

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

**Musical Instrument Animation
Blender Add-on**

Bachelor's Thesis

HANA DRTÍLKOVÁ

Brno, Spring 2025

**M A S A R Y K
U N I V E R S I T Y**

FACULTY OF INFORMATICS

**Musical Instrument Animation
Blender Add-on**

Bachelor's Thesis

HANA DRTÍLKOVÁ

Advisor: Mgr. Jiří Chmelík, Ph.D.

Department of Visual Computing

Brno, Spring 2025



Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

During the preparation of this thesis, I used ChatGPT for grammar check and to improve the coherence of my writing. I declare that I used this tool in accordance with the principles of academic integrity. I checked the content and took full responsibility for it.

Hana Drtílková

Advisor: Mgr. Jiří Chmelík, Ph.D.

Acknowledgements

First of all, I would like to thank my supervisor Mgr. Jiří Chmelík, Ph.D. for guiding me through the process of writing a bachelor's thesis. Secondly, and equally importantly, I would like to extend my sincere thanks to Mgr. Jakub Slezáček for coming up with the topic of this thesis and willingly advising me when I was lost in the specifics of motion capture and peculiarities of the Blender Python API. Most of all, I would like to thank my boyfriend for supporting me through all the ups and downs, assisting me during the motion capture recording process, and allowing me to capture his lovely guitar playing as a part of this thesis.

Abstract

Marker-based optical motion capture is a powerful tool capable of accurately recording a wide range of performances. However, for musical performances, it is crucial to accurately capture subtle but precise finger movements to produce believable animations. As of now, current marker-based motion capture techniques do not offer a practical solution that does not impair the dexterity of the musician's fingers.

Jakub Slezáček's master's thesis *Violin Performance Motion Capture* presents a solution for capturing the motion of a performing violinist without limiting their freedom of movement. The author developed a workflow where the full-body movement is tracked by an optical motion capture system, whereas the hand motions are recorded with an RGB camera and estimated using a pre-trained AI model. The data is then joined within a custom Blender add-on called *Musical Instrument Animation* to create an animated armature.

This thesis extends the existing add-on by implementing a responsively generated overlay of the estimated hand poses displayed over a reference video. Furthermore, it provides an exemplary motion capture dataset to demonstrate the usability of the add-on.

As a result, the Blender add-on is now more effective and user-friendly by offering immediate visual feedback on the accuracy of hand pose estimation and a reference to the actual desired hand positions. Additionally, as the recorded dataset features different instruments other than the violin, the thesis explores the current limitations of the *Violin Performance Motion Capture* system and aids future development of the original *Musical Instrument Animation* Blender add-on, as the reference is a powerful debugging tool and the dataset provides valuable testing data.

Keywords

Blender, HCI laboratory, motion capture

Contents

Introduction	1
1 Musical Performance Motion Capture	3
1.1 Used Technologies	3
1.1.1 Motion Capture	3
1.1.2 OptiTrack and Motive	4
1.1.3 Hand Capture	4
1.1.4 Blender	5
1.2 The State of Musical Performance Motion Capture . . .	5
1.2.1 The Workflow of the Violin Performance Motion Capture	5
1.2.2 The Limitations of the Current Musical Perfor- mance Motion Capture System	7
2 Reference Add-on Implementation	9
2.1 Video Reference	10
2.1.1 Importing the Video Reference	12
2.2 Estimated Hand Poses Overlay	13
2.2.1 Accessing the Estimated Hand Poses Data	14
2.2.2 Generating the Overlay Images	14
2.2.3 Placing the Overlay Images into the Reference Video	17
2.3 User Experience	20
2.3.1 User Interface	22
3 Motion Capture Exemplary Dataset	25
3.1 Motion Capture Recording Process	25
3.1.1 Hand Capture Configuration	25
3.1.2 Attachment of Markers to the Instruments	28
3.2 Motion Capture Results	29
4 Future Work	34
4.1 In-Scene Reference Video	34
4.2 Parallel Hand Pose Image Generation	35
Conclusion	36

A Electronic Attachments	37
Bibliography	38

List of Figures

1.1	Violin and bow marker placements [1, p. 14-15].	6
1.2	The final result after aligning the bow and violin to the armature [1, p. 20].	7
2.1	The 21 hand landmarks from the <i>Hand landmarks detection</i> <i>guide</i> [16].	9
2.2	The estimated hand poses JSON file structure.	10
2.3	The Video Sequencer is occupied warning dialogue. . . .	12
2.4	The video reference displayed through the Preview. . . .	13
2.5	An example of a generated hand pose image.	16
2.6	A diagram of <i>Edge Alignment</i>	17
2.7	A diagram of <i>Centre Alignment</i>	19
2.8	A screenshot of a Sequencer with fully loaded hand pose image strips in <i>Centre Alignment</i> mode.	20
2.9	A screenshot of a video reference frame with the estimated hand pose overlay.	20
2.10	The <i>Estimated Hand Poses Reference</i> panel.	22
2.11	The <i>Overlay Settings</i> panel.	23
3.1	The bassoon thumb keys.	27
3.2	A screenshot from the Hand Capture web application. . .	27
3.3	Marker placements on the recorder.	28
3.4	The marker placements on the bassoon and the guitar. . .	30
3.5	An example of bone twisting caused by an incorrectly imported hand pose.	32
3.6	An example of bone twisting resulting from an incorrectly estimated hand pose.	32

Introduction

When a musician plays a musical piece at the speed of 120 BPM¹, the frequency of sixteenth notes² is 8 notes per second. That means the position of their hands can change 8 times during a mere second of the performance. As animating such rapid precise movements manually is time consuming, using motion capture could significantly quicken the animation process while providing realistic results.

Jakub Slezáček's master's thesis *Violin Performance Motion Capture* [1] introduces a solution for capturing the motion of a performing violinist, including their fingers, with a special care for not limiting the violinist's freedom of movement. The author developed a workflow where the full-body movement is tracked by an optical motion capture system whereas the hand motions are recorded with an RGB camera through a web application. The application also takes care of processing the video and estimating the hand poses using a pre-trained AI model. The data is then joined within a custom Blender add-on called *Musical Instrument Animation* to create an animated armature.

Although Slezáček's solution produces quality data, polishing the resulting armature manually is usually necessary. A lack of visual reference of the estimated hand poses and the recorded video makes adjusting the animation a tedious task. Hence, the main goal of my thesis is to extend the existing add-on to provide a reference video with an overlay of the estimated hand poses. The reference should be synchronized with the playback of the animation in the scene and the overlay should be computed in real time.

Another significant part of my thesis is to create a set of testing data by capturing different musical performances.

As I am building upon Jakub Slezáček's master's thesis, the technologies used in my thesis have been chosen to be the same. Namely, OptiTrack Motion Capture system was used to produce the testing data and Blender Python API was used to extend the add-on and develop it further.

1. Beats per minute.

2. In a quarter note based time signature, i.e., a quarter note corresponds to one beat.

The thesis is structured in the following way: Chapter 1 introduces the used technologies and describes the current state of musical performance motion capture, with a focus on the workflow developed by Slezáček. Chapter 2 discusses the implementation of the reference add-on and discloses the details of the development process. Chapter 3 focuses on the exemplary motion capture dataset, describing the recording process and results. Finally, Chapter 4 proposes potential future enhancements to the developed reference add-on.

1 Musical Performance Motion Capture

The core of a musical performances is the music itself; however, the movements of the performer and how well they reflect the music, play an important role in amplifying the audience’s emotional engagement with the performance [2, p. 20]. Therefore, it is important to capture not only the finger movements but also the full-body motion of the musician.

Marker-based optical tracking systems offer an accurate and versatile means of recording the full-body movements, but they are generally unsuitable for capturing finger movements during a musical performance. Although there are marker-based solutions that address the problems associated with capturing hand movements—such as frequent marker occlusion or the complexity of hand motion, which requires a large amount of markers to capture accurately—it is still necessary to attach markers to the fingers or to wear a glove [3, 4, 5]. This can impair the dexterity of the musician’s fingers or even make playing the instrument impossible.

Jakub Slezáček’s solution avoids these issues by capturing hand movements with a standard RGB web camera and estimating the positions through the use of an AI model. As this thesis focuses on improving a specific part of his solution, the following sections introduce the technologies used at each stage before describing the full workflow developed by Slezáček.

1.1 Used Technologies

1.1.1 Motion Capture

Motion Capture (MoCap) is an umbrella term for technologies that allow to obtain data about a subject’s position and to track its movements [6]. As MoCap can refer to different technologies, a definition that does not include the details about the used technology appears to be the most timeless and accurate. Menache defines it in his book *Understanding Motion Capture for Computer Animation* in the following way: “*Motion capture is the process of recording a live motion event and*

translating it into usable mathematical terms by tracking them to obtain a single three-dimensional (3D) representation of the performance” [7, p. 2].

1.1.2 OptiTrack and Motive

OptiTrack is a company which specializes in motion tracking technology [8]. The MoCap system used in this thesis is their marker-based optical system that operates infrared cameras to track the positions of retroreflective markers attached to a subject. When paired with *Motive*, their MoCap software, the data from the IR cameras can be recorded, processed and edited [9].

In *Motive* there are two types of *Tracking Models: Skeleton* and *Rigid Body* assets. *Skeleton* assets allow for inner movement, so their purpose is to track human motion [10]. *Rigid Bodies* are a much less complex asset which represents objects with a constant shape. Each *Rigid Body* has to be defined with a minimum of three markers. *Motive* assumes that the distances between markers stay more or less the same and uses that fact for the computation of the object orientation and position as well as the identification of the *Rigid Body* [11]. On the contrary, *Skeletons* cannot rely on constant distances between the markers throughout the whole asset. They require additional structural information, therefore, to create a skeleton asset, a pre-defined *Skeleton Marker Set* template must be used. With the knowledge of the marker placement locations on the body, *Motive* can divide the body into semi-*Rigid Body* segments to aid its computations [12].

1.1.3 Hand Capture

Hand Capture is a web application developed by Slezáček that estimates hand poses and tracks hand motion from data recorded by an RGB web camera. The application has many useful features including a live preview of the estimated hand poses, adjustable camera settings, and a connection to *Motive* for a synchronized start of a recording within both applications.

The hand pose estimation is computed by Google’s pre-trained AI model *MediaPipe Hand Landmarker task* [1, p. 21]. The application only supports tracking one right and one left hand at once and the model labels the hand types. Therefore, the identification of the hands

is trivial, when made sure there is only one hand of each type visible in the camera view [1, p. 24].

1.1.4 Blender

Blender is an open source software that provides tools for creative work in 3D digital space. It encompasses modeling, animating, rendering and many other useful features. Aside from its large versatility, its merit stems from the extensive Python API that can be used to develop useful add-ons [13, 14].

1.2 The State of Musical Performance Motion Capture

This section describes the results of Slezáček's work which are the starting point for my thesis, therefore this whole Section 1.2, including its subsections, is referencing Slezáček's thesis [1] unless stated otherwise.

In his thesis he used a combination of motion tracking approaches to obtain quality data without limiting the natural movement of a violin player. To capture the full-body motion of the player, he utilized an optical motion capture system, and to track the movement of the player's hands, he created an AI powered hand pose estimation web application that uses video data from an RGB camera.

To process and integrate the obtained data, he created a Blender add-on called *Musical Instrument Animation*, which also features semi-automatic alignment of the violin and bow's 3D models to their respective rigid bodies.

1.2.1 The Workflow of the Violin Performance Motion Capture

Before starting the motion capture itself, retroreflective markers have to be correctly placed on the musician and their instrument. The referenced thesis thoroughly describes how to place these markers on the violin and bow as seen in Figure 1.1 in order to support the Blender add-on's alignment algorithm.

For the full-body motion capture almost any marker configuration can be used as long as it does not include finger markers. The sleeves of the MoCap suit are rolled up and the hand markers are placed on

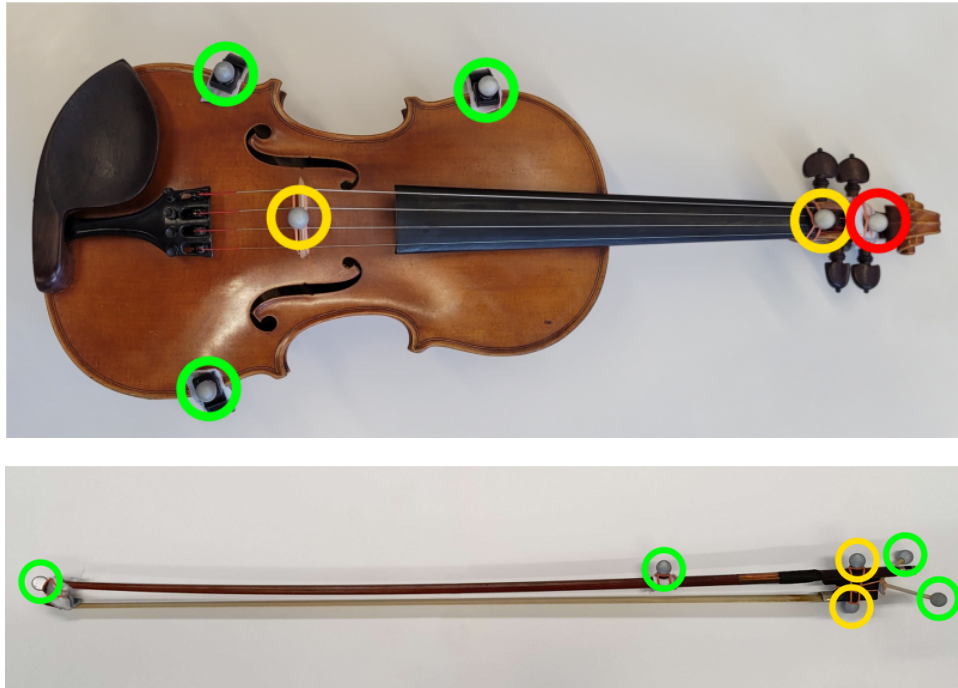


Figure 1.1: Violin and bow marker placements [1, p. 14-15].

a custom holder attached to the hand with skin glue to improve the Hand Capture's recognition of the hand. When all the markers are correctly placed, a skeleton asset is created to track the musician and a rigid body is created to track their instrument.

After configuring a web camera for the Hand Capture and connecting to Motive, the recording can start. The recorded data is then manually post-processed in Motive as it is necessary for any motion capture. When the MoCap inaccuracies are corrected, the data can be imported to Blender in the FBX format¹.

The Hand Capture data is then imported and processed using the custom *Musical Instrument Animation* add-on, which then attaches the hands to the skeleton. Finally, the add-on can be used to align the

1. FBX is a file format widely used for storing 3D data and possibly their motion. For more information, see <https://code.blender.org/2013/08/fbx-binary-file-format-specification>.

violin and bow models with their respective captured rigid bodies. The result is depicted in Figure 1.2.

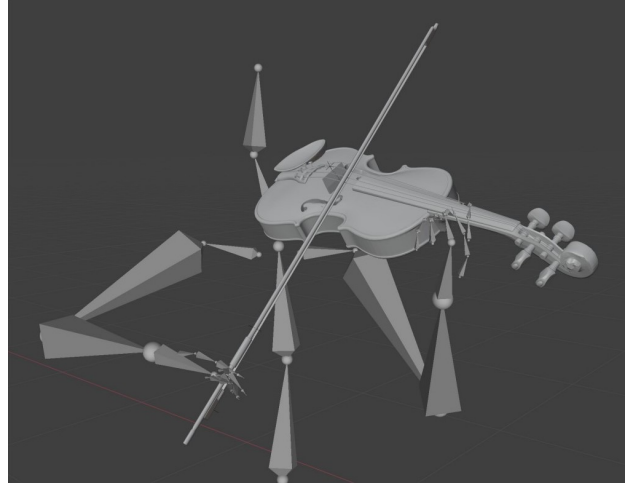


Figure 1.2: The final result after aligning the bow and violin to the armature [1, p. 20].

1.2.2 The Limitations of the Current Musical Performance Motion Capture System

The explained approach generates satisfactory realistic results, but some of the features are limited to capturing a violinist. The placement of the instrument markers is unique to the violin and bow, and the Blender add-on supports only the automatic alignment of their 3D models. That being said, capturing other instruments should be possible if the alignment would be done manually. This thesis challenges this idea, and tests the system's capabilities of recording other instruments besides the violin, which is described in Chapter 3.

Currently, the Hand Capture application is able to record and process only one video stream at once. As a consequence, it is impossible to capture performances of multiple musicians playing together or of a musician whose instrument does not allow for both hands to be seen from one angle at once, such as the Irish bodhrán. It could also pose an issue for recording the bassoon because an angle where some

parts of the hands are not constantly occluded does not exist as further explored in Section 3.1.1.

The weakness that this thesis mainly aims to solve is that although there is a preview with an overlay of the estimated hand poses in the Hand Capture web application, the outputted reference video does not contain the overlay. When polishing the final armature animation and correcting inaccuracies, the lack of an easily accessible estimated hand poses reference is inconvenient and significantly slows down the process. Having a visualization of the JSON data available directly in Blender, synchronized with the playback of the animation, would help the animator quickly identify the incorrectly estimated hand poses, find the cause of errors in the animation, as well as provide a video reference of the desired hand poses.

2 Reference Add-on Implementation

The add-on implementation consists of two parts—the video reference and primarily the estimated hand poses overlay, which are discussed in the following sections.

The starting point is the output data from the Hand Capture web application recording—an unmodified video and a JSON file, which contains the hand poses estimated by the Mediapipe AI model from the recorded video.

More specifically, the JSON file contains a list of hands. Each hand has its name, type—"right" or "left", and animation data which is a chronological list of hand poses found in the video by the model. The hand poses are not periodically spread out throughout the video, instead, each hand pose has a timestamp signifying the time of its occurrence in the video in seconds. The hand pose is defined through a list of 21 hand landmarks coordinates as shown in Figure 2.1. The coordinates come in two forms, world positions and normalized positions, which are equivalent to the world positions normalized by the height and width of the video [15]. The complete structure of the JSON file is depicted in Figure 2.2.

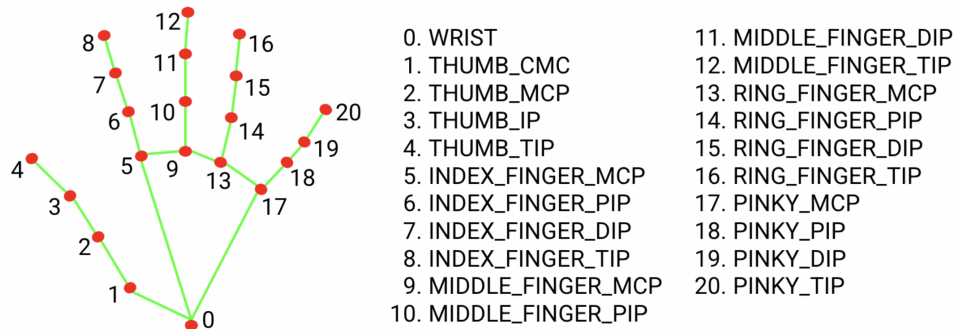


Figure 2.1: The 21 hand landmarks from the *Hand landmarks detection guide* [16].

JSON STRUCTURE

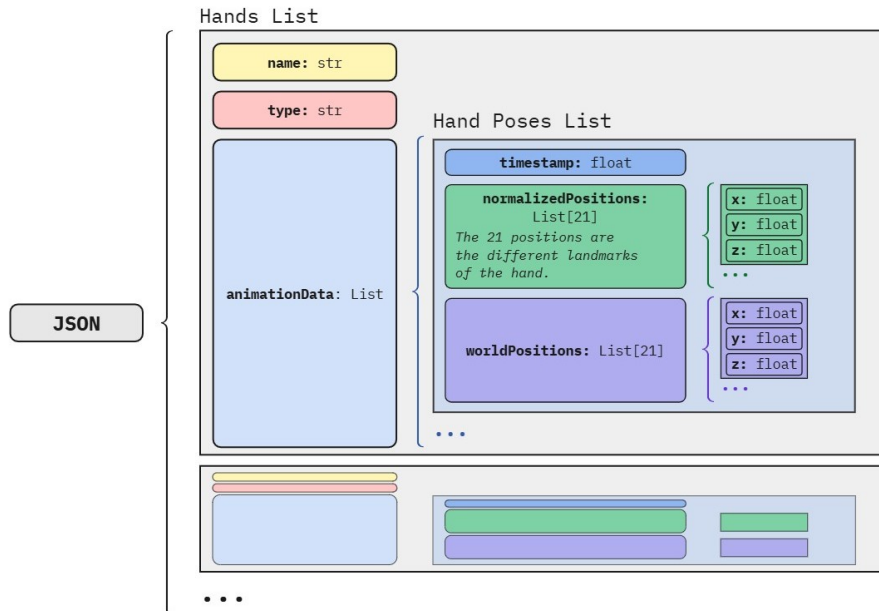


Figure 2.2: The estimated hand poses JSON file structure.

2.1 Video Reference

In Blender there are two reasonable options available for importing a video reference. Firstly, there is the option of importing the video directly to the Scene as a mesh plane, a background or a reference. This method is straightforward and it has the great advantage of being able to move the reference video around in the 3D Viewport freely, even allowing the animator to align the reference right behind the armature for comparison of the hand poses. The second, less conventional, option is using the Video Sequencer¹. As the reference is showed in the Preview view, it is in a separate area and it can be moved around only by adjusting the workspace layout.

1. The Video Sequencer is used for editing videos. For more information see the Blender manual: https://docs.blender.org/manual/en/4.3/editors/video_sequencer/introduction.html

User experience-wise, the first method is arguably the better choice. However, when taking into consideration the necessity to somehow integrate the estimated hand poses overlay into the reference video, the in-Scene option does not offer a practical solution. As there is the requirement for the video reference to operate responsively, overlaying the whole video with the hand poses beforehand and then displaying the already complete video reference is out of the question². The responsiveness could be achieved by separating the creation the overlaid reference into smaller video segments. However with this approach, I had concerns with its time effectiveness, as it would be necessary to render the videos on-the-fly, and with the method of cleanly combining the segments together.

On the other hand, the Video Sequencer has a simple way of responsively creating the overlaid video reference. I also had previous experience with editing videos so the Video Sequencer approach seemed more achievable to me. For those reasons, I decided to use the Video Sequencer to compile and view the video reference.

Taking a closer look at the capabilities of the Video Sequencer approach, there is one other disadvantage to be mentioned besides the inability to float it around in the Scene. As the Video Sequencer is fully used by the add-on, the user cannot use it for anything else in the active Scene. This could seem like a big issue, but in practice, it is unlikely for the user to edit a video in the same project and scene where they are polishing the MoCap data. Still, this rare case is handled by warning the user, as discussed in Section 2.1.1.

This slight downside gets outshined by the benefits of using the Video Sequencer. By adding the reference video to the Sequencer in the same Scene as the MoCap animation data, the synchronization of the video with the data becomes straightforward.

The Video Sequencer is essentially a part of the Scene, so their playbacks are synchronized by default³. By ensuring the video and the MoCap animation both have the same duration in seconds and start at the same frame, the animation and the reference video are completely synchronized. This works only because the MoCap and the

2. Although this thesis does not implement this approach, it is explored further in Chapter 4—Future Work.

3. This holds true even for the in-Scene approach, although adjusting the fps of the reference video to match the Scene’s fps is more problematic.

reference video were also recorded in sync using the Hand Capture's connectivity with Motive.

The greatest advantage of this approach is that the Video Sequencer facilitates iterative modification of the reference video. The add-on uses the video editing tools available in the Sequencer to edit the reference video in the background and gradually build the estimated hand poses overlay as needed. The process of creating the hand poses overlay is fully explained in Section 2.2.

2.1.1 Importing the Video Reference

To import the video reference, the user can press the *Setup Video Reference* button which encapsulates the few processes needed to fully prepare the video reference for the user.

First, the add-on checks the Video Sequencer availability. If the Video Sequencer is already occupied while trying to import the reference video, a warning dialogue pops up. The dialogue notifies the user that their work in the Video Sequencer will be deleted if they proceed, and gives them the option to abort the import.

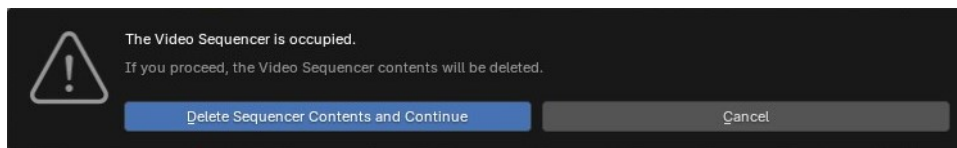


Figure 2.3: The Video Sequencer is occupied warning dialogue.

Next, the user is prompted to choose the video reference file through a file selection window. File filtering is configured to show only directories and movie files accepted by the Video Sequencer. However, the user is free to disable the filtering, therefore it is necessary to validate the selected file extension afterwards.

After the user selects a video file, the add-on prepares the preview for the video reference. The 3D Viewport is guaranteed to be present in the workspace as the UI for importing the reference video is in the sidebar of the 3D Viewport. Utilizing this fact, the 3D Viewport area is split into two parts and the new area is changed to the Preview view of the Video Sequencer.

2. REFERENCE ADD-ON IMPLEMENTATION

The add-on then places the reference video file into the first Channel of the Sequencer with the settings `adjust_playback_rate=True` and `use_framerate=False`. By doing so, Blender artificially changes the total number of frames of the video to keep its original duration when being played at the rate of the Scene's fps, therefore synchronizing it with the animation playback. The reference video placed in the Sequencer is shown to the user through the previously set-up Preview as shown in Figure 2.4

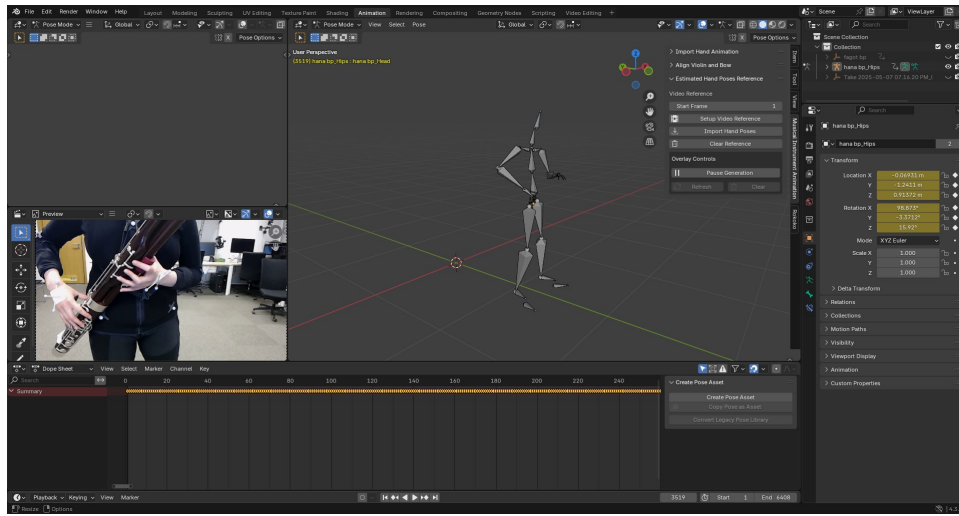


Figure 2.4: The video reference displayed through the Preview.

2.2 Estimated Hand Poses Overlay

The estimated hand poses overlay is activated after the user imports a JSON file with the estimated hand poses data. As mentioned in the previous Section 2.1, the overlay is created gradually. The add-on reacts to the changes in the playback's current frame and generates the overlay on-the-fly instead of computing it all at once. This approach prevents a lengthy wait time that could occur while generating an entire overlay for a longer reference video and improves the add-on responsiveness and smoothness of user experience.

2.2.1 Accessing the Estimated Hand Poses Data

After the user has imported a video as the reference, the *Import Estimated Hand Poses* button is enabled. Similarly to importing the video reference, a file selection window is opened which shows only directories and JSON files by default. A valid estimated hand poses JSON file is the output of the Hand Capture recording. Although it seems obvious, it should be noted that for the add-on to be useful, the user has to use the same JSON file for the creation of hand bones of the armature as for generating the estimated hand poses overlay.

When the user imports the JSON file, the add-on fully reads its contents and deserializes some of the data. It is necessary to use the same hand pose data for generating the overlay as well as building the hands of an armature when provided the same JSON file. Therefore, the add-on mirrors the behaviour of the Musical Instrument Animation add-on by Slezáček and saves up to one hand of each type and ignores the rest of the hands in the file [1, p. 29]. For each selected hand, a list of hand poses is saved which, as class attributes, hold their type—"LEFT" or "RIGHT", a timestamp and the normalized position of each hand landmark. The world positions get omitted as the add-on has no use for them.

After a change in the playback's current frame occurs, the add-on converts the current frame number to seconds by dividing it by the Scene's fps. With the newly computed timestamp, the Scene's current frame can be compared with the timestamps of the estimated hand poses. The add-on then uses binary search on each of the estimated hand poses lists to find the closest hand pose timestamp to the left of the Scene's calculated timestamp. If the found hand pose has not had its overlay image generated yet, the add-on draws the image and temporarily saves it to the `temp/hand_poses` folder.

2.2.2 Generating the Overlay Images

Generating the overlay images can be approached in various ways but reasonable solutions require usage of an external library. In fact, the Mediapipe Python package has its own function implemented for

drawing hand pose images⁴. Therefore, my first instinct was to use this already prepared function created specifically for the estimated hand poses data. However, there I encountered an unexpected roadblock—importing external libraries to Blender.

Blender uses its own Python installation, which means that even if the user has installed the required libraries on their system's Python, it will yield no effect on the Blender's Python installation. Furthermore, even though there are workarounds suggested in the Blender Python API documentation, they involve tinkering with the Blender files, which is not very user friendly [17]. Instead, I wanted the user to be able to install the necessary packages with a single pip command.

When trying to install external libraries directly to the Blender's Python installation with pip, I was met with a permission error, which can be defeated by running the installation as an administrator. However, that solution would make the add-on inaccessible to users without administrator rights. As an alternative, the libraries can be installed into the user site-packages⁵, as they are added to the `sys.path` within the add-on code.

With this approach I was able to successfully import the Pillow⁶ package, which I used for saving the hand pose images drawn by the Mediapipe package's `draw_landmarks` function. Contrarily, importing the Mediapipe package unfortunately resulted in a DLL load failure.

As a remedy I tried running the script for generating hands pose images in a subprocess with the system's python, where installing and importing the required packages should not pose any problems. This approach successfully generated the images; nevertheless, it was unfortunately highly inefficient as a new process was created each time a new image had to be generated. More specifically, when measuring the run time on my machine, the script generally ran around 0.02 seconds while the whole subprocess' run time was around 1.5 seconds. Waiting such a long time for each hand pose image is unacceptable as

4. The source code is available at: https://github.com/google-ai-edge/mediapipe/blob/f5c28d76f5d0da51a4452954a476b39e536d2b9c/mediapipe/python/solutions/drawing_utils.py#L119.

5. For information on user site-packages, see: <https://peps.python.org/pep-0370/>.

6. Pillow is a maintained fork of the Python Imaging Library (PIL). For more information, see: <https://pillow.readthedocs.io>.

they get generated on-the-fly during the playback. However, it should be noted that running the image generation in a separate process itself is not a bad idea if the process would be a parallel one as further explored in Section 4.2.

Finally, I decided to backtrack and rewrite the hand pose image generation to purely use the Pillow external library, which I was able to import without issues. Retrospectively, this approach is advantageous as it reduces the overhead of the add-on as Mediapipe is a large package with multiple dependencies⁷, which contains an entire suite of AI focused libraries [18]. On the contrary, the Pillow library is better suited for the task and allows for additional visual customization of the hand pose overlay. As I was already using it for saving the image into the PNG format with a transparent background, the external dependencies were reduced to the minimum. These facts heavily outweighed the disadvantage of having to code the drawing of the hand pose overlay images by hand instead of using the Mediapipe’s built-in `draw_landmarks` function.

The algorithm for drawing the hand pose overlay images works in the following way: the hand landmarks are organized into six groups – one for each finger, plus a wrist group. The groups are assigned a colour and a list of connections, which contain the landmark indices in the order of the connecting line’s path. To calculate the coordinates of each landmark, the algorithm fetches their normalized x and y positions and multiplies them by the image’s width and height, respectively. The algorithm first draws the connecting lines of each group with their assigned colour, dimmed by a factor of 0.6. Next, it draws circles filled with the original colour at the landmark positions themselves. Finally, the image is saved in PNG format with a transparent background.

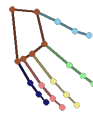


Figure 2.5: An example of a generated hand pose image.

7. See the `requirements.txt` file of the Mediapipe package: <https://github.com/google-ai-edge/mediapipe/blob/master/requirements.txt>.

2.2.3 Placing the Overlay Images into the Reference Video

After creating the hand pose image, the add-on places it into the Sequencer as an image strip. However, it first needs to solve the issue of where to place it on the timeline.

In the add-on, there are two hand-pose-to-timestamp alignment options provided, which decide how the range of each hand pose image strip on the timeline is calculated. My initial approach, *Edge Alignment*, see Figure 2.6, is straightforward and sets the image strip range in the following way: The start frame corresponds to the hand pose timestamp and the end is set to one frame before the starting frame of the next hand pose.

EDGE ALIGNMENT

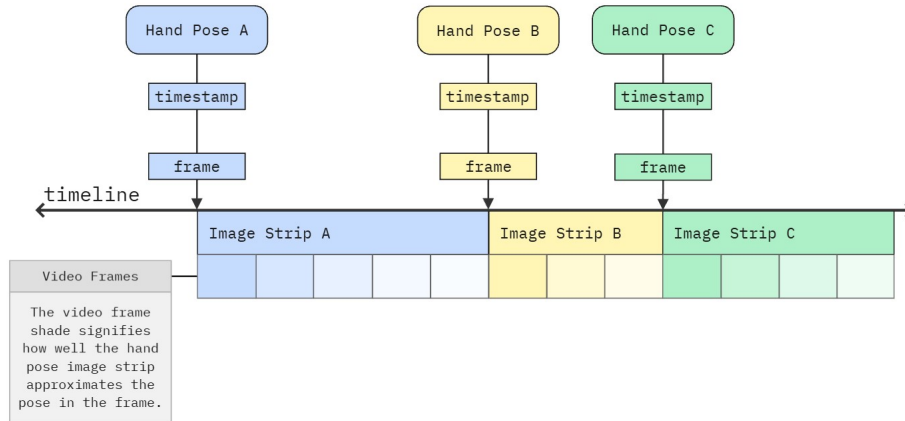


Figure 2.6: A diagram of *Edge Alignment*.

The advantage of *Edge Alignment* is that it accurately represents the exact timestamps of data in the JSON file. However, when the frequency of the estimated hand poses drops below the reference video fps, the drawback of this approach becomes apparent as the hand pose overlay appears to be lagging behind the reference video. Low frequency of hand poses can occur when capturing rapid movements. Nonetheless, it can also become an issue throughout the whole reference video when capturing the hand motion with a subpar camera or from an awkward angle—which is sometimes unavoidable when

capturing both hands at the same time depending on the instrument being captured⁸.

The second approach, *Centre Alignment*, see Figure 2.7, combats this disadvantage and distributes the hand pose image strips in a more natural manner. By doing so, the flow of hand pose image strips comes closer to the way the hand pose data is used to create the animation. More specifically, during the process of creating the hand animation, each hand pose is used to create a *keyframe*, which holds the information about the hand position. To create a smooth transition between each keyframe and animate frames with no hand position information, Blender interpolates the hand positions of nearby keyframes [19]. Therefore, the armature's hand position in a frame is affected not only by the hand pose before the frame but also by the following hand pose.

To mimic this behaviour, *Centre Alignment* sets the Image Strip range borders right in the middle of the spaces in between neighbouring timestamps. As a result, the lagging is significantly reduced. On the contrary, the overlay often moves slightly ahead of the reference video. However overall, the overlay seems to be more in sync with the video reference and the animation.

Here, a question arises as to whether an even smoother overlay could be achieved by interpolating the hand pose data. Although the interpolation would smooth out the transitions, deformations of the hand shape could occur. Furthermore, it would artificially create inaccuracies. For example, when quickly moving the hand in a arch, the interpolation could produce a position which in reality never occurred. As the goal of the add-on is to accurately visualize the hand pose data, this approach is inadequate since it would defeat the add-on's purpose.

To summarize, *Centre Alignment* produces a better flowing overlay, which reflects the usage of the hand pose data, whereas *Edge Alignment* shows raw hand pose data with their original timestamps, which is useful when debugging incorrect hand pose estimations.

The default option in the add-on is *Edge Alignment*. The user can toggle the alignment options in the Video Reference Settings panel at

8. As of now, Hand Capture cannot capture hands from multiple cameras yet.

CENTRE ALIGNMENT

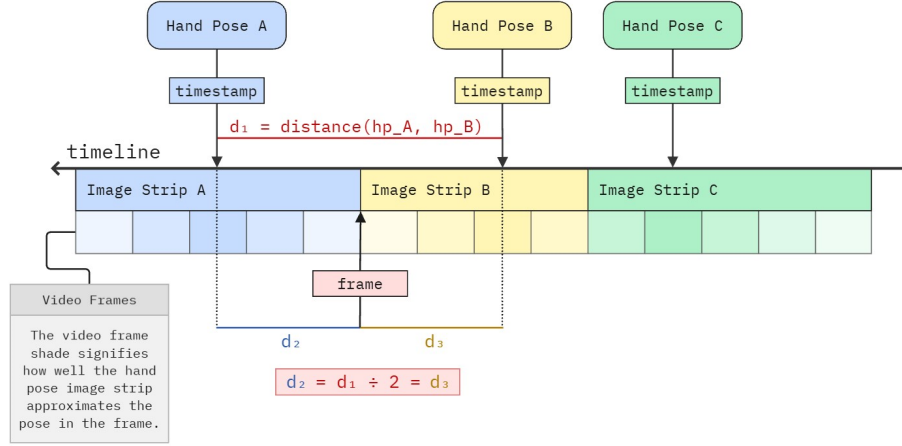


Figure 2.7: A diagram of *Centre Alignment*.

any time although it causes the overlay to refresh each time and delete the cache with already generated hand pose images.

No matter the chosen hand-pose-to-timestamp alignment, the hand pose image strips should create a continuous stream with one exception. When working with a JSON which contains incomplete hand data, lone hand poses with no neighbours close by on either side⁹ would get unreasonably prolonged. To prevent a display of a still, irrelevant hand pose image, the maximum length of a hand pose image strip is set to two seconds. More specifically in *Edge Alignment*, the strips are shortened from the right side to two seconds while in *Centre Alignment*, the strip is shortened to one second from each side, the centre being the timestamp, making up to a maximum length of two seconds in total.

This behaviour can be seen in Figure 2.8. The left hand has a sufficient amount of data, so it forms a continuous stream of image strips in *Channel 3*. The right hand contains data solely in one area of the timeline, so the outermost hand pose image strips in *Channel 2* get

9. When using *Edge Alignment*, only the distance to the following hand pose timestamp matters.

2. REFERENCE ADD-ON IMPLEMENTATION

shortened. The complete reference video with the estimated hand pose overlay is depicted in Figure 2.9.

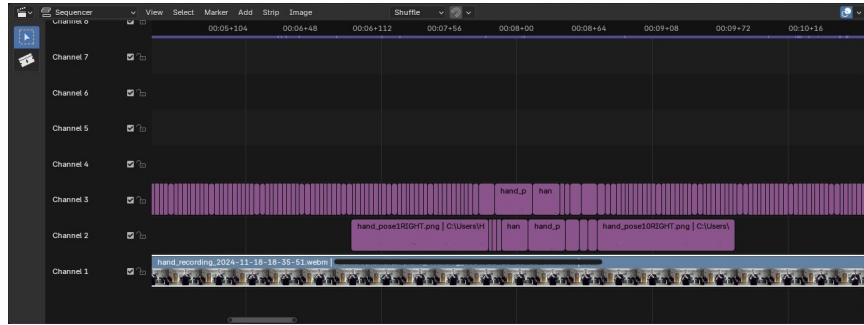


Figure 2.8: A screenshot of a Sequencer with fully loaded hand pose image strips in *Centre Alignment* mode.

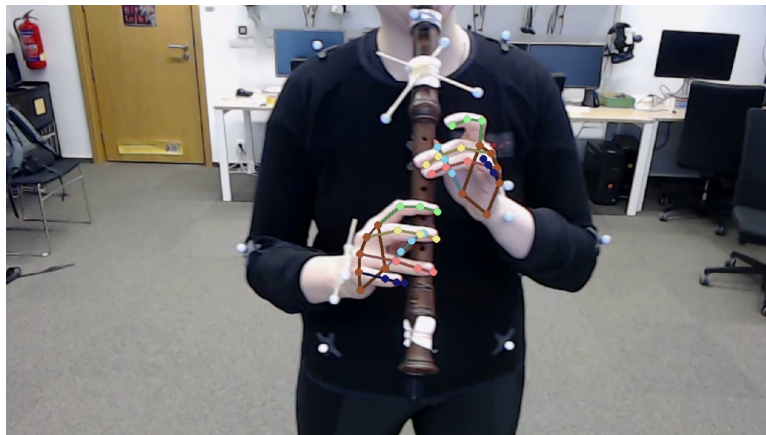


Figure 2.9: A screenshot of a video reference frame with the estimated hand pose overlay.

2.3 User Experience

The crucial part of the add-on's user experience is how much the reference affects playback speed. After experimenting with different

approaches and refining the algorithm, the add-on now performs sufficiently well; nevertheless, the playback is still noticeably slowed down.

The playback speed is mainly affected by two different factors: the overlay generation algorithm and the actual display of the reference in the Preview area. To fully understand the efficiency of these components, I utilized the *Hand Visibility* and *Pause Overlay Generation* controls available in the user interface, which are further described in Section 2.3.1. These options allowed configuring the add-on to perform these tasks separately, thereby enabling me to test their individual efficiency. Table 2.1 presents the rough estimates of the average playback performance through various configurations. The testing was done with the Scene frame rate set to 120 fps on two different machines, the first being a slightly outdated desktop with an *Intel Core i5-4570* CPU and 16 GB of RAM and the other being a Lenovo ThinkPad laptop with *AMD Ryzen 5 PRO 7540U* CPU and 16 GB of RAM.

Table 2.1: The results of testing the add-on’s impact on playback performance.

add-on configuration	1st PC	2nd PC
video reference without the overlay	45 fps	95 fps
overlay generation enabled overlay visibility disabled	36 fps	78 fps
overlay generation disabled while displaying pre-generated overlay	18 fps	44 fps
overlay generation enabled overlay visibility enabled	14 fps	38 fps

The results show that the difference of the playback slowdown between generating and not generating the overlay is small compared to whether the overlay is displayed. Therefore, the playback is slowed down mostly by the display of the overlaid video reference rather than the image generation. In fact, the display of the video itself has significant impact on the playback speed—even without the overlay.

Although the overlay generation partially slows down the Scene’s playback, once a part of the overlay has been generated, its impact on playback speed when replaying that section is minimal.

The current version of the add-on partially alleviates playback slowdown by offering options that temporarily increase playback speed when the overlay is not needed, in the same way as they were used during the performance testing. The thesis also proposes more thorough solutions in Chapter 4—Future Work.

The additional settings and overlay controls provided by the add-on are further described in the following section, which presents the user interface and explains the available tools aimed at enhancing the user experience.

2.3.1 User Interface

The user interface (UI) of the add-on is split into two panels. The first one, named *Estimated Hand Poses Reference*, is located right below the UI of the original add-on¹⁰ by Slezáček in the sidebar of the 3D Viewport in the *Musical Instrument Animation* category. This panel, see Figure 2.10, contains the general controls for operating the add-on, like importing the reference video and the estimated hand poses JSON file.

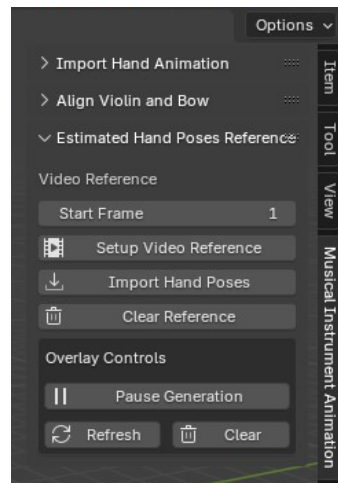


Figure 2.10: The *Estimated Hand Poses Reference* panel.

10. Only if the original Musical Instrument Animation add-on is also installed.

Additionally, it gives the user the option to offset the reference *start frame*, allowing it to be placed at a different point in the timeline, as well as a simple way to discard the reference, once it is no longer needed, using the *Clear Reference* button.

An important part of the panel are *Overlay Controls* which allow operating the overlay separately from the reference video. The highlight of the *Overlay Controls* is the *Pause Generation* button which allows the user to play through the armature animation without generating the overlay, even if the hand poses were already imported.

The second panel can be found in the *Video Reference* category in the sidebar of the Preview where the reference is shown to the user. As it hosts various settings regarding the hand pose overlay, it is named *Overlay Settings*. The full panel is shown in Figure 2.11.

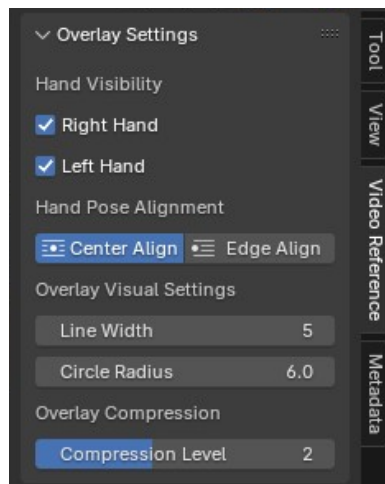


Figure 2.11: The *Overlay Settings* panel.

The *Hand Visibility* settings toggle the display of each hand individually. Aside from the straightforward usage of increasing clarity—such as displaying only one hand when the two hands are overlapping, or temporarily disabling both to better see the video underneath—they can also be used to increase the playback speed. As discussed previously in Section 2.3, displaying the overlaid video negatively affects the playback speed. By disabling the hands’ visibility when they are not

needed, the user can achieve higher playback speed without clearing the entire hand pose overlay.

Depending on the distance of the musician from the camera the user might want to adjust the thickness of the overlay. They can do so by adjusting the *Line Width* and the *Circle Radius*.

The last settings found in the second panel are the *Hand Pose Alignment* and the overlay images *Compression Level*. The *Hand Pose Alignment* allows the user to choose the method of placing the hand pose overlay images on the timeline as previously explained in Section 2.2.3. The *Compression level* setting was implemented to give the user the option of prioritizing either storage efficiency through higher compression or faster image creation with lower or no compression. In practice, the effect on playback speed is minimal; however, the option remains available in the add-on, with the default setting being level two as a reasonable middle ground.

3 Motion Capture Exemplary Dataset

As a part of the thesis, a MoCap dataset was created to demonstrate the capabilities of the add-on as well as to provide testing data for the development. I also wanted to experiment with capturing performances of musicians playing different instruments to further explore the usability of the workflow developed by Slezáček as it has not been used yet for recording other instruments besides the violin. The final dataset consists of the following recordings:

- A recorder performance of *Tarantella Napoletana* by Luigi Ricci
- A guitar performance of *Adelita* by Francisco Tárrega
- A bassoon performance of the first movement of the *Bassoon Concerto in B-flat major* by W. A. Mozart

3.1 Motion Capture Recording Process

The process of recording the MoCap dataset was generally the same as described in Section 1.2.1 with the exception of attaching the markers to the different instruments and finding the optimal camera position for each recording. The differences and specifics of the recording process are described in the following sections.

3.1.1 Hand Capture Configuration

Before recording, thorough preparation is necessary to achieve clean results. Aside from the usual preparation for motion capture, the Hand Capture must also be configured. Several factors can significantly affect the quality of Hand Capture results and should be considered when setting up the recording:

- **Web camera quality.**
 - Low **resolution** and blurry low-quality video impairs the AI model accuracy as it has less data available to clearly estimate the hand positions.

- If the camera supports a higher **frame rate**, the frequency of estimated hand poses in the resulting JSON will also be higher.
- It should be noted that the frame rate and resolution must be adequately set up in the Hand Capture settings to take advantage of a good quality camera.

- **Web camera position.**

- **Web camera angle.** The optimal angle depends on the instrument being captured. Sometimes it is necessary to find a compromise between a clear view of most of the fingers and avoiding a complete occlusion of some of the fingers.
- **Web camera distance.** Generally, it is best to place the camera as close as possible while being far enough to allow the musician to move freely without drifting their hands out of the frame. By doing so, the resolution of the camera is optimally utilized.

The camera used for the dataset recording was Logitech C920 set to 1280x720 resolution and 60 fps. It was fixed to a tripod for stabilization and for shooting the video from an optimal position.

The camera was angled at each instrument differently. The recorder was shot slightly from above. I had concerns with the partial occlusion of thumbs but the AI model fared quite well for the most part.

When recording the guitar, the camera was positioned right in front of the instrument. The angle provided a nice straight view of all fingers except of the left thumb. Unfortunately, the model did not manage to estimate the thumb positions well and for a polished animation, it is necessary to adjust them manually. However, as the thumb does not play the notes actively, there is a lower need for preciseness of its position.

The bassoon proved to be the most difficult instrument to record as there needs to be a compromise between a clear view of all the fingers in front, while providing at least some view of the thumb positions. Contrary to the guitar's left thumb case, when playing the bassoon, the thumbs have to operate the largest number of keys out

of all the fingers—see Figure 3.1, so their positions change drastically throughout the performance.

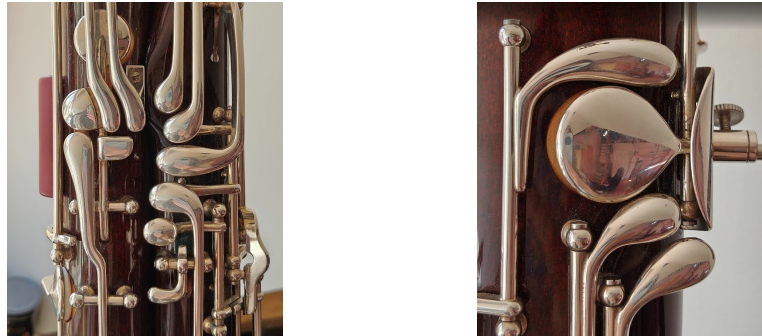


Figure 3.1: The bassoon thumb keys.

When searching for the optimal angle, the biggest drawback was that the AI model has trouble recognizing hands which have a large portion—specifically the palm—obscured, even if the fingers are visible. This behaviour can be seen in Figure 3.2, where the Hand Capture application fails to recognize the left hand at all. As a result, I believe that the optimal angle does not exist and two cameras would be necessary to fully capture both hands.



Figure 3.2: A screenshot from the Hand Capture web application.

Ultimately, I pointed the camera more at the right side of the player which resulted in a clear view of the fingers in front and a better approximation of the right thumb. Unfortunately, the left thumb positions were estimated poorly. Rather than visibly moving to the different keys, the left thumb mostly stayed in a single position.

3.1.2 Attachment of Markers to the Instruments

The attachment of markers to the violin and bow is very specific to assist the automatic alignment of their 3D models in the Blender add-on. However, as there is no automatic alignment for other instruments available, the alignment must be done manually.

This granted me more freedom with the marker attachments. Nonetheless, to make the the manual alignment of the 3D models easier, I placed some of the markers into clearly distinguishable places when possible, and took photos of the marker placements on each instrument to serve as a reference.

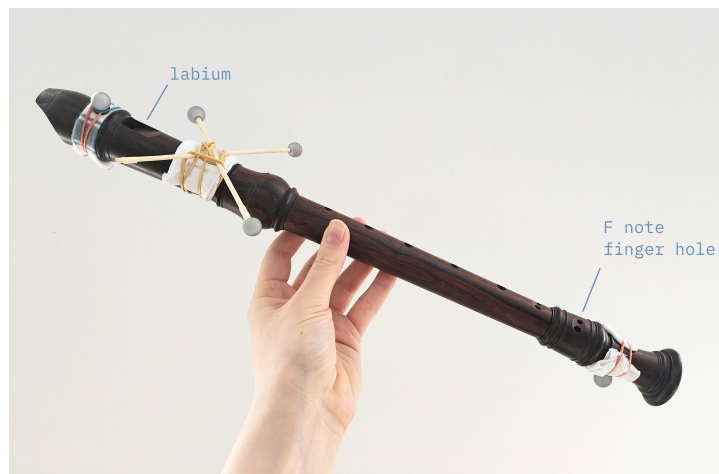


Figure 3.3: Marker placements on the recorder.

For example, when attaching markers to the recorder, I placed one marker directly above the left side of the labium and another one below the F note finger hole rotated to the centre back of the recorder. The rest of the markers were placed further away from the body of the recorder to better capture its rotation around the axis defined by

the two markers attached near the ends of the recorder. The complete placement of the markers can be seen in Figure 3.3.

For placing the markers onto the recorder and bassoon, I used rubber bands in the same manner as Slezáček did for some of his marker attachments [1, p. 14]. The rubber bands provided enough friction for the cylindrical shape of both instruments, even after placing fabric directly underneath the harsh surface of the markers for scratch protection. The markers stayed securely in place and I had Slezáček's original marker set available, so I could reuse it and save time on making new attachment contraptions.

With this method I only had to be careful with safely navigating the rubber bands around the keys of the bassoon without impairing their movement. When placing the bassoon marker attachments, see Figure 3.4, I also added one more marker, compared to the rest of the instrument's marker attachments, on the bocal, as it is a movable part which each bassoonist has configured slightly differently.

Attaching the markers onto the guitar was more problematic. The rubber bands could be used for attaching markers onto the headstock, however for the large body of the guitar I had to use a different method. Inspired by the attachment of the leg support used by the guitarist, I used a rubber suction cup for attaching one of the markers.

For the last marker I experimented with one more method. I attached painter's tape, which can be easily removed without straining the surface underneath, to the guitar. The painter's tape then serves as a protection layer to which the marker can be attached with a harsher method, like a double-sided tape. Although this method worked well and did not harm the surface of the guitar whatsoever, I would not recommend using it on very expensive instruments or those that do not have a solid layer of varnish. The Figure 3.4 shows the complete marker attachment for the guitar.

3.2 Motion Capture Results

To explore the recorded dataset, I imported each FBX file—containing the full-body MoCap exported from *Motive* after post-processing—into Blender. Then, I used Slezáček's add-on to import the hand data and set up the reference together with the hand pose overlay. Utilizing

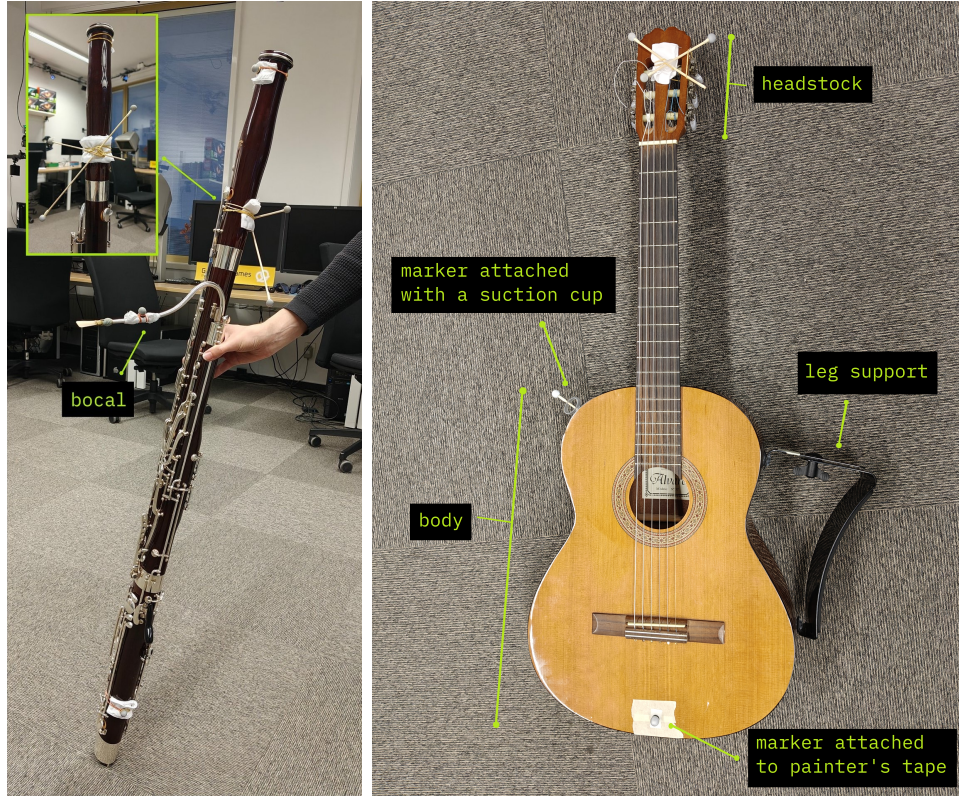


Figure 3.4: The marker placements on the bassoon and the guitar.

the hand pose reference add-on developed as the main part of this thesis, I was able to quickly assess the quality and deficiencies of the resulting armature animation.

When exploring the recorded dataset, I was focused on appraising two aspects: how accurately the Hand Capture estimated the hand poses and whether the hand animation was constructed correctly after importing the hand poses of the non-violin performance using Slezáček's add-on.

The hand pose estimation aspect mainly indicates if the Hand Capture was configured correctly and how well the AI model handled difficult or occluded hand positions. The second aspect tests the add-on's current capability to process hand pose data of performances of other instruments besides the violin.

More specifically, it explores the add-on's limitation as described by Slezáček in his thesis: *"Another aspect requiring refinement is the algorithm for transforming world space joint positions into local space. While the current algorithm works well for violin playing, it fails when the angle between two successive finger bones approaches 180°, resulting in incorrect twisting of bones around the y-axis (the y-axis points in the direction of the successor joint)"* [1, p. 39].

The capture of the recorder performance yielded the cleanest results. As the hands are positioned close to one another, they could be recorded from a short distance which increased the hand pose estimation accuracy. There were also minimal problems with occlusion, as the fingers remained mostly visible—except for the partially occluded left thumb, which was estimated with slightly less accuracy. The anticipated twisting of bones happened at several points but only as a short occurrence with a single finger, as the hands were generally positioned in a curved line that avoided the problematic 180° angle.

The guitar performance recording produced slightly worse results. The right hand ran into the bone twisting issue more frequently, and with more severe consequences—the entire hand would twist, rather than just a single finger. In contrast, the right hand's pose estimation appeared mostly clean.

The left hand did not run into any major bone twisting issues; however, the quality of hand pose estimation was worse. Aside from the left thumb issue which was already discussed in Section 3.1.1, the AI model also produced some incorrect hand poses during some of the musician's fast hand pose transitions or when playing vibrato. I believe this occurred because the camera was not positioned close enough.

Nevertheless, the recording served as a valuable source of data for testing the reference add-on. The reference helped quickly identify the corrupted keyframes and improve the animation by deleting or adjusting them. The add-on was especially useful for clarifying whether a convoluted hand position in the armature was due to the incorrect estimation of the hand pose or caused by the hand import algorithm.

For example, the Figure 3.5 shows an armature with a visibly incorrectly animated right hand. The visualization of the hand data provided by the reference add-on indicates that the error occurred when importing the hands and not during the hand pose estimation.

3. MOTION CAPTURE EXEMPLARY DATASET

Similarly, in Figure 3.6, it can be deduced that the hand bone twisting on the armature was due to the incorrect hand pose estimation rather than the limitation of the original add-on.

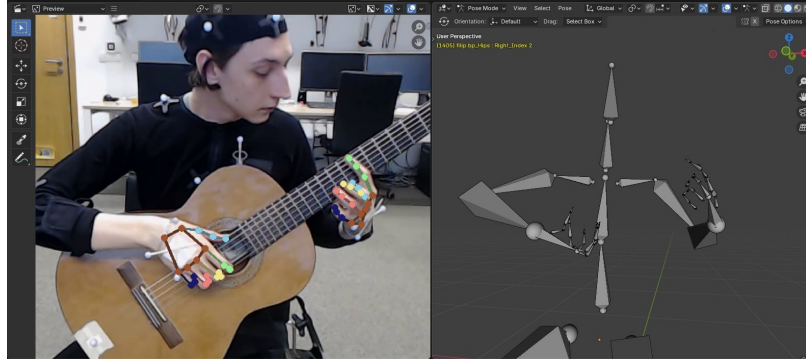


Figure 3.5: An example of bone twisting caused by an incorrectly imported hand pose.

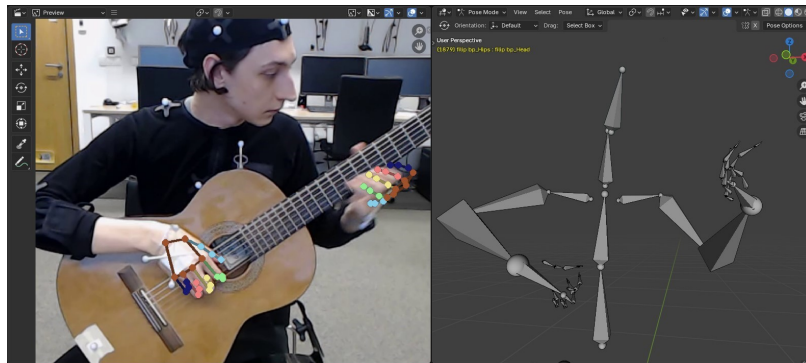


Figure 3.6: An example of bone twisting resulting from an incorrectly estimated hand pose.

The armature animation of the bassoon performance failed to produce satisfactory results. Although the hand poses were estimated surprisingly well—excluding the left thumb, as discussed in Section 3.1.1—the algorithm for constructing the hand animation failed to correctly animate the hand positions. Since the finger positions tend to be quite straight when playing the bassoon, the bones of the left hand were almost constantly twisted, while the right hand often rotated

into a noticeably less twisted position. The current workflow is unable to create an accurate animation of the bassoon performance; however, the recording can be utilized as testing data in future algorithm development focused on preventing bone twisting.

4 Future Work

Although the add-on significantly enhances the process of polishing the armature animation, one area of possible improvement is its efficiency. The following sections explore the potential enhancements of two different areas: the image generation and the reference display.

4.1 In-Scene Reference Video

The difference of the playback slowdown between generating and not generating the overlay is small in comparison to the Preview area being opened and closed. Therefore, the playback is slowed down mostly by the display of the overlaid video reference, rather than the image generation. In fact, the display of the video itself has significant impact on the playback speed, even without the overlay.

A feature that would help mitigate this issue could be an operator that generates all of the necessary hand pose images, exports the overlaid reference video and adds it to the Scene as a reference. This solution is not responsive in the slightest and does not adhere to the requirements set by the thesis description. In fact, this process could take several minutes, depending on the length of the recording. However, if the user prefers to sacrifice this time in order to have a smoother experience with the faster playback later on, it would be highly beneficial to them. After experimenting with this approach by manually replicating all the steps, it seems that the exported in-Scene reference has minimal impact on the playback speed which is a tremendous improvement.

Apart from the improved playback speed, another advantage of this solution is that it combines the two approaches discussed in Section 2.1. The benefit is that the in-Scene reference allows the user to float it around within the Scene, thereby placing it exactly where they desire. At the same time, if the user only wants a quick access to the reference and waiting for the rendering of the in-Scene reference would not be advantageous, the original Video Sequencer implementation would still be available to them.

4.2 Parallel Hand Pose Image Generation

The hand pose overlay generation runs at a high speed; nonetheless, additional improvement is possible. By running the image generation in a separate thread, the image generation could be further optimized. Currently, the image generation is triggered by a change in the current frame. As a consequence, when the playback is idle, no hand pose images are generated. Parallel image generation would enable utilizing this idle time to create the following hand pose images in advance without occupying the main thread.

This approach would inevitably introduce several challenges. Firstly, it is necessary to ensure the playback waits until the image generation for the following frame is complete, in order to prevent it from "outrunning" the image generation. Next, although it is favourable to gradually create overlay images for upcoming frames, the add-on still needs to react quickly to abrupt changes in the current frame, such as when the user manually jumps to a random point on the timeline, rather than playing through it sequentially. It is also necessary to account for the fact that in Blender the playback can be played in reverse, therefore it is advisable to compute not only the following frames but also the preceding ones in advance.

Due to these reasons, the order of generating the overlay images is convoluted. The solution to this issue could be implemented using a queue or perhaps even a priority queue. The main thread would request the generation overlay images for the neighbouring hand poses by adding them to the queue. The manual change of the current frame could be handled by clearing the queue and rebuilding it from the neighbours of the new current frame.

Conclusion

This thesis aimed to develop a Blender add-on that enhances the existing *Musical Instrument Animation* add-on by displaying a reference video overlaid with estimated hand poses. An additional objective was to record an exemplary motion capture dataset to demonstrate the usability of the add-on.

I explored the current state of the previously developed musical performance motion capture workflow and its underlying technologies, before proceeding to the reference add-on implementation. The development process involved overcoming several challenges: finding a method to display the reference video while enabling iterative construction of the hand pose overlay, increasing the efficiency of hand pose image generation, and developing methods of placing the hand pose image onto the reference video. A considerable amount of effort was also put into improving the add-on's usability and performance by providing the user with a variety of overlay options and controls in the UI. Finally, I recorded the exemplary motion capture dataset and evaluated the results using the newly developed reference add-on.

The usefulness of the add-on was showcased well with the use of the recorded dataset, as the add-on allowed to distinguish the defects of the hand pose estimation from errors which stemmed from the limitations of the original *Musical Instrument Animation* add-on. In turn, the reference add-on facilitated the evaluation of the recorded dataset's quality.

Although the performance of the reference add-on could be further improved, this thesis succeeded in making the Blender add-on more effective and user-friendly by providing the animator with visual feedback on the accuracy of the estimated hand poses and a reference to the actual desired hand positions. Furthermore, the created motion capture dataset—aside from serving as a useful source of testing data for the add-on—allowed for testing the *Violin Performance Motion Capture* system's capabilities for recording a variety of musical performances. The add-on, together with the exemplary dataset, can significantly aid future development of the original *Musical Instrument Animation* Blender add-on, as the reference serves as a powerful debugging tool and the dataset provides valuable testing data.

A Electronic Attachments

The following list presents the electronic attachments of the thesis:

- MusicalInstrumentAnimationReference.zip
The *Musical Instrument Animation Reference* Blender add-on.
- ExemplaryMoCapDataset.zip
The exemplary motion capture dataset.

Bibliography

1. SLEZÁČEK, Jakub. *Violin Performance Motion Capture*. Brno, 2024. Available also from: <https://is.muni.cz/th/p6qrp/>. Master's thesis. Masaryk University, Faculty of Informatics. Supervised by Jiří CHMELÍK.
2. TONG, Fei. Human motion and musical expression: Exploring the link between physical movement and emotional resonance in performance. *Molecular & Cellular Biomechanics*. 2024, vol. 21, no. 2, p. 401. Available from DOI: 10.62617/mcb.v21i2.401.
3. *Active Finger Marker Set – EXTERNAL OptiTrack Documentation* [online]. OptiTrack [visited on 2025-05-21]. Available from: <https://docs.optitrack.com/markersets/hand-and-fingers/active-finger-markerset>.
4. *Cracking Knuckles, Metacarpals and Phalanges* [online]. Vicon [visited on 2025-05-21]. Available from: <https://www.vicon.com/resources/case-studies/cracking-knuckles-metacarpals-and-phalanges/>.
5. *Smartgloves - affordable quality finger and hand motion capture* [online]. Rokoko [visited on 2025-05-21]. Available from: <https://www.rokoko.com/products/smartgloves>.
6. BREGLER, Chris. Motion Capture Technology for Entertainment [In the Spotlight]. *IEEE Signal Processing Magazine*. 2007, vol. 24, no. 6, pp. 160–158. Available from DOI: 10.1109/MSP.2007.906023.
7. MENACHE, Alberto. *Understanding Motion Capture for Computer Animation*. 2nd ed. Morgan Kaufmann, 2011. The Morgan Kaufmann Series in Computer Graphics. ISBN 9780123814975.
8. *About OptiTrack* [online]. OptiTrack [visited on 2024-11-02]. Available from: <https://optitrack.com/about/>.
9. *Motive* [online]. OptiTrack [visited on 2024-11-02]. Available from: <https://optitrack.com/software/motive/>.

10. *Assets – EXTERNAL OptiTrack Documentation* [online]. OptiTrack. Version v2.3 [visited on 2024-11-02]. Available from: <https://docs.optitrack.com/v2.3/motive/assets>.
11. *Rigid Body Tracking – EXTERNAL OptiTrack Documentation* [online]. OptiTrack. Version v2.3 [visited on 2024-11-02]. Available from: <https://docs.optitrack.com/v2.3/motive/rigid-body-tracking>.
12. *Skeleton Tracking – EXTERNAL OptiTrack Documentation* [online]. OptiTrack. Version v2.3 [visited on 2024-12-04]. Available from: <https://docs.optitrack.com/v2.3/motive/skeleton-tracking>.
13. *About – blender.org* [online]. Blender Foundation [visited on 2024-11-02]. Available from: <https://www.blender.org/about/>.
14. *Features – blender.org* [online]. Blender Foundation [visited on 2024-11-02]. Available from: <https://www.blender.org/features/>.
15. *NormalizedLandmark* [online]. Google [visited on 2025-05-13]. Available from: <https://ai.google.dev/edge/api/mediapipe/java/com/google/mediapipe/tasks/components/containers/NormalizedLandmark>.
16. *Hand landmarks detection guide* [online]. Google [visited on 2025-05-13]. Available from: https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker.
17. *Bundled Python & Extensions – Tips & Tricks – Blender Python API* [online]. Blender Foundation [visited on 2025-04-11]. Available from: https://docs.blender.org/api/4.3/info_tips_and_tricks.html#bundled-python-extensions.
18. *MediaPipe Solutions guide* [online]. Google [visited on 2025-05-13]. Available from: <https://ai.google.dev/edge/mediapipe/solutions/guide>.
19. *Keyframes – Introduction – Blender Manual* [online]. Blender Foundation [visited on 2025-04-03]. Available from: <https://docs.blender.org/manual/en/4.3/animation/keyframes/introduction.html>.