 26 MB. At this level, we have clearly reached a point of diminishing returns. Despite increasing the model size by a factor of 4, the improvements are not a big jump up. The jump in accuracy from `FULL` to `HEAVY` is similar to the jump from `LIGHT` to `FULL`, again increasing the accuracy by 1 mm. The x-axis also gets an increase in accuracy but proves to still be too unstable and imprecise to be of any use. At this point, the axis lateral to the image frames (Y and Z) are considerably accurate, reaching an average accuracy of 5 and 7 mm respectively. Just as with the smaller models, larger movements lead to a slight drop in accuracy, there is especially an increase in outliers.

Deviation (mm)	X	Y	Z
mean	38.662930	4.880106	6.931904
std	46.034748	6.095308	7.833226
min	0.000757	0.000524	0.000945
25%	5.908032	1.228680	1.864810
50%	17.719570	2.879538	3.958334
75%	61.890476	5.876115	8.929086
max	293.278487	90.609627	65.955275

Table 9: The total deviation from the `maurice_drum_small` measurement. Model: `HEAVY`, Marker type: `Landmark`.

Deviation (mm)	X	Y	Z
mean	48.331793	5.311494	9.191098
std	66.957378	6.810102	10.310719
min	0.006057	0.000938	0.000114
25%	9.469759	1.529734	2.387941
50%	25.329217	3.363876	5.430044
75%	61.941299	6.533454	12.837094
max	622.943655	67.798017	90.548038

Table 10: The total deviation from the `maurice_drum_regular` measurement. Model: `HEAVY`, Marker type: `Landmark`.

Deviation (mm)	X	Y	Z
mean	61.698936	5.610732	7.894003
std	69.377889	7.535471	12.391467
min	0.007052	0.000091	0.000255
25%	7.327052	1.214547	1.887956
50%	25.502218	2.906578	4.056903
75%	112.404883	7.307336	9.338038
max	678.986135	121.335236	587.988858

Table 11: The total deviation from the `maurice_drum_fast` measurement. Model: `HEAVY`, Marker type: `Landmark`.

4.2.2.4 Signal stability

One metric that has been left out, so far, is the signal stability. The deviation tables above hint that a heavier model might not just provide a small increase in accuracy, but also provide an increase in signal stability. This is hinted at by the lower standard deviation and the lower 75% percentile values. Having a more stable signal, meaning less outliers and a more consistent deviation difference between frames, is also an important aspect. The following tables plot the signal stability for every model on the `maurice_drum_regular` measurement. Remember that the signal stability is calculated as the absolute difference in deviation between consecutive frames, the lower the values the better.

Stability (mm)	X	Y	Z
mean	7.764771	1.352941	2.096501
std	10.085330	2.809363	4.220660
min	0.000014	0.000023	0.000073
25%	1.026297	0.208817	0.217875
50%	3.969899	0.595858	0.739438
75%	10.707028	1.440624	2.116421
max	118.670178	116.634008	107.199950

Table 12: The signal stability from the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

Stability (mm)	X	Y	Z
mean	7.579568	1.106332	1.694458
std	10.574589	2.057231	3.608910
min	0.000109	0.000078	0.000012
25%	0.752476	0.165129	0.176408
50%	3.633556	0.468424	0.560157
75%	10.260663	1.137847	1.569148

Stability (mm)	X	Y	Z
max	112.231639	46.151331	52.179156

Table 13: The signal stability from the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

Stability (mm)	X	Y	Z
mean	6.122471	0.979171	1.511921
std	8.788107	1.873792	3.426255
min	0.000077	0.000090	0.000053
25%	0.736543	0.133611	0.139911
50%	2.849786	0.392975	0.450573
75%	8.104202	0.981124	1.274915
max	108.713253	40.173077	51.322185

Table 14: The signal stability from the `maurice_drum_regular` measurement. Model: `FULL`, Marker type: `Landmark`, Marker: `Left Wrist`.

Table 12, Table 13 and Table 14 clearly demonstrate that a larger model not only increases accuracy but also the signal stability. The effect is modest, with the most significant difference being the values going from the `LITE` model to the `FULL` model. This effect is also noticeable when viewing the inference visualization, where we can see that the `FULL` model provides a more stable result. The markers are less jittery and their movements are 'smoother'.

4.2.2.5 Conclusion

Now that we have compared the accuracy of all three models, we have a clear view of the expected accuracy. For any lateral movement, that is movement lateral to the image frame, the horizontal and vertical axis, an accuracy of 5-10 mm can be achieved with some deviations from that accuracy of at most 1 centimetre. There is however also the possibility for some jitter to occur in the resulting signal which can some major deviations from the actual movement, but these are only of short duration.

The depth axis, the x-axis, is considerably less accurate. The accuracy is around 40-60 mm with some deviations of up to 100 mm. The depth is as mentioned *"obtained via the GHUM model fitted to 2D point porjections."* Unfortunately, this depth is not very usable for an air drumming application. The depth is also very unstable, with a high standard deviation and a high number of outliers. This is especially the case when the movements are fast and big.

Comparing the models, there is little accuracy to be gained from choosing a larger model. However, larger models provide a more stable signal, which can be essential as the drum-

ming application mostly looks at the relative movements instead of absolute values. As we are developing an application to be used live, the largest model that can achieve real-time inference is preferred. The inference time is of course dependent on the hardware, which means that in some cases the HEAVY model can be used but in other cases the LITE model is the only one that can be properly run in real-time, Section 4.2.3.

4.2.3 Achievable framerate

The frame rate that can be achieved should be high enough for a proper real-time application. The higher the frame rate, the more responsive the application will feel and the more accurate the tracking can be. A higher frame rate also allows for faster motions to be captured, which is handy in the case of fast drumming motions. The frame rate is dependent on the hardware used, but also on the model that is used. The larger the model, the more computationally expensive it is to run the inference. The following table (Table 15) lists the maximum frame rate that can be achieved for each model, as well as the device that was used to run the inference.

Device (fps)	LITE	FULL	HEAVY
CPU Intel Core i7-10750H	25	20	7
GPU Intel UHD Graphics 630	19	17	11
CPU AMD Ryzen 5 5600X	40	30	10
GPU NVIDIA RTX 2060	45	42	40

Table 15: The maximum framerate (fps) that can be achieved for each model with different devices.

The table shows that the LITE model can be run in real-time on all devices. The FULL model can also be run in real-time on most devices, but the HEAVY model is too computationally expensive to run in real-time on most devices. The HEAVY model can only be run in real-time on the NVIDIA RTX 2060 GPU. It is no surprise that the GPU is the most performant of the four devices. However, despite being able to run the HEAVY model at a constant 40 fps, the GPU's performance on the smaller models is surprisingly not a lot higher. We can conclude that the GPU has a lot of power but lacks the speed of a CPU[16], hence the good results for the HEAVY model and the not much improved results for the smaller models. This difference between power and speed also becomes apparent when comparing the Intel Core i7-10750H and the AMD Ryzen 5 5600X. The AMD Ryzen 5 5600X has a higher clock speed and can run the FULL model at 30 fps, while the Intel Core i7-10750H can only run the FULL model at 20 fps. The HEAVY model is too computationally expensive for both devices. On the LITE model, however, they are on par with the GPU's. The Intel Core i7-10750H can run the LITE model at 25 fps, while the AMD Ryzen 5 5600X can run the

[16]A CPU typically has a higher clock speed than a GPU, allowing to perform many more consecutive operations per second. A GPU is mainly designed to perform many operations simultaneously.

LITE model at 40 fps. The Intel UHD Graphics 630 and the NVIDIA GeForce 2060 GPU can run the LITE model at 19 and 45 fps respectively.

The conclusion is that the FULL model is the best model to use for real-time applications. It provides a good balance between accuracy and frame rate. The HEAVY model is too computationally expensive to run in real-time on most devices, while the LITE model is somewhat unstable to rely on. The FULL model can be run in real-time on most (modern) devices, providing a good balance between accuracy and frame rate. If the device is not powerful enough to run the FULL model in real-time, the LITE model can be used as a fallback.

4.2.4 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 Figure 20, the jitter is clearly visible around the 20-second mark. 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 partly explains the increased signal stability using these models. 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.

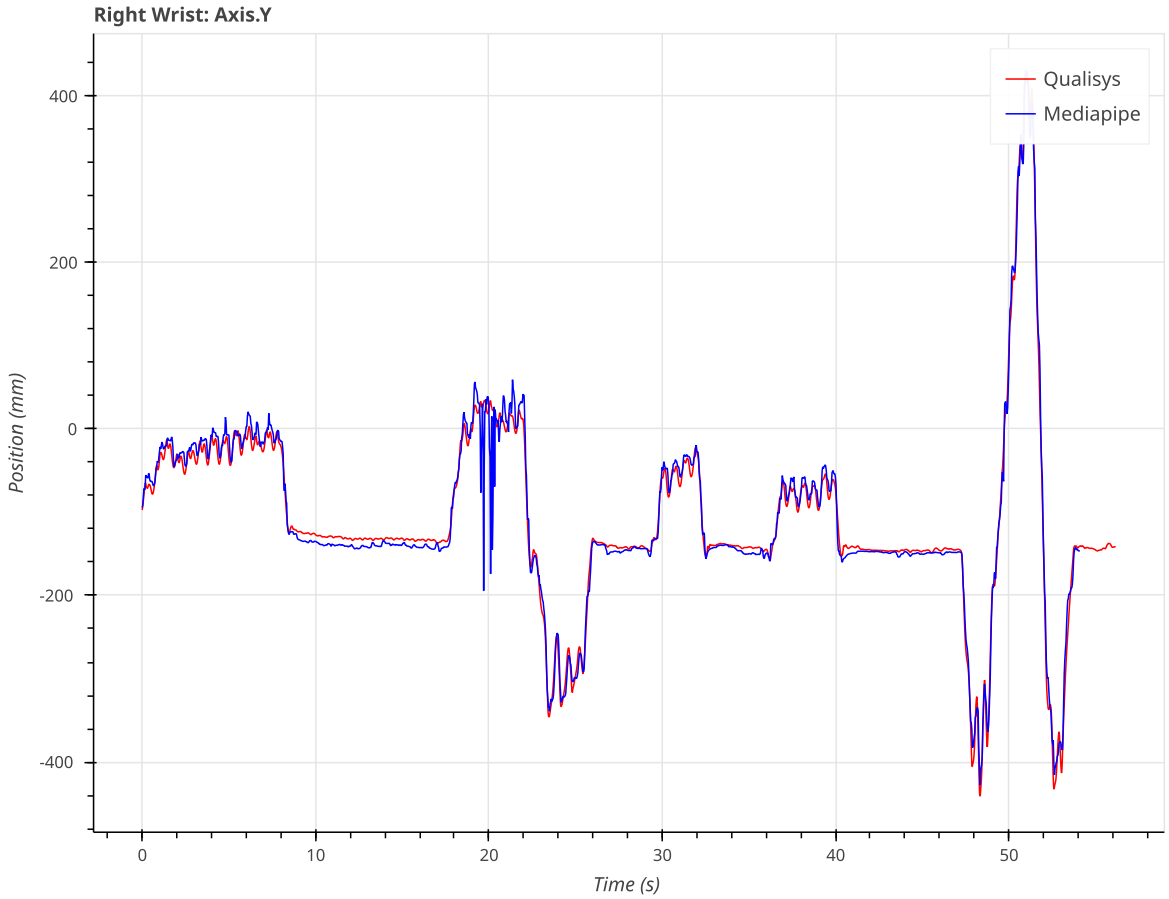


Figure 20: A case of jitter in the `maurice_drum_fast` measurement around the 20 seconds mark. Model: `LITE`, Marker type: `Landmark`, Marker: `Right Wrist`.

4.2.5 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 Figure 21. 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. The signal stability tables also clearly show that the noise is present in the `LITE` model but is reduced in the `FULL` and `HEAVY` models. (Table 16)

4.2.6 Resolution

Following the description of the model network of the MediaPipe Pose Task, we know that the input has a fixed size of 256x256x3. This means that the resolution of the input image is 256x256 pixels. As a result of this fixed input size, we should see no significant difference in accuracy when using different resolutions. There might be a slight difference in the accuracy due to the image being resized to fit the input size, but this difference should be negligible. This hypothesis is tested by comparing the deviation of

LITE Stability	X	Y	Z
mean	8.537066	1.053139	2.158305
std	8.288110	1.261807	2.967522

FULL Stability	X	Y	Z
mean	6.025950	0.589802	1.300819
std	8.030914	0.914598	2.288527

HEAVY Stability	X	Y	Z
mean	5.830511	0.441127	1.026949
std	6.083284	0.685263	1.925742

Table 16: The signal stability from a noisy signal in the the `maurice_drum_regular` measurement. Models: `LITE`, `FULL`, `HEAVY`. Marker type: `Landmark`. Marker: `Right Heel`.

the `maurice_drum_regular` measurement at different resolutions. The resolutions used are 1080p, 720p, and 480p.[17] The deviation values are shown in Table 17. The deviation values

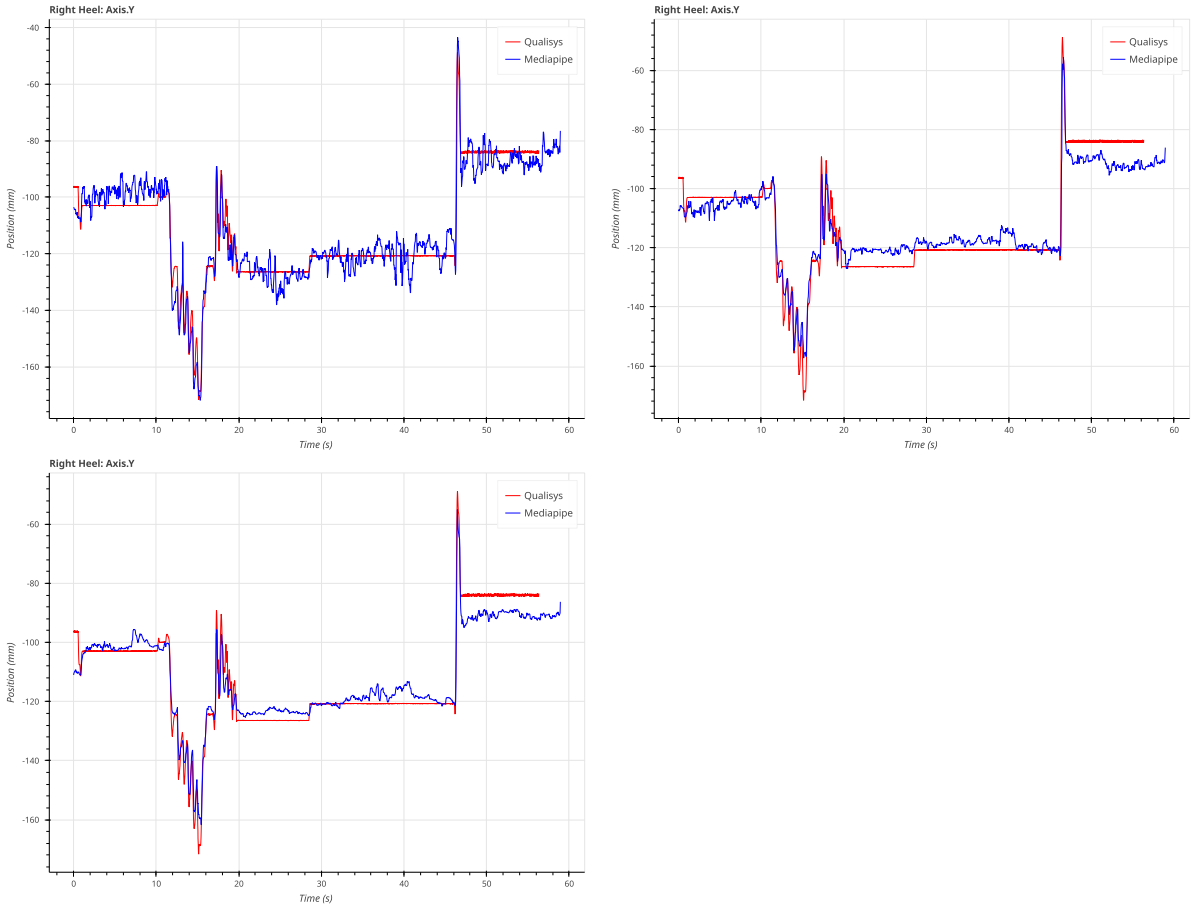


Figure 21: A noisy signal in the `maurice_drum_regular` measurement. Models: `LITE` (top left), `FULL` (top right), `HEAVY` (bottom left). Marker type: `Landmark`. Marker: `Right Heel`.

[17]The original measurement video was also captured in 1080p but was encoded again with the same encoding settings used to achieve the smaller resolution videos, except for the resolution, of course. This was done to make sure that the video quality and encoding are the same. It is also the reason for slightly

1080p Deviation (mm)	X	Y	Z
mean	40.076227	10.396079	12.230394
std	49.000929	10.724212	14.250999

720p Deviation (mm)	X	Y	Z
mean	39.937372	10.387030	12.473164
std	48.821626	10.771447	14.450003

480p Deviation (mm)	X	Y	Z
mean	41.670763	11.361194	11.975922
std	48.763101	11.131461	14.568390

Table 17: The accuracy of different resolutions of the `maurice_drum_regular` measurement. Model: `FULL`. Marker type: `Landmark`.

are very similar. Thus follows that the resolution of the input image does not significantly affect the accuracy of the model. This is good news, as it means that the model can be used on different devices with different resolutions without a significant loss in accuracy.

4.2.7 World Landmarks

All previous results are from measurements with the `Landmark` as marker type. These are points that have coordinates in the image frame with an added depth value. MediaPipe also provides `WorldLandmarks` as a marker type. These are real-world 3D coordinates. The values are in meters and are relative to the midpoint between the hips. MediaPipe tries to predict the size of the person in the frame and uses this to scale the world landmarks. `WorldLandmarks` allows to decouple the marker locations from the image frame. With them, one can track the movements relative to the person instead of the image frame. This can be useful when the person is moving around in the frame or when the person is moving towards or away from the camera. As this adds another layer of uncertainty (the scale of the person in the frame is but a prediction), the accuracy of the `WorldLandmarks` is expected to be lower than the `Landmarks`.

The deviation values for the `WorldLandmarks` are shown in Table 18. The deviation values are indeed higher than the deviation values for the `Landmarks` by a value of 2 to 5 mm.[18] The same can be observed for the signal stability in Table 19. For the most accuracy tracking the regular `Landmarks` should be used. However, if the application requires tracking movements that are relative to the person instead of the image frame, then the `WorldLandmarks`

different results for the 1080p resolution in previous tables, which the very attentive reader might have noticed.

[18] Note that the depth values are somewhat improved with the `WorldLandmarks` marker type. This is attributed to a better scaling factor than the somewhat arbitrary scaling factor of 0.5 that was chosen in the x-axis alignment. The depth values are still as inaccurate as before, but the entire scaling of the depth axis is just a bit better. These `WorldLandmark` depth scale would still converge to zero if the Golden-section search is applied.

need to be used. One might also opt for a combination of both, as MediaPipe always outputs both types of landmarks.

`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

`WorldLandmark` Deviation (mm)	X	Y	Z
mean	40.064675	10.425273	12.136874
std	49.167113	10.624388	13.952320
min	0.003811	0.000245	0.001166
25%	7.237019	3.242694	2.720210
50%	22.192832	7.156534	7.904396
75%	57.645794	13.916220	16.739466
max	364.849119	81.609742	147.278370

Table 18: The accuracy of `WorldLandmarks` compared to the accuracy of `Landmarks`. Model: `FULL`. Measurement: `maurice_drum_regular`.

`Landmark` Stability (mm)	X	Y	Z
mean	7.579568	1.106332	1.694458
std	10.574589	2.057231	3.608910
min	0.000109	0.000078	0.000012
25%	0.752476	0.165129	0.176408
50%	3.633556	0.468424	0.560157
75%	10.260663	1.137847	1.569148
max	112.231639	46.151331	52.179156

`WorldLandmark` Stability (mm)	X	Y	Z
mean	3.967809	1.851960	2.493500
std	6.406780	2.779646	4.056749
min	0.000046	0.000069	0.000042
25%	0.377586	0.276647	0.221359
50%	1.447640	0.808166	0.881040
75%	4.694866	2.216550	3.038803

WorldLandmark Stability (mm)	X	Y	Z
max	87.592255	29.960801	48.716450

Table 19: The stability of `WorldLandmarks` *compared to the stability of* `Landmarks`*. Model:* `FULL`*. Measurement:* `maurice_drum_regular`*.*

Apart from the decrease in accuracy, there is also the scale that needs to be considered. The scale of the `WorldLandmarks` is not the same as the scale of the `Landmarks`. The `Landmarks` are in pixels, while the `WorldLandmarks` are in meters. This means that the scale of the `WorldLandmarks` is dependent on the size of the person in the frame. This scale is not always accurate, as it is a prediction. That the scale is not quite accurate is easily proved by applying our alignment method. The Golden-section search returns a scale factor for these `WorldLandmarks` so they would properly align. Two measurements have been compared, with the same camera setup but with two people of different size. The recording of the person with a height of 170cm resulted in the scaling factor of 0.9. This indicates that the person is smaller than the predicted size by MediaPipe. The other recording features a person of 184cm, which resulted in a scaling factor of 1.1. This indicates that the person is larger than the predicted size by MediaPipe. This is a clear indication that the scale of the `WorldLandmarks` is not always accurate and some alignment is needed to get the proper scale.

4.2.8 Depth Issues

In all the measurements it is pretty clear that something is wrong with the depth axis. It has a way higher deviation than the other axes and is also more unstable. It does of course make sense that the accuracy and stability differ from the other axes as the depth is inferred via the seperate GHUM model. But how bad is the depth axis actually?

A trajectory of the depth from one recording is displayed in Figure 22. It is obvious that the MediaPipe signal (blue) jumps all over the place. However, with some imagination, it can be seen that when there are actually peaks and valleys in the signal, the MediaPipe signal follows the general trend. This is a good sign, as it means that the signal is not completely random. The scale of them is not quite right and inconsistent though. Unfortunately we see that peaks and valleys are also present in the MediaPipe signal when none are present in the actual signal with a scale similar of that of the actual peaks and valleys. This makes it really hard to use the depth signal for any application. If the depth signal was just a bit inaccurate it could still be used to get a general idea of the depth and movement in an application, but with the current signal there is just no way to tell if something is an actual peak or valley or just a random spike in the signal.

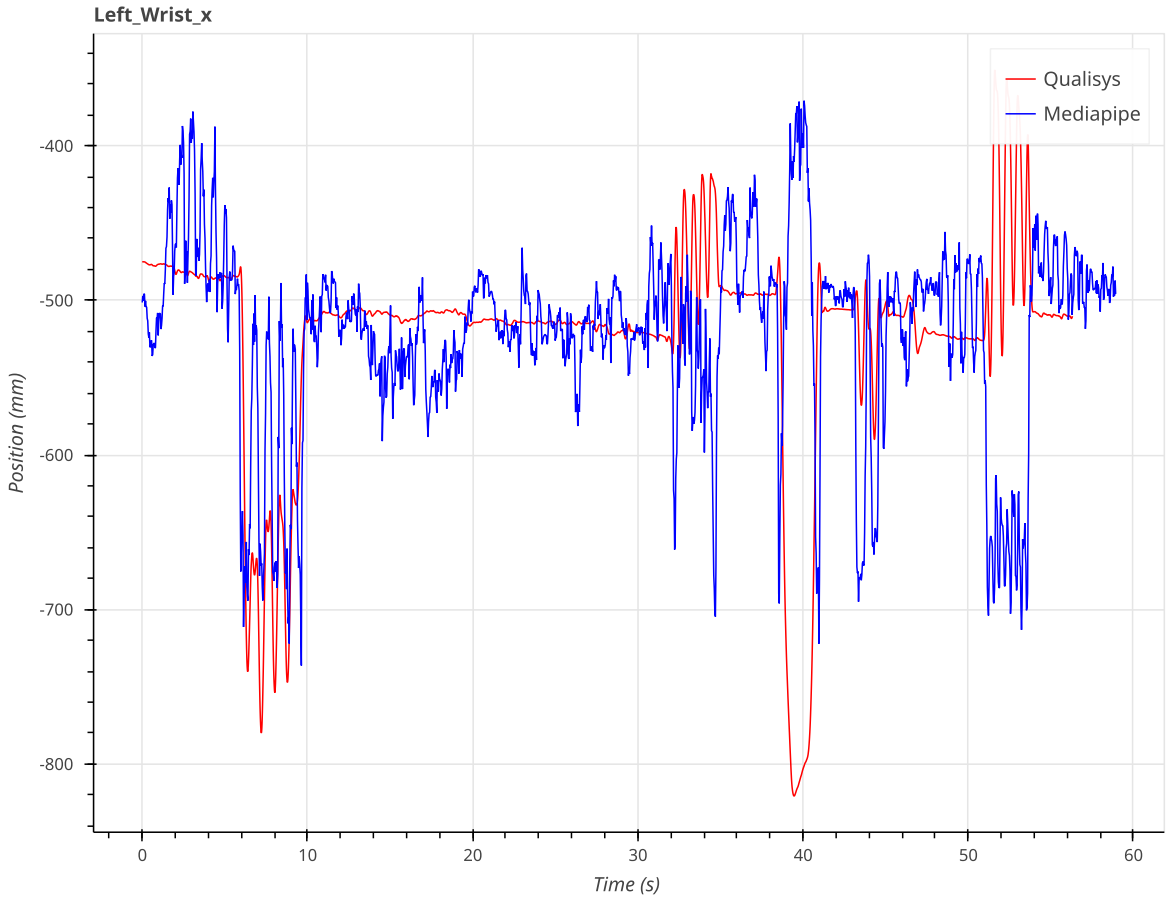


Figure 22: A sample of the inferred depth trajectory. Model: `LITE`*, Marker Type:* `Landmark`*, Marker:* `Left Wrist`*.*

5 Reducing Jitter and Noise

The measurements show that the pose estimation suffers from jitter and noise. It is, of course, preferred to have as little of this jitter and noise as possible. This section describes a post-processing method to reduce the jitter and noise from the pose estimation. While not entirely eliminating the jitter and noise, the method is shown to reduce it.

5.1 Method

The method presented is based on the prediction of points in the image. The idea is to predict the position of the points in the image based on the previous positions. This prediction is then used to correct the pose estimation. Every incoming point is compared to the predicted position, based on distance. The distance is used to determine if the newly detected point is either caused by jitter or noise, or if it is just caused by the movement of the object.

The method principle is best explained with a figure, Figure 23. 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 estimaiton, is then compared to the predicted position E. If the distance 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 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.

5.1.1 Prediction

One aspect of the method is the prediction of the position of the points. The prediction is based on the previous positions of the points. 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 can somewhat be predicted. The higher the framerate the more accurate the prediction can be. Advanced prediction methods are therefore not necessary.

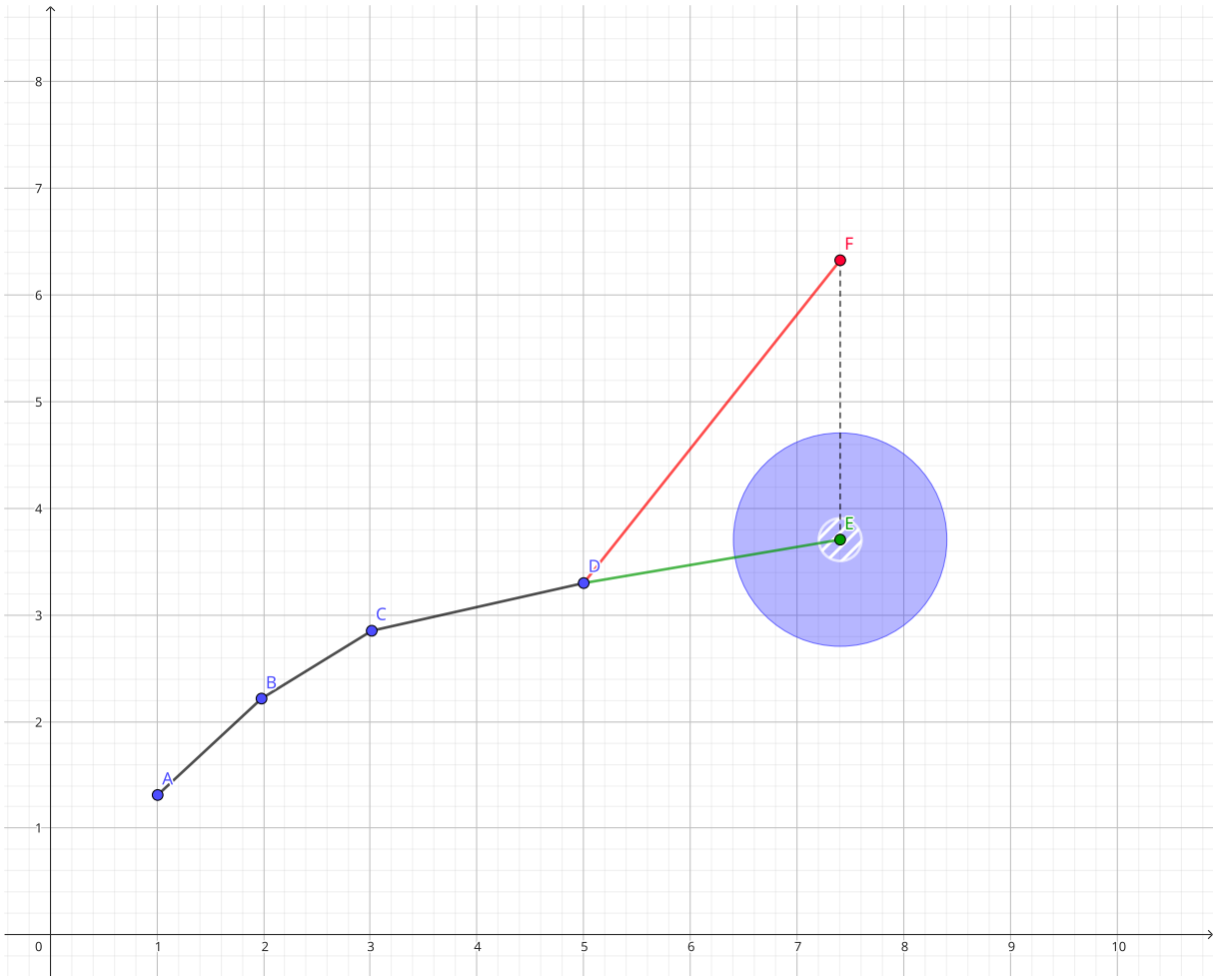


Figure 23: Noise and Jitter reduction method principle. The horizontal axis is the time axis. The vertical axis is the value of the dots. Dots A to D are previous positions, E is the predicted position, F is the current position from MediaPipe.

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 done because in a live stream setting, the time difference between frames can vary because of 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: $P_3' = P_2 + \vec{v}.(t_3 - t_2)$ 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.

The prediction requires keeping a memory of a given amount of previous points. The description above only considers the last 2 points. However, the method can be extended to consider more points. When considering more points, the vector $\vec{v}$ is calculated by av-

eraging the n vectors of all consecutive points stored in memory. This can be done to reduce the effect of noise and jitter in the prediction. However, when too many points are considered, the prediction might not be reactive enough to more sudden changes in the movement of the object. One might also opt for a weighted average, where the most recent points have a higher weight in the average, to give more importance to the most recent points. Using a weighted average has not been tested in this research.

5.1.2 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 distance is calculated using the Euclidean distance. 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 than: $\hat{P} = (1 - b).P' + b.P$.

The chosen interpolation function is: $f(x) = e^{-(\log_e(x))^2}$, Figure 24. 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 important 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. Only when the frame rate is higher, the average and expected distance between two consecutive points is smaller. In that case. the interpolation function can be more stringent and more precise at discarding outliers.

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

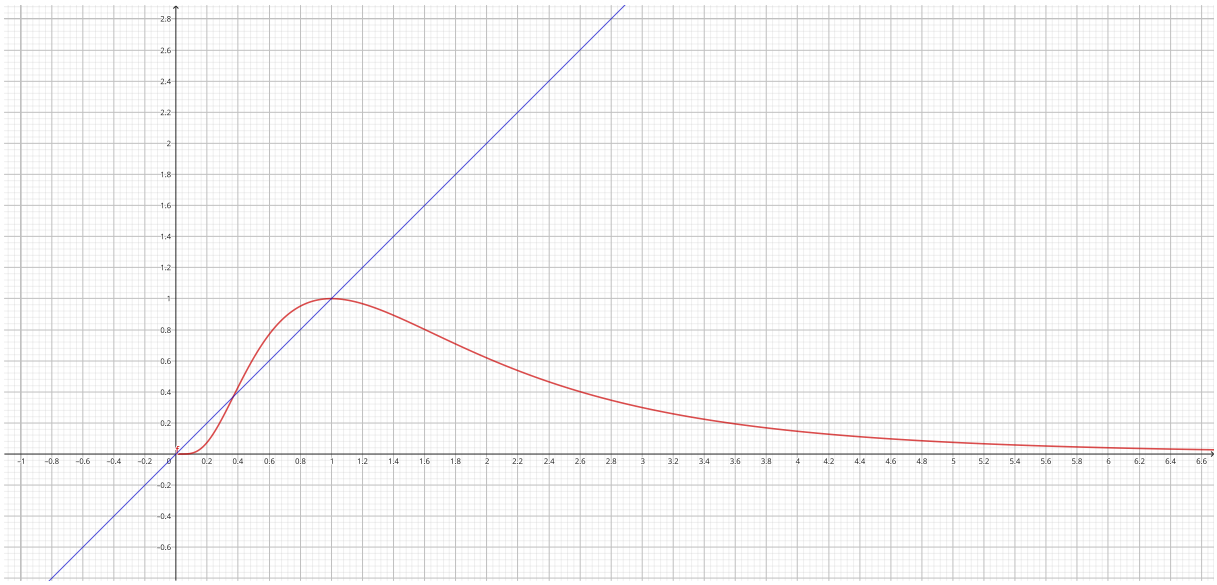


Figure 24: Interpolation function. The horizontal axis is the distance between the predicted position and the actual position. The vertical axis is the interpolation factor. The blue line, $g(x) = x$, is purely a visual aid.

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, the interpolation function becomes: $f(x) = e^{-\left(\log_e\left(\frac{x}{d}\right)\right)^2}$. The effect of this parameter is shown in Figure 25. Note that it not only changes the distance at which the interpolation function reaches its maximum value, but also the steepness of the function. By setting d to a lower value, we lower the acceptable range of movement. This is particularly useful as we are working in the image frame where the positions are already normalized to be between 0 and 1.[19]

A second parameter further controls the "tightness" of the interpolation around the maximum value. Call this parameter s, the interpolation function becomes: $f(x) = e^{-s.\left(\log_e\left(\frac{x}{d}\right)\right)^2}$ and is shown in Figure 26. The parameter s controls the falloff of the function. A higher value of s means that the function falls off more quickly on both sides of the maximum value. This means that the interpolation is stricter.

5.2 Results

The following section compares the results achieved by the method with measurements without any post-processing. Results of recordings at 30 fps and 60 fps are discussed, as well as the impact of the memory size of the method.

[19]Remember the `Landmark` type of marker. The positions are normalized to be between 0 and 1. This means that the distance between two points will be, on average, a lot smaller than 1.

Figure 25: Interpolation function with $d = 0.5$. 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.

5.2.1 30 fps

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 memory size of 2 frames, the peak $d = 0.015$, and the 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

Figure 26: 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..

signal stability. Table 20 shows the stability of the signal with and without processing. For the Y and Z axis, 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. This impact is better displayed in the box plots from Figure 27. The box plot shows the distribution of the signal stability. The processed signal has a smaller range and a smaller median. The outliers are also reduced. The method has a positive impact on the signal stability.

One might argue that the X axis is also improved. However, given the results from the measurement chapter, we know that the X axis is far from accurate. The increase in stability for the X axis is purely attributed to the method reducing the noise and jitter. The method does not improve the actual accuracy of the X axis, even though the results might suggest otherwise.

Stability unprocessed (mm)	**X**	**Y**	**Z**
mean	9.230469	1.814981	2.656927
std	13.762306	4.700073	8.271183
min	0.000365	0.000031	0.000004
25%	1.143315	0.210036	0.251675
50%	4.848148	0.634218	0.849195
75%	13.118014	1.727039	2.558757
max	676.041508	180.404176	592.605731

Stability processed (mm)	**X**	**Y**	**Z**
mean	5.940251	1.419173	2.109245
std	9.745013	3.663793	8.273394
min	0.000060	0.000002	0.000004
25%	0.870641	0.072621	0.070995
50%	3.905946	0.258439	0.251171
75%	9.016494	0.967466	1.169109
max	674.856917	116.912891	593.312640

Table 20: The signal stability from the `maurice_drum_fast` measurement without processing (top) and with processing (bottom). Model: `LITE`, Marker type: `Landmark`

Before we conclude that the method achieves its goal, we should also consider the impact of the method on the accuracy.[20] Table 21 clearly shows that the method has no negative impact on the accuracy. The accuracy is ever so slightly improved, but the improvement is not significant enough to conclude that the method improves the accuracy.

[20] Remember that the accuracy has been defined as the deviation from the Qualisys recordings.

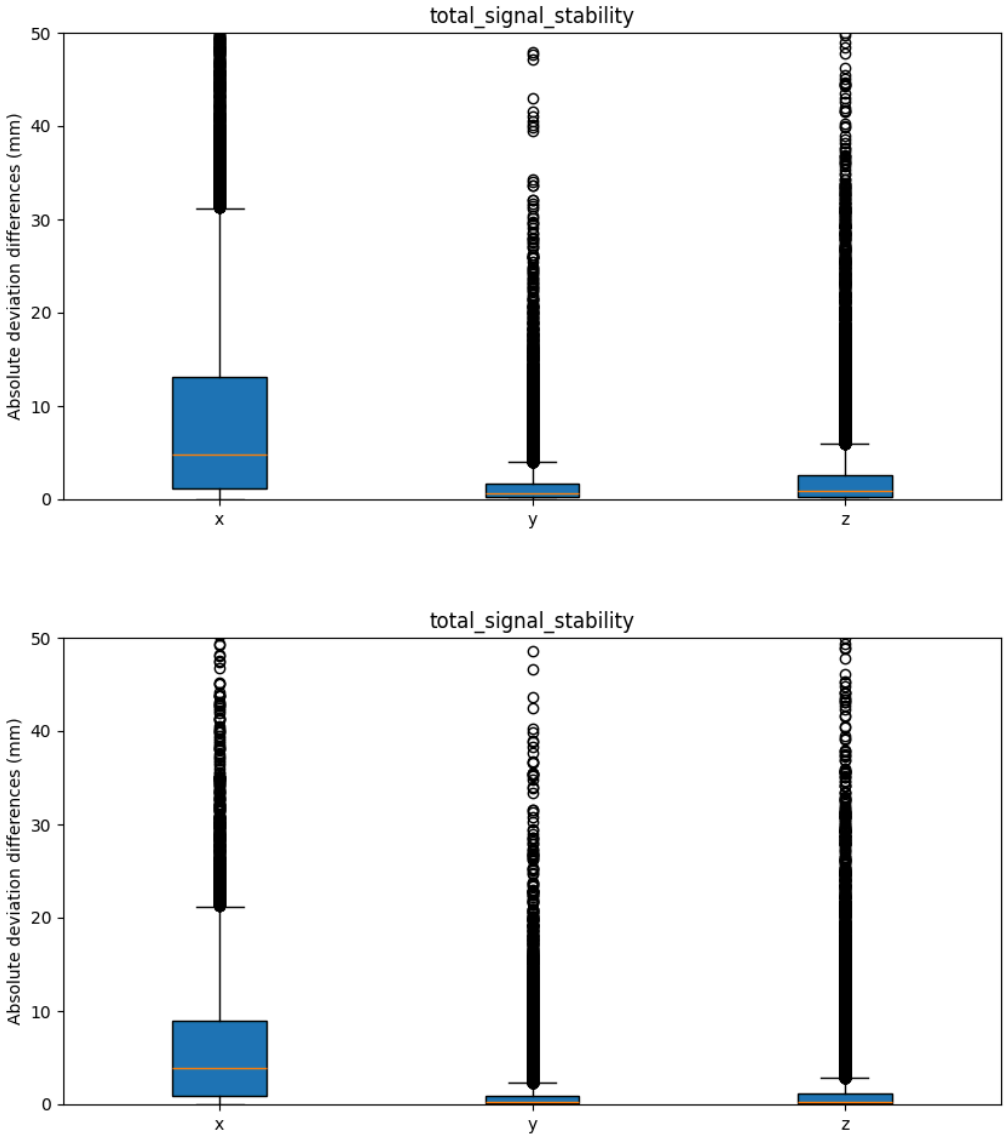


Figure 27: The signal stability from the `maurice_drum_fast` measurement in a boxplot, without processing (top) and with processing (bottom). Model: `LITE`, Marker type: `Landmark`

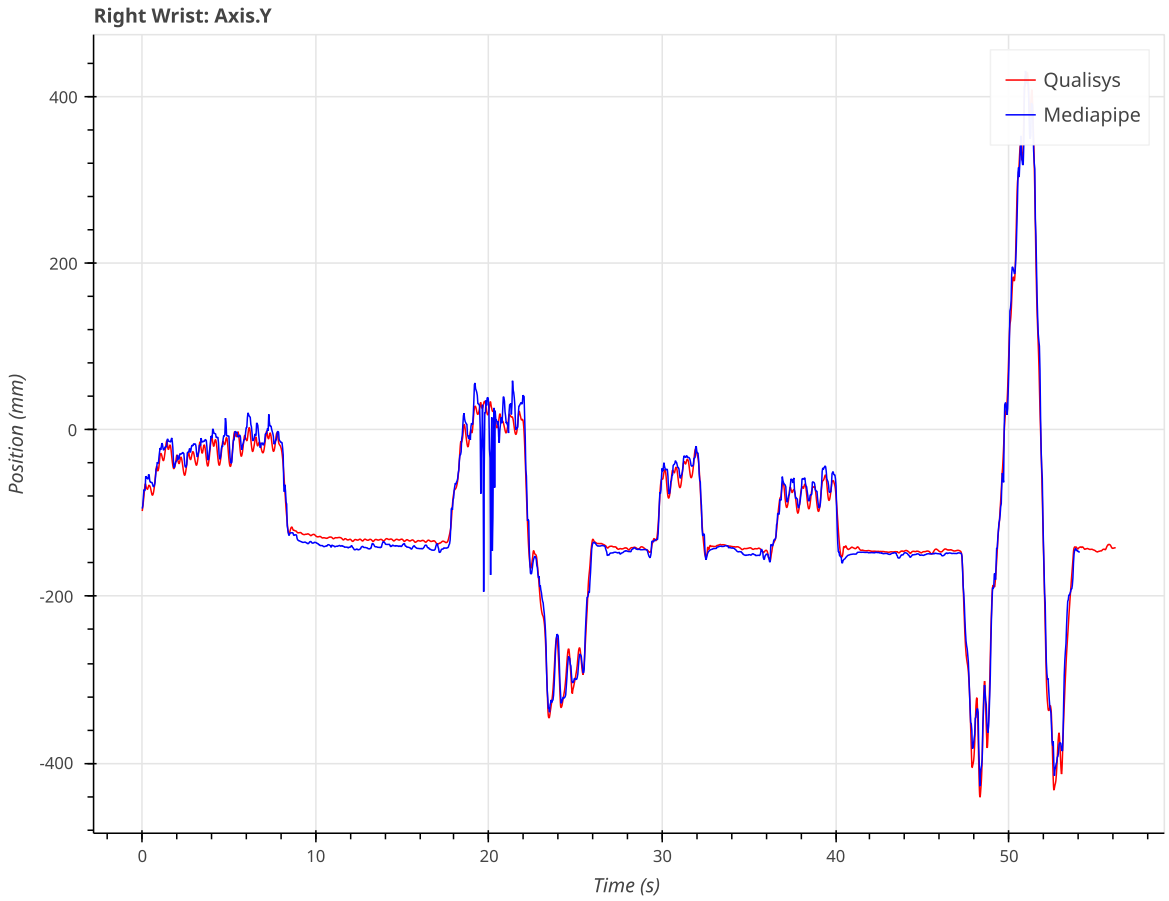
Deviation unprocessed (mm)	X	Y	Z
mean	64.872857	7.181918	10.046277
std	70.935324	9.058099	14.284985
min	0.006952	0.000276	0.000796
25%	9.690526	2.168622	2.386345
50%	31.462373	4.908393	5.571245
75%	113.099430	8.903545	12.760656
max	680.599720	226.170262	592.765323

Deviation processed (mm)	X	Y	Z
mean	62.469941	7.216940	10.001153

Deviation processed (mm)	X	Y	Z
std	70.122023	9.067662	14.967685
min	0.000379	0.000839	0.000065
25%	9.050388	2.007845	2.220629
50%	27.936964	4.717725	4.811401
75%	112.598154	8.676960	12.744544
max	679.639393	146.986826	594.480109

Table 21: The deviation from the `maurice_drum_fast` measurement without processing (top) and with processing (bottom). Model: `LITE`, Marker type: `Landmark`

Lastly, we can see the reduction in jitter clearly in the Y axis, trajectory plots from the Right Wrist marker in Figure 28. The jitter around the 20-second marker is reduced, and the trajectory is smoother. The method has a positive impact on the jitter.



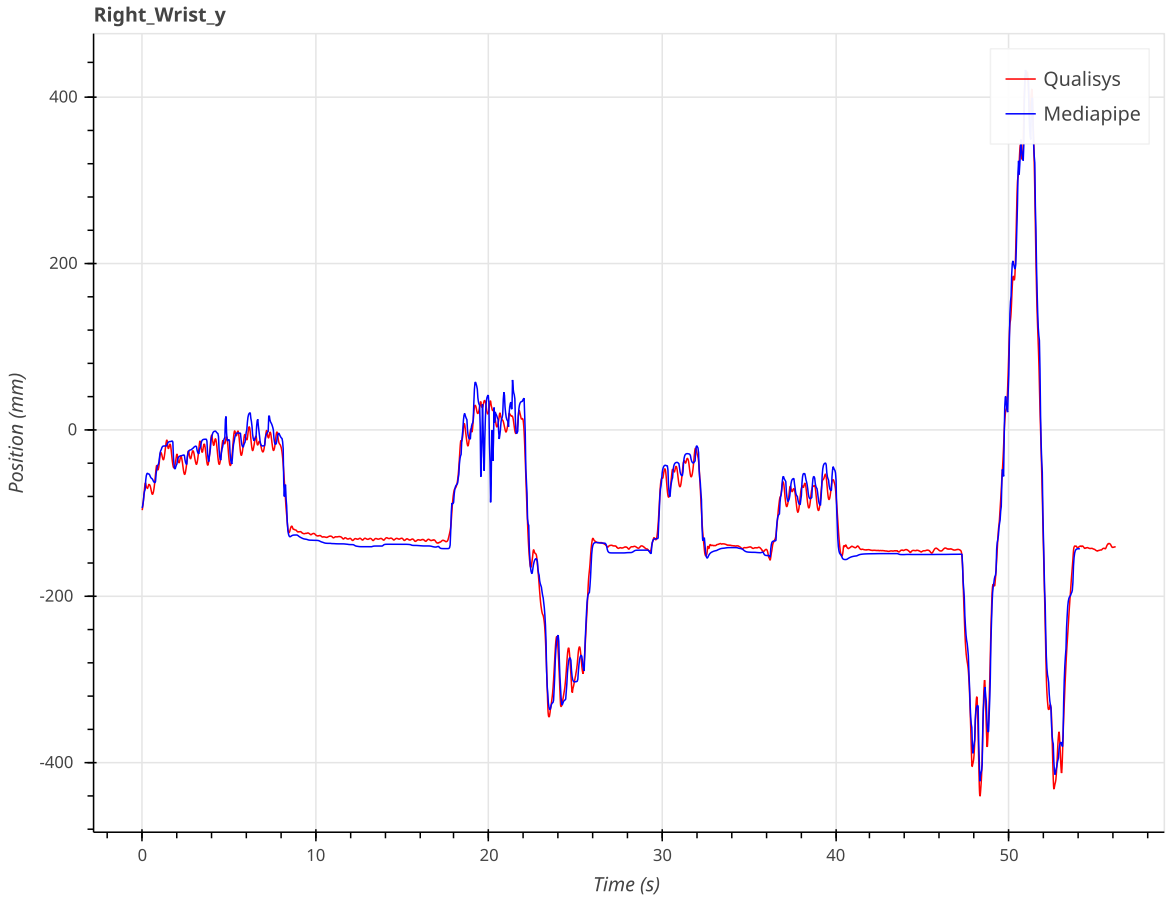


Figure 28: The signal stability from the `maurice_drum_fast` measurement in a boxplot, without processing (top) and with processing (bottom). Model: `LITE`, Marker type: `Landmark`

A final note on the memory size of the method. The above results are achieved with a memory size of 2 frames. This is the minimum memory size that can be used, as we need at least two frames to calculate the direction vector $\vec{v}$. The explanation of the method already made clear a concern that increasing the memory size might make the method less responsive. During the experimentation, we found this concern to be true. Setting the memory size to 4 caused some markers to lag behind the changes in the actual movement. Even higher memory sizes could cause the markers to lag even more. The method is most effective with a memory size of 2 frames at 30 fps.

5.2.2 60 fps

The `maurice_drum_60fps` recording at 60 fps is used to compare the results. The following results are with parameters set to a memory size of 2 frames, the peak $d = 0.01$, and the tightness $s = 0.7$. The results are displayed in Table 22. Just like with the 30 fps recording, the method has a positive impact on the signal stability. However, the impact is less pronounced than with the 30 fps recording. This is somewhat surprising, as it was expected that the method would have a more significant impact on the 60 fps recording. More frames per second should allow the method to better predict the direction vector $\vec{v}$ and

a narrower interpolation function. As the frames are closer together, the possible range of movement should be smaller.

Stability unprocessed (mm)	X	Y	Z
mean	6.084695	1.456349	1.782536
std	7.498098	3.513297	3.320853
min	0.000074	0.000007	0.000027
25%	1.302660	0.168504	0.187517
50%	3.542529	0.510763	0.676077
75%	8.028906	1.362623	1.968414
max	102.800047	122.740564	83.073242

Stability processed (mm)	X	Y	Z
mean	4.251094	1.277513	1.539274
std	4.003413	3.142163	3.161759
min	0.000385	0.000002	0.000005
25%	1.167122	0.082504	0.075393
50%	3.173341	0.344433	0.348346
75%	6.307637	1.050543	1.529996
max	38.699978	63.47007	47.21325

Table 22: The signal stability from the `maurice_drum_60fps` measurement without processing (top) and with processing (bottom). Model: `LITE`, Marker type: `Landmark`

At 60 fps it is however possible to increase the memory size without causing the markers to lag behind the actual movement. Using more frames to predict the current position leads to a more stable prediction. But even at just 4 frames it is shown that the prediction is a bit too stable. The stability shown in Table 23 is slightly worse than with a memory size of 2 frames.

Stability processed (mm)	X	Y	Z
mean	4.209328	1.327719	1.634284
std	4.037256	3.022464	3.120164
min	0.000018	0.000010	0.000007
25%	1.179050	0.136544	0.149714
50%	3.084383	0.431487	0.511141
75%	6.089892	1.167036	1.695795
max	40.776414	54.463532	53.594910

Table 23: The signal stability from the `maurice_drum_60fps` measurement with a memory size of 4 frames with processing. Model: `LITE`, Marker type: `Landmark`

A possible explanation for the less pronounced impact of the method on the 60 fps recording could be attributed to the jitter. When closely looking at the jitter, we can see that the duration (in time) is the same across the 30fps and 60fps recordings. For example where the jitter would last 2 frames in the 30 fps recording, it would last 4 frames in the 60 fps recording. This means that the method has less of an impact on the 60 fps recording, as the jitter is already less pronounced. At 60 fps the jitter has a smoother curve instead of the sharp peaks at 30 fps.

As with the 30 fps recording, the method does not have a negative impact on the accuracy.

5.2.3 Conclusion

At 30 frames per second the method has a positive impact on the signal stability and the jitter. The same is true for the 60 frames per second recording, but the impact is less pronounced due to the smoother, stretched out, jitter.

The method does not have a negative impact on the accuracy. The method does not improve the accuracy, but it does not worsen it either.

6 The Air Drumming Application (DrumPy)

The Air Drumming application is a demo application that showcases the use of on-device body pose estimation.[21] 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 following section describes the key components of the application without going into too much detail. For a more in-depth explanation, please refer to the source code and documentation.[22]

The application is designed to be easy to use and understand, with a simple command-line interface and graphical user interface. 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 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. Every

[21] DrumPy is the official application name, referring to the main technology, Python, with which it is made.

[22] The application and source code are publicly available on GitHub: *Mouwrice/DrumPy* Note that this application is as much part of this thesis as the measurements and results presented in the previous sections, and should be considered as such.

calibration step along with information about the detected hit is outputted to the user in the console as shown in the example below, Listing 3.

For more usage information along with an installation guide, please refer to the README file in the source code repository.[23]

```
Snare Drum calibration start                                    console

Calibrating Snare Drum
        Hit count: 0

Volume: 1.000
Left Wrist: Snare Drum
Distance: 0.000
Velocity: -0.867
Position: [-0.128, 0.099, -0.827]


Calibrating Snare Drum
        Hit count: 1

Volume: 1.000
Left Wrist: Snare Drum
Distance: 0.006
Velocity: -1.056
Position: [-0.148, 0.110, -0.824]


Calibrating Snare Drum
        Hit count: 2

Volume: 1.000
Left Wrist: Snare Drum
Distance: 0.015
Velocity: -1.101
Position: [-0.188, 0.126, -0.830]
```

Listing 3: Console output of the calibration process.

6.1 Technologies

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

- MediaPipe: For on-device body pose estimation.

[23] *https://github.com/Mouwrice/DrumPy?tab=readme-ov-file#installation*

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

The application has been tested on both Linux (Arch Linux) and Windows 11. It should work on other platforms as well, but this has not been tested. Note that unfortunately, GPU support is not available for the MediaPipe library on Windows, so the application will run on the CPU only. This may result in lower performance compared to running the application on a Linux machine with GPU support and a dedicated GPU.

With the help of Nuitka,[24] the application can be compiled into a standalone executable that can be run on any system without the need to install Python or any of the required libraries. This makes it easy to distribute the application to users who do not have Python installed on their system. The compiled executable is available in the GitHub repository for easy download and use. Unfortunately, the compiled executable is only available for Windows currently, as the binary for Linux did not work as expected.

The code repository has been automated with GitHub Actions to automatically build and release the compiled executable for Windows whenever a new release is created. This makes it easy to keep the compiled executable up to date with the latest changes in the codebase.

6.2 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.

From the measurement results, we know that the depth values are not reliable and therefore are not used in the hit detection and drum positions. This means that the entire air drumming application works at a 2D level, it does not make use of any depth information. This is a limitation of the application but has not proved to be a problem in practice as most drumming gestures are made in the same plane.

[24] Nuitka is a Python compiler that can compile Python code into standalone executables. *https://nuitka.net/*

From the measurements, we also know that `WorldLandmarks` are less reliable than the regular `Landmarks`. For a more reliable hit detection, the application uses the regular `Landmarks` to detect the drumming gestures and to store the positions of the drum elements. Regular `Landmarks` have coordinates relative to the image frame, while `WorldLandmarks` have coordinates relative to the detected hip midpoint. If the `WorldLandmarks` are used, the drum elements would move with the user, which is not desirable. The drum elements should be fixed in space, so the regular `Landmarks` are used instead.

6.2.1 Hit detection

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.

Consider the following example vertical trajectory of a marker, Figure 29. The hit detection algorithm looks for a downward movement followed by an upward movement. Or in other words, it looks for a peak or breakpoint in the vertical trajectory. When such a peak is detected, a hit is registered. It does so by keeping a memory of a certain number of previous positions and calculating the trend of the vertical position. Given a range of positions, a downward trend is detected when the average decrease in vertical position is greater than a certain threshold. An upward trend is detected when the average increase in vertical position is greater than a certain threshold. This allows to tweak the sensitivity of the hit detection algorithm. For example, a higher threshold would require a more pronounced downward movement to be detected as a hit. The threshold can also be lowered to allow for more subtle movements to be detected as hits. This is the case for the hit detection of the feet. The feet are tracked to detect the kick drum pedal. When using a pedal, the user might not immediately lift their foot after hitting the pedal. Setting the upward threshold to a lower value allows detecting the hit even when there is no real upward movement. The upward threshold is even set to a negative value, but not as low

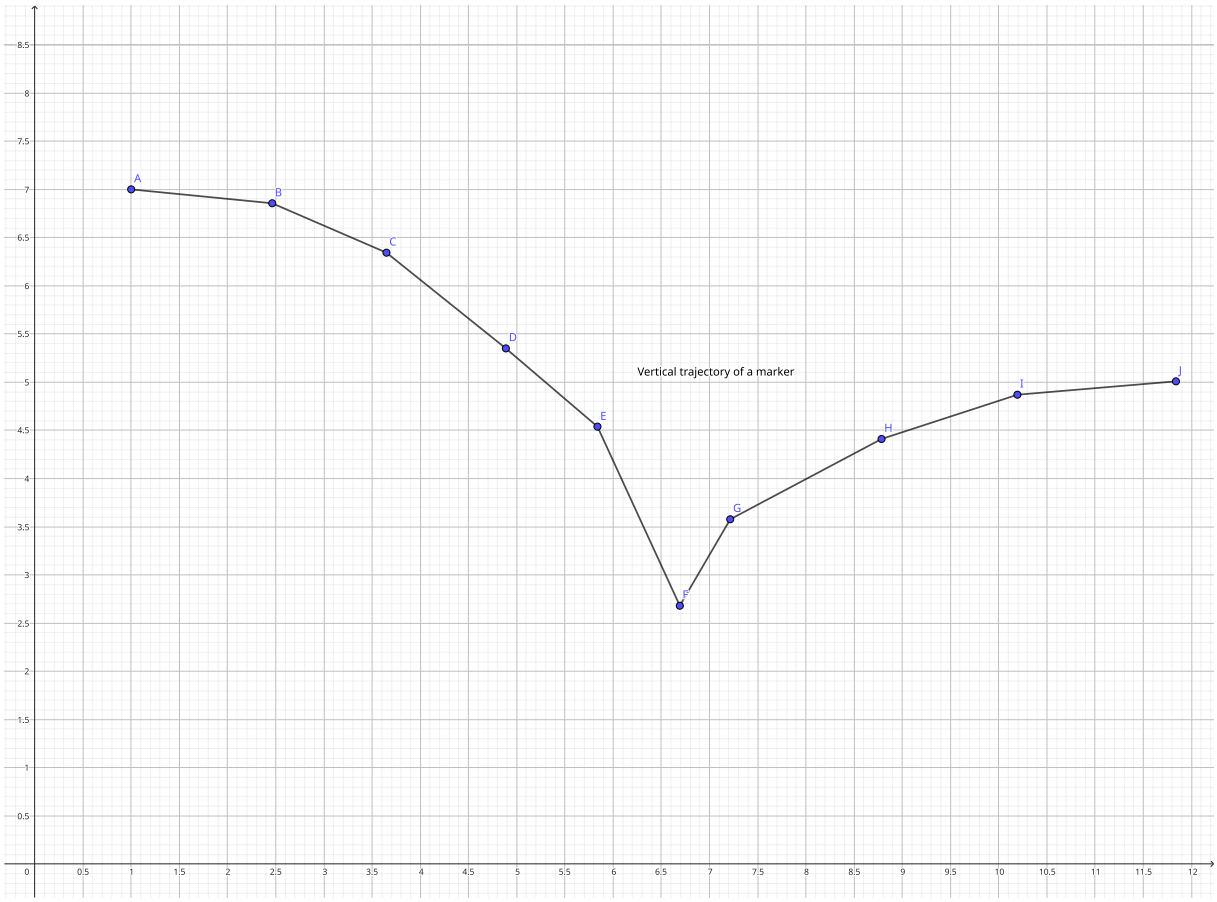


Figure 29: Example vertical trajectory of a marker. Every dot represents a given position at a given time. The horizontal axis is time, the vertical axis is position.

as the downward threshold. Now a foot induced hit can even be detected when the foot is still slightly moving downwards after the hit.

Another important aspect that affects the hit detection is the notion of the current position. When the algorithm receives the position of marker F from the example (Figure 29), it has no way of knowing that this point F is the “hit point”. It would need to be able to look ahead. Of course, this is not possible, this is why the current position is defined as being some time in the past. For example, if we take the current position to be the position of the marker 2 frames ago, we have the artificial ability of looking ahead 2 frames to detect if an upward trend follows the current position. Note that there is a tradeoff between the current position and the number of frames used to detect the upward trend. The more frames are used to detect the trend, the more accurate the hit detection will be, but the less responsive it will be. This is because the hit detection will be delayed by the number of frames used to detect the upward trend. The current position is a compromise between accuracy and responsiveness.

6.2.2 Finding the nearest drum element

Every element of the drum set has a position which is set during the calibration fase. During the calibration any detected hit will result in the position of the element to be updated and the calibrated drum element to play its sound.

After a hit is registered, a corresponding drum element needs to be found. The application calculates the distance between the hit position and the position of each drum element. The drum element closest to the hit position is considered to be the drum element that has been hit. This is a simple way to determine which drum element has been hit and works well in practice. The distance is calculated using the Euclidean distance formula, which calculates the distance between two points in 2D space (the location of the hit and the location of the drum element). There is also a threshold distance that is used to determine if a hit is close enough to a drum element to be considered a hit on that drum element. If the distance between the hit position and the drum element is less than the threshold distance, the hit is considered to be on that drum element. This threshold distance can be adjusted to make the hit detection more or less sensitive. A lower threshold distance will require the hit to be closer to the drum element to be considered a hit, while a higher threshold distance allows hits that are further away to be considered hits on the drum element. This can be useful to fine-tune the hit detection to the user's preferences and playing style. It would be unrealistic if the user would trigger a drum sound when hitting the air far away from the drum element. Setting the threshold too low, however, might result in missed hits. Especially when the threshold gets close to the noise level of the measurements, the hit detection might become unreliable.

6.2.3 Latency

In this section, we will discuss the latency of the application. Latency is the time it takes for a user to perform an action and see the result of that action. In the context of the Air Drumming application, latency is the time it takes for the user to hit a drum and hear the drum sound. Latency is an important factor in interactive applications, as high latency can reduce the responsiveness of the application and make it less engaging for the user.

The latency of the Air Drumming application is affected by several factors:

- The frame rate of the camera: The frame rate of the camera determines how often the application receives new video frames. A higher frame rate allows the application to process new frames more frequently, reducing the latency of the application. The frame rate of the camera is set to 30 frames per second in the application, which is a common frame rate for video capture. Especially, most webcams support this frame rate.
- The processing time of the body pose estimation model: The body pose estimation model processes each video frame to estimate the 3D pose of the person. The process-

ing time of the model depends on the complexity of the model and the hardware it is running on. A more complex model or slower hardware will increase the processing time of the model, increasing the latency of the application. The MediaPipe Pose model used in the application is optimized for real-time performance on mobile devices and runs efficiently on most modern hardware.

- The hit detection algorithm: The hit detection algorithm processes the estimated 2D pose of the person to detect drumming gestures. The algorithm analyses the trajectory of the marker to detect downward and upward movements, which indicate a hit on a drum element. The hit detection algorithm is designed to be fast and efficient, yet it requires some frame buffer to detect the upward trend. This buffer introduces a delay in the hit detection, which affects the latency of the application. The size of the frame buffer can be adjusted to balance accuracy and responsiveness.

So the first limitation is the frame rate of the camera. The application has the frame rate set at 30 fps. This translates to a video frame every 33,33... ms. The application processes the video frame and estimates the pose of the person in the frame. This processing time is the second limitation. However, during the measurements, it was found that the processing time of the pose estimation model is very low. The pose estimation model runs at around 30 fps on a CPU. This means that the pose estimation model processes a frame in less than 33,33... ms. The hit detection algorithm is the third limitation. The hit detection algorithm requires a frame buffer to detect the upward trend. This buffer introduces a delay in the hit detection, which affects the latency of the application. The size of the frame buffer can be adjusted to balance accuracy and responsiveness. The buffer is set to 2 frames, which means that the hit detection is delayed by 2 frames. This results in a latency of 66,66... ms. Now we can calculate the maximum latency of the application.

Now we can calculate the maximum latency. For the pose estimation model to process the frame, suppose the worst-case scenario, but still achieving 30 fps. The processing time of the model is then 33,33... ms. Finally, the hit detection algorithm introduces a delay of 2 frames, resulting in a delay of 66,66... ms. This results in a maximum latency of 100 ms. This maximum latency is a tenth of a second and might be noticeable by the user. However, if the inference time of the model is lower, the latency will be lower as well. The latency can be further reduced by increasing the frame rate of the camera or optimizing the hit detection algorithm. The latency of the application is acceptable for most users and does not significantly affect the user experience.

6.3 Notion of Origin and Coordinate System

This section briefly discusses the differences in the notion of origin and coordinate system between the MediaPipe library and typical motion capture systems. With the Qualisys motion capture system, the origin is calibrated at one specific point in the room. The origin is fixed and does not move during the capture session. The origin is used as a

reference point to calculate the position of the markers in 3D space. The position of the markers is calculated relative to the origin, which is why the origin is an essential part of the motion capture system.

In the MediaPipe library, the origin depends on the marker type used. With the `Landmark` marker type the entire coordinate system is in the image frame. The origin point is one of the corners of the frame and every other point is relative to this origin in the frame. This has major implications on the perceived size of motion.

7 Future work

In this section, some ideas are listed that could be interesting to explore in the future.

7.1 Increasing signal stability

The signal stability has been increased by the method introduced previously, reducing the jitter and noise. However, the signal is still not perfect. It would be interesting to explore other methods to increase the signal stability even further.

As it stands, the method is based on a simple prediction model combined with an interpolation method between the predicted and the measured value. This can be generalized to a statistical problem where the goal is to predict a state given a previous state, together with the uncertainty of the prediction. One could use a Kalman filter to estimate the next state and its uncertainty. The Kalman filter can work as a two-phase process where the first phase is the prediction given the previous state and the second phase is the update phase given the measured value [26]. This would allow making a more informed decision on how to interpolate between the predicted and the measured value.

The prediction of every marker is currently independent of the other markers. However, the markers are not independent of each other. It would be interesting to explore methods that consider the dependencies between the markers. Constructing a skeleton model of the human body that takes into account the dependencies and constraint between the markers could be a way to increase the signal stability. For example, the distance between connected markers (by a bone) should be constant. This could be used to correct the predicted value of a marker if the distance between the connected markers is not constant. Another example is that the angle between connected markers is limited to a certain range of values.

7.2 Depth estimation

From the measurements, it was clear that the depth estimation has a low accuracy and suffers from major instability. Future work could focus on improving the depth estimation. This could be achieved in two ways. The first is by simply using a different model or training the computer vision model to more accurately predict the depth. The second way is to use additional sensors to estimate the depth. For example, a depth sensor could be used to measure the depth of the markers. This could be combined with the computer vision model to increase the accuracy of the depth estimation. Or by using a multi-camera setup, the depth could be estimated by triangulating the position of the markers in the different camera views.

7.3 Real-time application

There are various ways to build upon the application. One simple way to improve the application is by simply replacing the currently used MediaPipe model with a better model if one is found. An increase in depth is certainly welcome, but also an increase in performance would be beneficial. Currently, the application runs at around 30 frames per second, which limits the ability to track fast and complex movements. A faster model would allow for a higher frame rate and thus a more responsive application.

Another way to improve the application is by integrating a better prediction model such as the ones mentioned in the previous section.

An entirely different application can be built by using the same principles. Not only is body pose estimation improving, but also hand pose estimation is becoming more accurate. Similar research can be done to discover the feasibility of an application that requires accurate hand and finger tracking. For example, a virtual piano application.

Other future research can focus on the usage of body pose estimation in a different context, such as in a medical rehabilitation setting. Such a system could help medical professionals to analyse the movements of patients. Or it could be used as a tool for patients that need to do exercises at home. The system could provide feedback on the correctness of the exercises and give tips on how to improve the exercises.

8 Conclusion

In this thesis, we have presented a method to compare body tracking data from a computer vision model with a motion capture system. Using the method we have shown that an average accuracy of 5-10 mm can be achieved with MediaPipe Pose for the movements relative to the camera. We have shown that the accuracy of the depth estimation is a low lower and suffers from major instability. MediaPipe Pose is able to track the movements of the body in real-time with a frame rate of around 30 frames per second on consumer hardware. And reaching up to 45 fps on a powerful GPU. 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 application was able to track the movements of the user in real-time and play the drum sounds accordingly. The simple yet very effective procedure for detecing drum hits has been described as well. The application technology stack consists of MediaPipe Pose for the body pose estimation, PyGame for the visualisation and sound, and Python for the application logic. The application and its source code have been made publicly available for download and can be found on GitHub.

The future of body pose estimation is promising. The accuracy and performance are shown to already be sufficient to construct a simple 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. We have seen that body pose estimation can bring the expensive and complex technology of motion capture to consumer hardware. This opens up a whole new world of possibilities for applications that require body tracking.

References

[1] Jian Liu *et al.*, “Deep Learning-Based Object Pose Estimation: A Comprehensive Survey,” May 2024.

[2] Vladimir Mandic, “Human: AI-powered 3D Face Detection & Rotation Tracking, Face Description & Recognition, Body Pose Tracking, 3D Hand & Finger Tracking, Iris Analysis, Age & Gender & Emotion Prediction, Gaze Tracking, Gesture Recognition.” [Online]. Available: *https://github.com/vladmandic/human*

[3] Vladimir Mandic, “Human: 3D Motion Visualization.” [Online]. Available: *https://github.com/vladmandic/human-motion*

[4] Sterzentsenko *et al.*, “A low-cost, flexible and portable volumetric capturing system.” [Online]. Available: *https://vcl3d.github.io/VolumetricCapture/*

[5] OpenMMLab, “OpenMMLab Pose Estimation Toolbox and Benchmark..” [Online]. Available: *https://github.com/open-mmlab/mmpose*

[6] Jiang *et al.*, “RTMPose: Real-Time Multi-Person Pose Estimation based on MMPose.” [Online]. Available: *https://arxiv.org/abs/2303.07399*

[7] 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*

[8] Fang, Hao-Shu and Xie, Shuqin and Tai, Yu-Wing and Lu, and Cewu, “RMPE: Regional Multi-person Pose Estimation.” 2017.

[9] Google, “MediaPipe Audio Classification.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/solutions/vision/pose_landmarker*

[10] Google, “MediaPipe.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe*

[11] Google, “MediaPipe Solutions.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/solutions/guide*

[12] Google, “MediaPipe Audio Classification.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer*

[13] Google, “MediaPipe Audio Classification.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/solutions/vision/object_detector*

[14] Google, “MediaPipe Text Classification.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/solutions/text/text_classifier*

[15] Google, “MediaPipe Audio Classification.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/solutions/audio/audio_classifier*

[16] Google, “MediaPipe Framework.” Accessed: Apr. 30, 2024. [Online]. Available: *https://developers.google.com/mediapipe/framework*

[17] V. Bazarevsky, I. Grishchenko, K. Raveendran, T. Zhu, F. Zhang, and M. Grundmann, “BlazePose: On-device Real-time Body Pose tracking.”

[18] H. Xu, E. G. Bazavan, A. Zanfir, W. T. Freeman, R. Sukthankar, and C. Sminchisescu, “GHUM & GHUML: Generative 3D Human Shape and Articulated Pose Models.”

[19] COCO Consortium, “COCO 2020 Keypoint Detection Task.”

[20] Google, “On-device, Real-time Body Pose Tracking with MediaPipe BlazePose.” Accessed: May 04, 2024. [Online]. Available: *https://research.google/blog/on-device-real-time-body-pose-tracking-with-mediapipe-blazepose/*

[21] A. Bulat and G. Tzimiropoulos, “Human pose estimation via Convolutional Part Heatmap Regression.”

[22] Google, “Model Card: MediaPipe BlazePose GHUM 3D.”

[23] Paul Lindner, “Definition of tab-separated-values (tsv).” Jun. 1993. Accessed: Apr. 27, 2024. [Online]. Available: *https://www.iana.org/assignments/media-types/text/tab-separated-values*

[24] J. Kiefer, *Sequential minimax search for a maximum*. 1953.

[25] Wikipedia, “Golden-section search.” Accessed: Apr. 28, 2024. [Online]. Available: *https://en.wikipedia.org/wiki/Golden-section_search*

[26] Wikipedia Contributors, “Kalman Filter.” Accessed: May 20, 2024. [Online]. Available: *https://en.wikipedia.org/wiki/Kalman_filter*