Faculty of Electrical Engineering

Department of Radio Engineering

Motion Data Interpolation in Sign Language Synthesis

Diploma Thesis

Cherepanov Alexander

Master's degree study program: Electronics and Communication

Specialization: Audiovisual Technology and Signal Processing

Supervisor: Ing. Martin Bernas CSc.

Prague, 2021

Fakulta elektrotechnická

Katedra radioelektroniky

Interpolace pohybových dat při syntéze znakového jazyka

Diplomová práce

Cherepanov Alexander

Magisterský studijní program: Elektronika a Komunikace

Studijní obor: Audiovizuální Technika a Zpracování Signálů

Vedoucí práce: Ing. Martin Bernas CSc.

Praha, 2021

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Cherepanov Alexander** Personal ID number: **397830**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Radioelectronics**

Study program: **Electronics and Communications**

Specialisation: **Audiovisual and Signal Processing**

II. Master's thesis details

Master's thesis title in English:

Motion Data Interpolation in Sign Language Synthesis

Master's thesis title in Czech:

Interpolace pohybových dat při syntéze znakového jazyka

Guidelines:

Use recordings of a speaker of Sign Language made with optical motion capture system.
At first process the whole sentences and then process separately the individual signs of the same sentences. Find a suitable method of concatenating individual signs to get the whole sentence. Apply processed motion data of the speaker of Sign Language on the human body model and analyse the differences between original and combined sentences.

Bibliography / sources:

[1] Bloem, R.: Handbook of Model Checking. Springer-Verlag, 2018, ISBN: 3319105744
[2] Nirme, J.; Haake,M; Gulz,A; Motion capture-based animated characters for the study of speech–gesture integration. Behav Res 52, 1339–1354, 2020. https://doi.org/10.3758/s13428-019-01319-w

Name and workplace of master's thesis supervisor:

Ing. Martin Bernas, CSc., Department of Radioelectronics, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **29.01.2021** Deadline for master's thesis submission: **04.01.2022**

Assignment valid until: **30.09.2022**

Ing. Martin Bernas, CSc.	doc. Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

Date of assignment receipt

Student's signature

Acknowledgment / Affidavit

I would like to thank Mgr. Lenka Okrouhlíková, Ph.D., of Institute of Deaf Studies FF UK for the help with a list of basic signs to record, as well as hew tremendous help support as a translator during the recording session. Also, I would like to thank Mgr. Radim Krupička, Ph.D., of Faculty of Biomedical Engineering, CTU in Prague for an opportunity to use motion capture system. Especially I would like to thank my supervisor Ing. Martin Bernas CSc. For his helpful advice and patience.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Poděkování / Prohlášení

Rád bych poděkoval Mgr. Lence Okrouhlíkové, Ph.D. z Ústavu jazyků a komunikace neslyšících FF UK za pomoc se seznamem základních znaků k nahrávání a za velkou pomoc jako překladatele při nahrávání. Dále bych chtěl poděkovat Mgr. Radimovi Krupičkovi, Ph.D. z Fakulty biomedicínského inženýrství ČVUT v Praze za možnost využít systém snímání pohybu. Zvláštně bych chtěl poděkovat svému vedoucímu Ing. Martinovi Bernasovi CSc. za jeho užitečné rady a trpělivost.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne . . . . . .

Abstract / Abstrakt

This project started as an idea to create an automated translator for a Sign language. In that work we cover basics behind sign language and computer-generated animation. Proposed idea is to record parts of the SL using motion capture system and find a way to connect them in a natural manner. Thesis aims to find simple, yet effective methods to connect these elements into continuous speech. As an out this project not only shows effectiveness of basic interpolation techniques, but also covers entire process of creating a database for a Sign Language and how to work on it later on.

Human Animation, Motion Capture, Sign Language Synthesis, Interpolation.

Tento projekt začal jako nápad vytvořit automatický překladač pro znakový jazyk. V této práci pokrýváme základy znakového jazyka a počítačově generované animace. Navrhovaná myšlenka je zaznamenat části ZJ pomocí systému zachycení pohybu a najít způsob, jak je spojit přirozeným způsobem. Práce si klade za cíl najít jednoduché, ale účinné metody, jak tyto prvky spojit do souvislé řeči. Výsledkem je, že tento projekt ukazuje nejen efektivitu základních interpolačních technik, ale také pokrývá celý proces tvorby databáze pro znakový jazyk a jak s ním později pracovat.

Lidská animace, Motion Capture, syntéza znakového jazyka, interpolace

Table of Contents / Obsah

Chapter I
Introduction

Short note on overall motivation for using and improving Motion Data Processing in Sign Language.

One of the most frequent questions regarding deaf people and why do they need an interpreter is "Why can't we just use subtitles or captioning?". It is important not to ignore the key word 'Language' in a phrase Sing Language. Spoken version appears to be a foreign language for people native in SL. According to the research done by Rose, S. - reading comprehension score for 17- and 18-year-old deaf students is equivalent to that of fourth grade hearing students. This happens due to the simple fact that it is virtually impossible to learn typical Spoken Language even in its written form without any audible feedback [1,22].

Therefore, it is essential to provide information for mute-deaf people in a natural for them way of communicating. Typical today's solution is to hire a translating interpreter (illustrated in *figure 1*).

Figure 1 Live news segment with Sing Language interpreter

But with a real sign language interpreter comes all sorts of challenges. Main of which being their availability. It is not realistic to expect every public space, every business, or ever every institution to have a person capable of translation at all times. This is a hard profession coming at a high price. And even when interpreter can be afforded, like on TV, for example, they are not error proof. Working with a fast speech its common to omit finer details.

Modern animation technology offers a cheap and fast synthesized solution for presenting information to people with hearing impairments. Of course, giving the fact that we solve a problem of connecting animated signs in efficient manner. Without this animation might come out as robotic and very mechanical and unpleasant for the Signer.

This work is a small part of a much bigger project with a goal of creating a human-like animation without being computationally demanding but still keeping decent level of intelligibility.

Withing this project I plan to take on basic methods of processing motion capture data and create a dataset of Sign Language for future student to experiment on as well as test further methods.

Chapter II
Theory

This thesis is separated intro three main parts. First part (*Chapter 2*) presents an overview of basics behind the Sign Language itself. Describes technology and software needed to create a dataset. Explores if it is possible to construct comprehensible animation from separate Signs using simple non-computation-power-demanding methods and outlines more sophisticated methods that might be useful. Second part (*Chapter 3*) diligently describes experimental part from harvesting motion capture data, processing it, interpolating signs and to applying onto a 3D model. Last part (*Chapter 4*) is presenting outcome, comparing the result, as well as discusses the topic of future steps for this project.

2.1. Sign Language

Deprived of a standard sound-generating apparatus, deaf-mutes use the complex structure of joints and muscles to reproduce one's thoughts.

To clear a common misconception, sign language is not just representation of a spoken language using hands, or letters – using fingers, however there is such a component as sign alphabet. But more on components further.

While spoken language is a linear form of delivering information, Sign Language is able to present many linguistic units simultaneously. Something that might seem like an advantage leads to inefficiency of live translation performed by a human yet might not present a challenge for a properly programmed 3D avatar.

2.1.1. Components

Sing language structure is complex and unusual of a common speaker. It can be comparable to a structure of a natural language of corresponding country, while still being independent in structure, syntax, and morphology.

Let us take a short introduction into different units and components of SL, each introducing additional meaning.

→ *Finger Alphabet*

Finger alphabet is not the same as SL, but simply is a representation of written language. However, it plays important part in communicating terms and expressions that does not have special assigned. This work is mostly oriented at classical spelling of a sign, ignoring finger alphabet. It is a different task that should be processed and programmed further into the project and incorporated in some manner.

→ *Manual Components*

These components carry the most information and therefore would be the main focus of this work.

- shape of a hand
- place of articulation
- hand to body relation/orientation
- mutual position of hands (for two-handed signs)
- hands movements

→ *Non-Manual Components*

- facial expressions
- upper body postures
- head motions
- mouth articulation

First role of these components is to convey additional information, such as mood, intonation, facial expressions.

While a professional artist can design and/or pre-program a beautiful animation containing all of these aspects, it is virtually impossible for a machine to add mood or to the simulated Sign Language in real time. Those parts of a Sign Language are capable of modifying meaning, but unfortunately that simply is not data that can be processed based on text or even speech alone.

Second role is to indicate a type of a sentence – negation/agreement or question.

These features are something like prosody in spoken languages. They mark phonological contrast, adverbial markers, attitude or affect, and they have a crucial role in syntactic functions - agreement, negation, WH-questions [22].

There is no denial that non-manual articulations are a fundamental component of all sign languages, but for this particular project, most important non-manual information is being generated with a mouth. One part of the function is so-called mouthing – imitation of Spoken Language components. Also, oral actions serve to distinguish pairs of signs, where the manual part is identical, and the only differences are oral [22].

Giving these facts, for the purpose of this work, it was decided to keep amount of Non-Manual Components to a bare minimum. Only rare head motions and mouth articulations are taken into consideration. It is explained in detail later in work when we talk about capture system.

2.1.2. *Fixed and Produced signs*

Over time any type of a Sign Language develops so called fixed signs or sing connection. Usually, this term refers to shorter phases that are in common use. Also, idioms, some comparisons, as well as whole proverbs and sayings.

Produced signs are characters composed ad hoc according to the situation of the user, composed of morphemes available in the language.

Signs do not always relate to some specific words and can be modified using special tools: constructors, classifiers, specifiers. These tools can cover everything from shape and color to body parts or animal species. All of that using different shapes, semantics, gradient, and sizes of movement.

At this stage our project works exclusively with fixed signs.

2.1.3. *Sign Concatenating*

It is virtually impossible to capture every version of every connection between Sings even trying to cover a single topic. Meaning shorted parts has to be recorded and then compiled into realistic sentences. Interpolations can be performed as working with particular coordinates, or computing angles between adjacent segments.

2.2. *Animation*

Animation itself is a very broad and complex topic. Here, however, we cover only necessary basics, to understand minimal requirements needed for out project. a. Sign language synthesis require animation of humans with emotion and personality, and requires knowledge about biomechanical and kinesthetic constraints [22].

2.2.1. *Human Body Modeling*

The human body can be structured in three levels: the skeleton, the net of the muscles and the enveloped surface of the skin [11]. For this particular project we are not going to explore that deep into modeling, and ignore muscle layer, leaving that for the later stages of development. Luckily for us most of the systems base their modelization on this "anatomic" structure and generate a geometric model, articulate in a similar way to the skeleton, a sculptured surface model similar to the skin or a mixed model dealing with both levels, skeleton, and skin [11]. Skin in this context means a character mesh, or Topological network of vertices representing the model of the figure.

Even though historically human 3D animation went through many stages over last 30-40 years:

→ *Stick Figure Models*

Simple and efficient models. Basic hierarchical system of rigid segments(limbs) connected at joints.

These models are called articulated bodies [11].

→ *Volume Models*

The volume models approximate the structure and the shape of the body with a collection of primitive volumes, such as cylinders, ellipsoids, or spheres.

Those models were developed at the early of human animation when graphic systems had very limited capabilities [11].

→ *Surface Models*

Surface models are conceptually simple, containing a skeleton surrounded by surfaces composed of planar or curved patches, simulating the skin [11].

Producing way better results than Volume models

→ *Multi-layered Models*

In this approach, normally a skeleton is used to support intermediate layers that simulate the body volume (bones, muscles, fat tissues and so on) and the skin layer. Sometimes a clothing layer can be also considered [11].

As mentioned just now, at this stage of development project doesn't require such a deep modeling technique as Multi-layered and would suffice with a *Surface model.*

Important to emphasize again that main focus for us right now is animation hands and face. In general modeling these parts is not going to differ from rest of the body, but I still would like to add some details.

2.2.2. *Hand modeling*

Hand (as well as face) animation is generally treated separately from the rest of the body. The motivation for a different deformation technique for the hands comes from their very specific behavior. The interior side of the hand is crisscrossed by lines and wrinkles that create discontinuities on the surface during deformation. The palm covers five skeleton segments and is very flexible. Finger deformations are very complex compared to other parts of the body, as they involve a wide range of angles' variations and configurations applied to very short skeleton segments [11].

However, having a properly rigged structure of bones that is compliant with basic rigid body features and kinematics is sufficient for applying motion capture data onto virtual avatar.

2.2.3. *Face modeling*

Despite all of the challenges of modeling a hand, *face modeling* proves to be far more complex. It involves all the problems concerned in the body modeling, but as its shape is especially irregular and the movements associated to it are rather complex, its difficulties are greater. The problem is further compounded with its interior details such as muscles, bones and tissues, and the motion which involves complex interactions and deformations of different facial features [11]. At this point, properties of facial expressions are being studied for almost 50 years, and classical 3D animation studios got close (example any Disney/Pixar movie released in a last decade) yet there is still no definitive answer.

Even for a typical spoken language it is said that human face plays most important role for identification and communication. It is even more true for a Sign Language giving reasons described in 2.2.1. Components

2.2.4. *Hierarchy*

Once a body model has been defined, it can be manipulated, animated, and used for simulation purposes, meaning body parts are being assigned with a motion.

Motion is a change of position of an object with respect to a reference. This treatment is split into two fields.

- *kinematics* – geometry of motion regardless of its physical realization
- *dynamics* – Newton's law based motions

Working with motion capture though, dynamics are not taking into consideration and defaultly given by an actor's performance.

But set of segments compliant to kinematic/dynamic laws by itself is not enough to create a body model. An articulated structure is a set of rigid bodies (or segments) connected by joints is needed. An open-chain structure can be represented by a hierarchy (or tree) of nodes, that represent either a joint or a segment. This introduces a parent-child relationship among the nodes, where each node has a single parent, except the ancestor of all other nodes, which is the root of the hierarchy [12].

Every connection from one node to another is a joint and its purpose is to compute a local transformation matrix with parameters representing either translational or rotational degrees of freedom. Precise and complex joint models exist, but for the purpose of working with mocap data simple ball-and-socket joint should suffice. Even though this type of joint has almost unlimited degree of freedom, it will be regulated by motion data obeying necessary human anatomy forced by a rigid body of a 3D model.

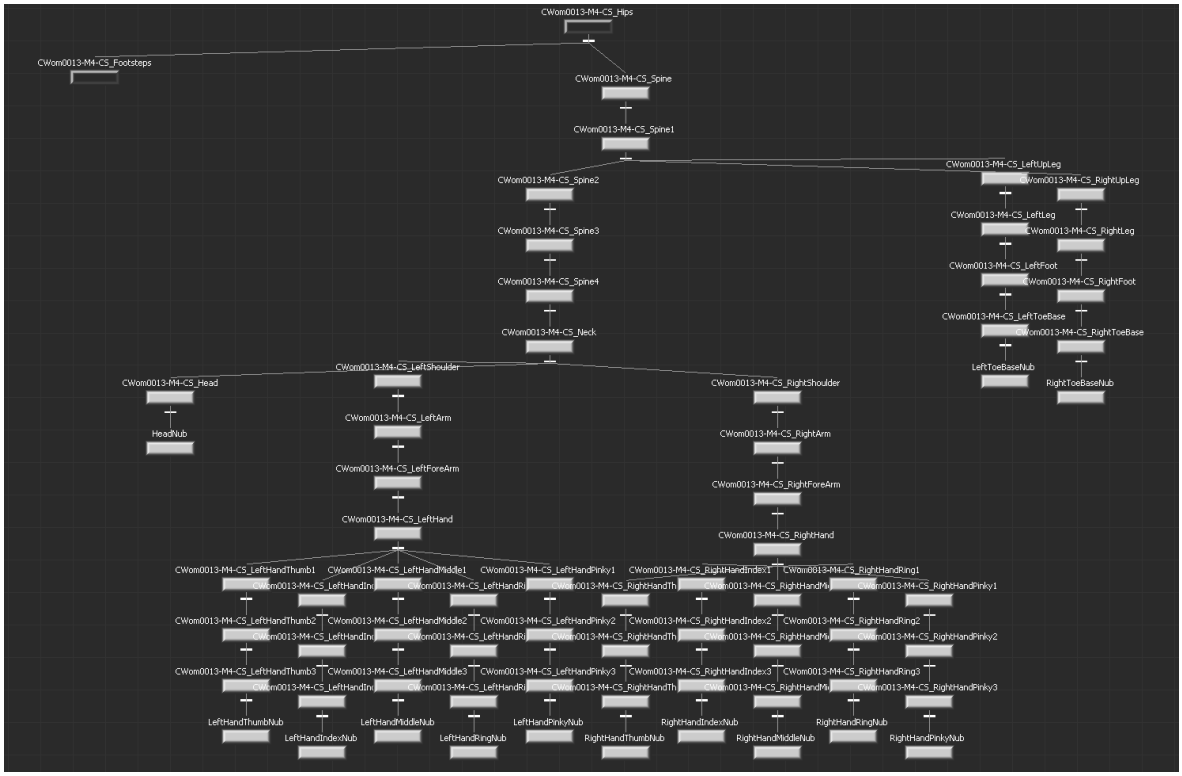

Figure 2 Example of an open-chain hierarchy of nodes starting at pelvis.

2.2.5. *3D model*

One of the first steps to pick or create a model is to decide its purpose. In our case it is obviously a realistic human being with high range of motions for the top body half. Therefore, it has to have very detailed mesh, especially around most bendy parts, such as elbow or shoulder joints. This way any movement (withing reasonable human anatomy of course) would not cause any distortions. This project doesn't need to go into complex details of the mesh, such as elasticity or deformation model. It is common for a model not to high res defined fingers.

Anatomically based Skeleton Design is highly preferable. Yet, it doesn't mean model needs to have all 206 bones defined. All we need is specific segments to be defined. Usually, spine, neck, head, shoulders, arm, forearm. Fingers need special attention though, and possibly even a individual way of capturing movements, but more on that later.

2.3. Motion Capture

One of the approaches to make the 3D model's acting precise is *Motion Capture* (referred as *MoCap*).

Motion capture is the process of recording real life movement of a subject as sequences of Cartesian coordinates in 3D space [4]. These movements are later mapped on computer models. This technology started as a mean to portray a realistic human motion simulation in such a field as education, sport, or even medicine. But being more precise and faster to work with then standard key frame animation it found its way into an entertainment industry.

In theory Mocap technology can be used to animate any kind of model, but best and most accurate results are produced when working with human-like creatures, where movement is bound by laws of physics. And motion that is limited to realism is exactly what we need in this project. The model idealistically should have the same proportions as the actor to avoid the disturbing artefacts like reallocation of model's body parts. The action sequences involving other models might also need additional editing [23].

2.3.1. Optical Systems

Optical motion capture involves using multiple camera sensors. Having same object be captured in 2D from different angles, and knowing exact position of each sensor. It is possible to triangulate this objects position in 3-dimentional space.

→ *Active*

Active systems operate on set of actively emitting infra-red diodes. Each unit has its own integrated processor and synchronization part. Multiple LED systems use software to identify each LED by their relative positions, somewhat akin to celestial navigation.

Since inverse square law provides one quarter the power at two times the distance, main advantage of such a system is high capture distance [5].

Active marker systems (so called *Time modulated active marker*) can further be refined by strobing one marker on at a time or tracking multiple markers over time and modulating the amplitude or pulse width to provide marker ID. 12-megapixel spatial resolution modulated systems show more subtle movements than 4-megapixel optical systems by having both higher spatial and temporal resolution.[7]

→ *Passive*

Passive systems use set of markers coated with a retro-reflective material reflecting infra-red light that is generated near the cameras lens. The camera's threshold can be adjusted so only the bright reflective markers will be sampled, ignoring skin and fabric. If two calibrated cameras see a marker, a three-dimensional fix can be

obtained. Typically, a system will consist of around 2 to 48 cameras. Systems of over three hundred cameras exist to try to reduce marker swap. Extra cameras are required for full coverage around the capture subject and multiple subjects [5].

Biggest advantage of this kind of system is that unlike active marker systems and magnetic systems, passive systems do not require the user to wear wires or electronic equipment [5]. But results in a lower signal-to-noise ratio, resulting in higher marker jitter and a resulting lower measurement resolution.

Passive system can be improved to work in natural lighting conditions using high speed cameras to create *Semi-passive imperceptible marker* system. The specially built multi-LED IR projectors optically encode the space. Instead of retro-reflective or active light emitting diode (LED) markers, the system uses photosensitive marker tags to decode the optical signals. By attaching tags with photo sensors to scene points, the tags can compute not only their own locations of each point, but also their own orientation, incident illumination, and reflectance.[7]

→ *Markerless*

Markerless systems are based on Computer Vision and obtain information directly from camera images. System is analyzing video sequences from multiple view cameras and using complex mathematical algorithms is reconstructing 3D paths.

Systems do not require subjects to wear special equipment for tracking.

2.3.2. *Non-Optical Systems*

All sorts of capturing methods involving magnetic mechanic, inertial motions. Most of the time technology is being self-explanatory by its name.

→ *Inertial System*

Most *inertial systems* use inertial measurement units (IMUs) containing a combination of gyroscope, magnetometer, and accelerometer, to measure rotational rates. As described on Xsens web [8], these rotations are translated to a skeleton in the software. Much like optical markers, the more IMU sensors the more natural the data. No external cameras, emitters or markers are needed for relative motions, although they are required to give the absolute position of the user if desired.

Overall precise but expensive technology.

→ *Magnetic system*

Magnetic systems calculate position and orientation by the relative magnetic flux of three orthogonal coils on both the transmitter and each receiver. The markers are not occluded by nonmetallic objects but are susceptible to magnetic and electrical interference from metal objects in the environment [6].

With magnetic systems, it is possible to monitor the results of a motion capture session in real time. But nonlinear response from causes lower resolution.

→ *Mechanical system*

Mechanical system is using an exoskeleton to capture joint movements directly from an actor. In theory any type of a skeleton can be rigged to capture movement of even unrealistic non-anthropomorphic creature. Whole capture process is very reminiscent of a classical key frame animation technique, except using a physical 3D object.

Mechanical systems tend to be low-cost option, while precision depends mostly on the user. System can capture only part of the body and mechanical gloves have potential to be main capturing method for this project.

2.3.3. *Motion Capture Software*

List of modern software that can manage motion capture animation is vast. Yet not it is not interchangeable and highly format dependent.

2.3.3.1. *Motion Capture formats*

Mocap formats can vary quite a bit. From simple txt/csv files with readable X Y Z coordinates to company specific types that require a decoder.

→ *.C3D*

The most used format in this project is *C3D.* Originally was created for biomechanical research facilities, but later standardized as main source bey Vicon company. The C3D file usually stores a small number of common parameters that describe the 3D data and then allows the users to define, generate, and store within the file any number of user or lab defined data items so that anyone opening the C3D file can access them. This format been in continuous use since 1987 [9].

Unfortunately, C3D neither cannot be read without specialized software, or directly converted into another format. However, there are third-party addons for most of 3D software products designed to load the data. And luckily it is stated on developers web [9] unlike various other 3D formats (Biovision BVH files, OpenSim .TRC and .MOT files etc.,) the .C3D format is a standard that does not change each time a manufacturer releases a new product so data stored in the C3D format will remain readable for a long time.

C3D can also be directly imported from Vicon software as *TXT* or *CSV* file. Coordinates of each marker are stored in a single cell using coma to divide frames.

→ *.BVA*

Is format from Biovision co. that is supported directly by most of 3D software. Is not meant to be read directly but very suitable for visual analysis. BVA describes data as joints forming a skeleton. For each bone in the skeleton there are nine channels that can be used for animation. Translation, rotation, and scale values are represented for each bone and for every frame. Each bone is described in its actual position where there is no hierarchy definition [10].

.BVA was later improved into *.BVH* extending it with ability to store hierarchical information in a same file as 3D data.

Some parts of this project (database) might be stored in BVH for future student to try new approaches using different software.

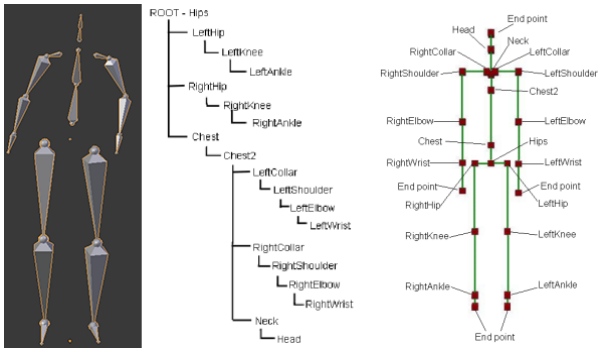

Figure 3 Example of a .BVH skeletal structure (to the left) and hierarchy (to the right)

→ *.FBX*

Film box is a specialized format from Autodesk, which nonetheless is compatible with most of available 3D animation software out there. Unlike previously mentioned formats FBX is not just a motion data file. It can contain every part of a 3D animation project from keyframes to a character data.

FBX motion data part comes in two versions: ASCII text or binary.

Outcome of this project is going to be stored as set of FBX files, and could be reviewed using Autodesk's Motionbuilder directly, or Autodesk's Maya or Blender with a help of .fbx reading extension.

Due to inconsistencies and specifics of different formats mentioned earlier working on this project I tried many different editors. Here is a short description of what could be useful while working of Sing Language Motion Capture Database.

→ *Vicon Nexus*

Nexus is a native software from cameras that was used to capture motion data for this project. It takes feed from up to 24 camera sensors and recalculates it into 3D data on board. The more cameras are used (more covered angles), the higher the quality of marker tracks. This software offers a few builds in features to post-process the data. Such as filling the gaps and denoising. *Shogun* from a same Vicon company is a much better post-processing tool, that is unfortunately was not used in that project because of its high cost.

Nexus is capable of real-time visual feedback on recorded data. This option in combination with a special third-party plugin to stream data directly into 3D animation application could allow for a very quick and easy database creation. That, of course, could happen only in case of a perfect, error free pipeline, which unfortunately was not achieved during in this project. But more on that later.

→ *Motionbuilder by Autodesk*

3D keyframe animation software that natively works with every format described above. Motion builder is based on a real-time 3D engine.

Main output of this project was made within motion builder software.

→ *3ds Max by Autodesk*

3ds Max is a 3D modeling and rendering software for design visualization, games, and animation has its own format *.MAX* that cannot be loaded by any other software, however option to export a 3D character in a .fbx file exists.

In this project this software was used only to open max filed containing characters, rig, texturize them and export in .fbx.

→ *blender*

Free and easy to work with open-source 3D animation software. Natively works with BVH and FBX data. Covers every step of animation production from creating a character to working with key frame animation. Very coder friendly, has good script support and compatible with python. Technically, blender can open C3D data with a third-party plugin, but further process leaves much to be desired.

If question of C3D to BVH conversion is solved, I highly recommend making an attempt of scripting workflow of preparing Sign Language database using *blender*.

→ *MATLAB*

Normally Matlab is most obvious and comfortable option when it comes to cost efficient and scripted data processing. But working on a project that works with lots of visual data I encountered quite a bit of problems that I describe in a next sub-chapter.

Matlab natively doesn't support any motion capture file format. Though external toolboxes do exist, each having its tradeoff.

- C3Dserver
  Paid toolbox directly from format developer
- ezc3d
  python-based toolbox built by an enthusiast (as of 2022 still under development)
- BTK toolkit
  open-source Biomechanical toolkit

→ *C3D specific software*

Mostly paid applications from Motion Lab System with very limited functional.

- C3Deditor
- C3Dserver (external software)

Open-source software to analyze bio data powered by biomechanical toolkit mentioned earlier.

- MLSVIEWER
- Mokka

2.4. *Interpolation*

A realistic animation of virtual characters recorded using motion capture system can elevate user experience and enhance comprehension of animated sign language. However, this method is effective only for pre-determined animations and incapable of real-time generations. Farther, it is virtually impossible to capture every version of every connection between Sing Language modules. Machine learning-based motion synthesis of the whole animation and algorithm-based dynamic simulations are procedural but too complex and compute power demanding.

First reasonable solution that comes to mind is to record shorter parts of a Sign Language covering specific topics and compile them into realistic sentences. Meaning, create an animation interpolation that generates full animation sequences by assembling pre-defined motion primitives or key poses. Interpolations can be performed either tracking particular coordinates, or computing angles between adjacent segments.

In this thesis we prepare test SL dataset, perform basic interpolations, and explore possibilities of a more complex interpolational operations.

This thesis deals with interpolations on two stages of the project.

- First, we have to fill in gaps within each recorded segment, which are introduced by technology imperfections. More about causes of gaps and process of correcting it can be found in detail in *3.2.4. Gap Interpolation.*
- Then we attempt to connect segments withing each other preserving natural look.

Recorded segments are saved in .c3d format where motion data is presented as moving vector (point) x y z in cartesian coordinate system. Therefore, each interpolation can be approach either from 1D or 3D perspective (or in other terms this can be understood as calculations related to coordinates vs angles). Let us take a look at available options starting from the simplest one then moving to more sophisticated options.

2.4.1. *1D approach*

1D or other way to look at it is more typical "signal processing" approach will most definitely be suitable for the filling of gaps and might even be sufficient enough to create a segment between Signs. In this thesis we test in practice efficiency of these basic algorithms.

Timeline for every moving marker can be split into three channels for each of the coordinates *x y z* as it presented in *figure 4*. Withing a reasonable length of interpolation the data can be processed independently for every dimension without introducing extra motional error.

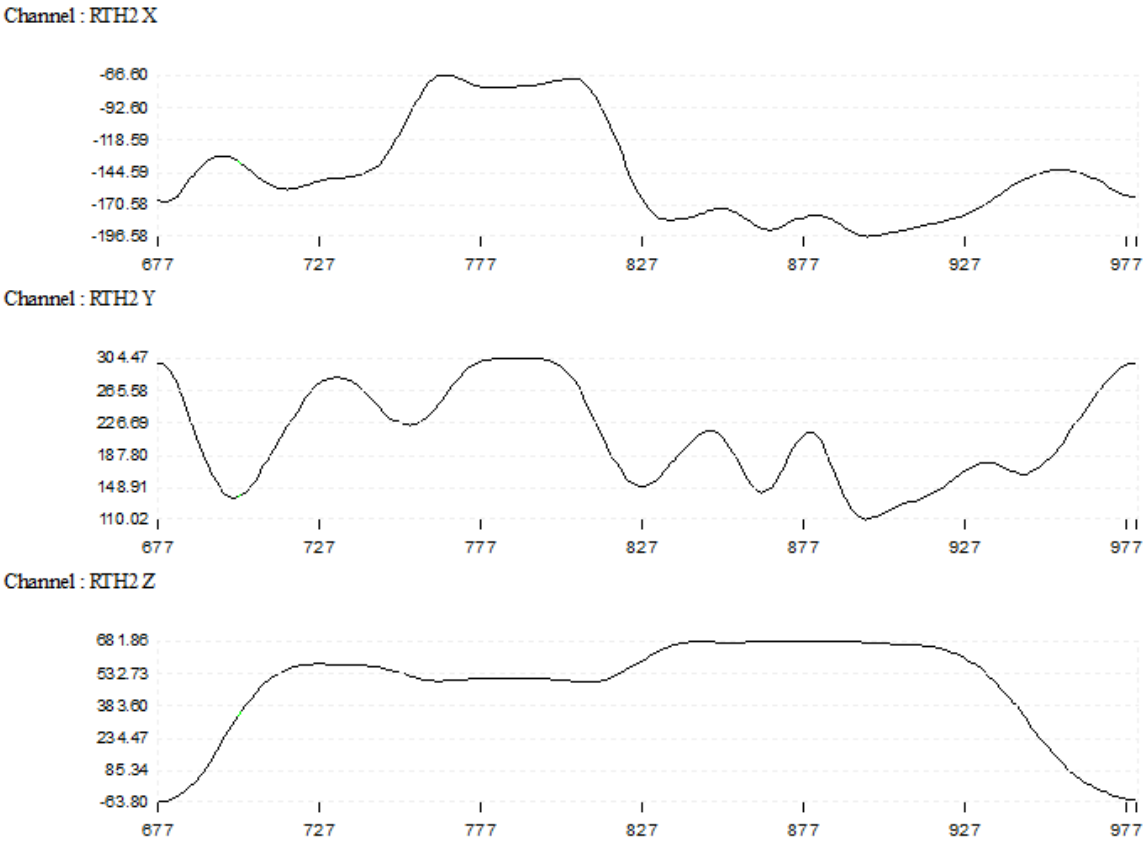

Figure 4 Example of a marker trajectory on axes X, Y, and Z separately

→ *Linear interpolation*

Just to define interpolation it is better to start with the very basic way to connect two points as it is a method of curve fitting in use dating back to 150 BC. Linear interpolation is achieved by geometrically rendering a straight line between two adjacent points on a graph or plane.

It uses linear polynomials (continuity of C^0) to construct new data points withing the desirable range. Our two points (t_0, x_0) and (t_1, x_1) are given by the number of frame t and position relative to an axis x, y, or z. Then solving equation of slopes for unknown x fills in a required straight line.

$$x = \frac{x_0(t_1 - t) + x_1(t - t_0)}{t_1 - t_0} \qquad (2,1)$$

Since preserving naturalistic motion look is of an essence, I do not have high hopes for this as a relevant method. Yet it still might be just not demanding enough to fill 1-2 frame gaps.

→ *Cubic Polynomial Interpolation*

Cubic spline is a general term covering many methods to satisfy a dataset. Here we are talking specifically about a *cubic polynomial* (or shape-preserving piecewise cubic) interpolation (of a continuity C^1) that can be used to intersect 3 or 4 points uniformly spaced points.

Given a set of data points (t_i, x_i) where $a = x_0 < x_1 < \cdots < x_i = b$, then interpolated polynomial curve of degree 3 is a function $S(x)$ satisfying

$$S(x) \in C^2[a, b] \qquad (2,2)$$

and C corresponds to a cubic function $C = a_i + b_i x + c_i x^2 + d_i x^3, i = 1, \ldots, n.$ (2,3)

→ *Spline Interpolation*

Confusing as these namings are a *Spline* is a type of a Cubic spline that is meant to be an improvement on a cubic interpolation and used to intersect 4 point or however more (working within continuity of C^2). A *spline* is piecewise continuous function of cubic polynomials. Spline interpolation using not-a-knot end conditions. The not-a-knot end condition means that, at the first and last interior break, even the third derivative is continuous. The interpolated value at a query point is based on a cubic interpolation of the values at neighboring grid points in each respective dimension.

Comparing to what here is called cubic, spline constructs the curve In almost the same way, with a difference in a choice of a slope to make curve more continuous.

→ *Akima Interpolation*

As described in [14] and explained by MathWorks.com the *Akima algorithm* (or non-smoothing spline) performs a cubic interpolation to produce piecewise polynomials with continuous first-order derivatives (continuity of C^1). Given a set of "knot" points (t_0, x_0) the Akima spline will go through each of the given points (at least 2). The algorithm preserves the slope and avoids undulations in flat regions. A flat region occurs whenever there are three or more consecutive collinear points, which the algorithm connects with a straight line. To ensure that the region between two data points is flat, insert an additional data point between those two points.

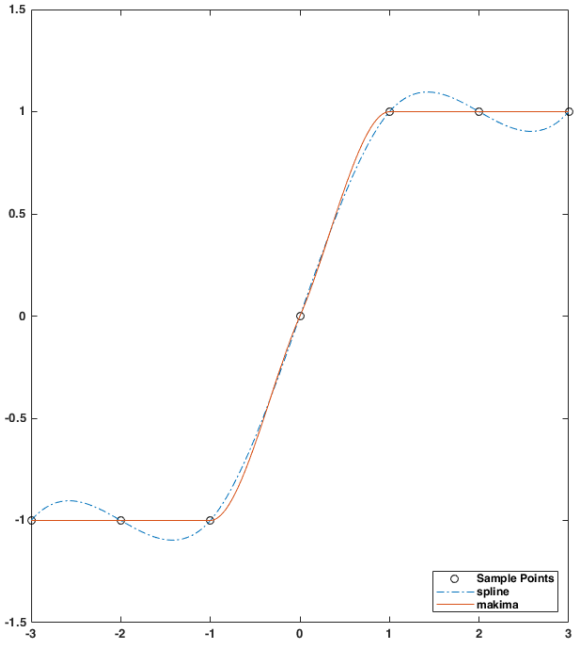

Figure 5 Comparison of akima's behavior interpolation on a steep slope vs. spline

When two flat regions with different slopes meet, the modification made to the original Akima algorithm gives more weight to the side where the slope is closer to zero. This modification gives priority to the side that is closer to horizontal, which is more intuitive and avoids the overshoot.

Compared to the spline algorithm, the Akima algorithm produces fewer undulations and is better suited to deal with quick changes between flat regions. Which might be better suited for the purposes of this project.

→ *Catmull-Rom Spline Interpolation*

Catmull-Rom model (or CM1/2) as well as its modified version (*CMP*) are a third-degree polynomial method. These are the methods that is most frequently used in computer graphics. Catmull-Rom Spline creates a cubic polynomial between two consecutive points in the dataset. The only differentiation modified version has is a method of computing required tangent vectors.

If selected methods in general can be expressed as:

$$\boldsymbol{P}(x) = \boldsymbol{X}^T \boldsymbol{M} \boldsymbol{B};\ \boldsymbol{U}^T = [x^3, x^2, x, 1] \qquad (2.4)$$

where x is an interpolant, $\boldsymbol{X}$ - a matrix containing interpolation parameters, $\boldsymbol{M}$ – fixed coefficients, $\boldsymbol{B}$ – vector containing geometrical information. $\boldsymbol{P}$ – read as control points defining the curve $(P_{i-1}, P_i, P_{i+1}, P_{i+2})$. The j^{th} segment interpolates between two middle control points P_i, P_{i+1}, and P_{i-1}, P_{i+2} are used to calculate tangents.

Knowing that the vectors for CM1/2 are generated using points in the dataset are generated using following equations:

$$P'_i = \frac{1}{2}(P_{i+1} - P_{i-1}), \qquad (2,5)$$

$$P'_{i+1} = \frac{1}{2}(P_{i+2} - P_i), \qquad (2,6)$$

where P_i – i[th] point in the dataset for which the tangent is computed.

The Blended Parabolas methos also belongs to the same family of interpolation methods, but as described in [17] while being mathematically different produces imperceivably different results from Catmull-Rom, and therefore does not require to be treated separately. Difference between regular and modified models were also tested in [17], concluding that CMP not only complicated, but also cannot compute proper tangents in a significant number of points.

→ *Nexus specific interpolations*

Built in interpolation features of Nexus software are also going to be explored in practical part of this project. Included methods are Woltring Quintic Spline, Pattern fill, and Rigid body fill.

Vicon does not disclose exact methodic behind these names, but Woltring spline in literature is generally described as regular polynomial interpolation of a 5[th] order. Pattern fill means taking coordinates from adjacent markers and translating it onto missing markers path. Then Rigid body fill calculates distance between markers that are linked withing same segment, then computing same distances for a missing coordinate.

2.4.2. *3D approach*

Any method described above can be extended to work with 3D coordinates. What is meant by extending is solving polynomial equations as a System of equations (plus in any software every method above is most likely being implemented as a matrix, rather than separate vectors for each dimension). This approach generally can be called as Curve Global Interpolation and normally would not be considered an actual 3D approach.

More complex operations would be required for this task. Operations that are typical when working with computer 3D graphics. Computer animation understands this task as combination of translations and rotations. Meaning interpolation based on computing angles between adjacent segments

This thesis is exploring at theory behind: linear Euler interpolation, spherical linear quaternion interpolation, and spherical spline quaternion interpolation.

But first we need to lay grounds about Euler space, quaternions and define rotation as it is described in [15].

→ *Interpolation of rotation*

Rotation can be represented by two modalities

- *Euler angles* (or rotation matrixes)
- *Rotation represented by quaternions*

On the bright side, besides rotation, 3x3 homogenous *Euler matrix* can represent other transformations, like projection, scale, and translation. Even though Euler angles are considered a "classic" method used by most of the 3D content creation software it doesn't come without its caveats. Rotation matrixes treat rotations independently along each axis. This approach leads not only to rotation order affecting an interpolation curve, but also can cause a so-called gimbal lock.

Gimbal lock is a common for Euler rotation artifact when we lose one degree of freedom as it is shown with explanation in the *figure 6* [1].

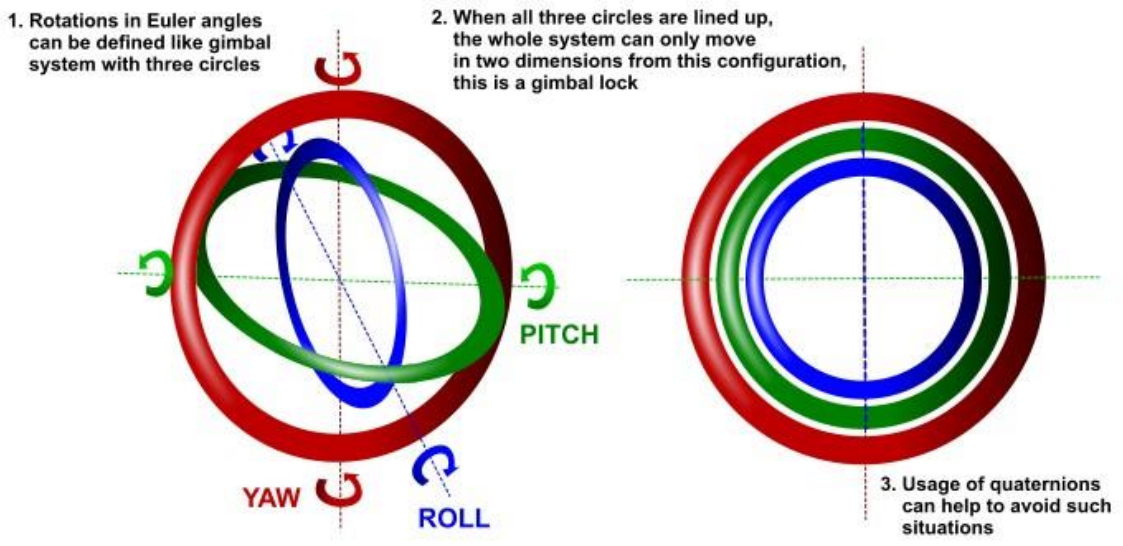

Figure 6 Explanation of a gimbal lock

Rotation in Quaternions, on the other hand, is performed along only one axis, which produces more predictable results.

[1] Image source: https://around-the-corner.typepad.com/adn/2012/08/avoid-gimbal-lock-for-rotationdirection-maya-manipulators.html

A *Quaternion* is a compact data storage of only four numbers (instead of nine in Euler matrix). It has a big advantage of simpler composition for sequential rotations. As a tradeoff for complex mathematical model quaternion system is more computationally efficient.

Just like for Euler model most of modern 3D software has Quaternion support as next interpolation that is done with quaternion-vector pairs (quaternion for rotation and vector for translation).

→ *Linear Euler interpolation model (lerp)*

Linear interpolation normally performed between two segments of Euler angles. If a point in 3D space represented like $P = [x, y, z] \in \mathbb{R}^3$, then vector from world's origin to that point is vector $v_0 = (x_0, y_0, z_0) \in \mathbb{R}^3$. Let's imagine a unit spere, where multiple vectors lie on its surface. Another vector can be represented as $v_1 = (x_1, y_1, z_1) \in \mathbb{R}^3$.

Linear interpolation between these vectors can be described as:

$$Lerp(v_0, v_1, h) = v_0(1-h) + v_1 h \qquad (2.7)$$

where $h \in [0,1]$ is an interpolation parameter, describing position between two point 0 is a starting position at v_0 and 1 is a destination point at v_1.

Figure 7 [2] shows an example of rotation of 8 points connected into a rigid body rotated within X Y Z axes by an angle of 10, 50, 70 degrees respectfully.

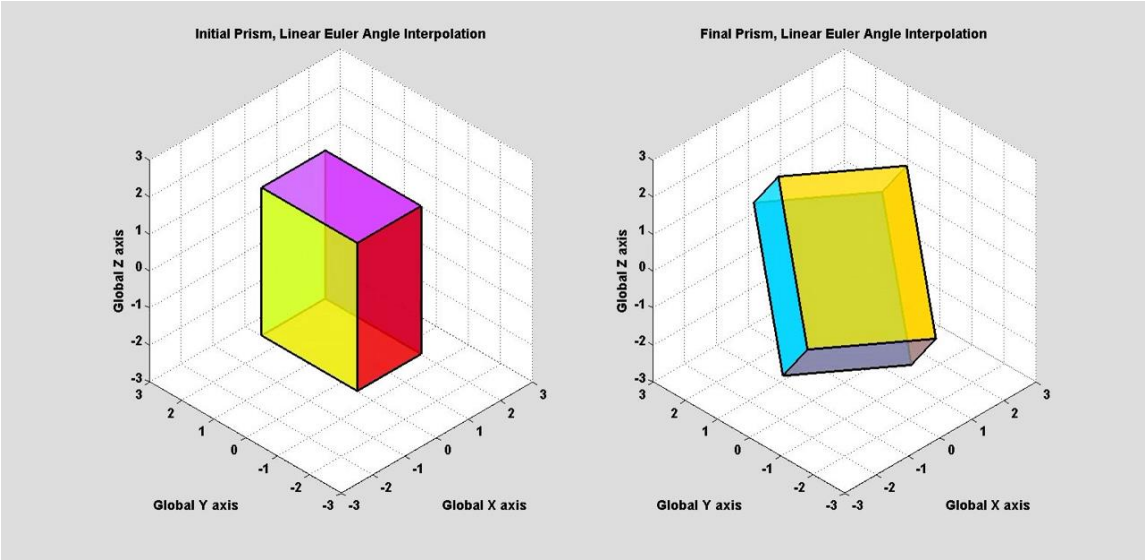

Figure 7 Rotating rigid body using Euler Linear Interpolation.

[2] Image source: https://youtu.be/80Y3DwOqaEA

→ *Quaternion interpolation*

Typical form of a quaternion is $q = s + ix + jy + kz$, where $s, x, y, z \in \mathbb{R}$ and set of complex numbers $i^2 = j^2 = k^2 = ijk = -1$. This four-coordinate system with scalar and 3D vector is used to describe rotation of a solid body.

When two quaternions in a four-dimensional space, connected by a straight line, this line gives us orientation of an interpolation. In addition, angles between segments with a common joint (articulation angles) needs to be converted into quaternions.

Giving these basics two interpolation models can be defined.

→ *Spherical Linear model (slerp)*

This model interpolates rotation along the shortest path on a unit sphere at a constant velocity.

If H is a set of quaternions, with $p, q \in H, \cos(\Omega) = p * q$, then *Slerp* is represented by next equation coming from 4D geometry:

$$Slerp(p, q, h) = \frac{p \sin\big((1-h)\Omega\big) + q \sin(h\,\Omega)}{\sin(\Omega)} \qquad \textbf{(2,8)}$$

where $h \in [0,1]$ is an interpolation parameter of a same structure as in Euler model.

This approach might cause a sudden change of angular direction when performing a series of rotations (in other words the curve is not smooth at the control points), making keyframes visible. Plus, typically we have more than two key poses to connect. Over these reasons *spherical spline* quaternion interpolation model might offer an improvement over standard linear model.

→ *Spherical Spline model (squad)*

In a search of a higher order of continuity we can explore a spline variation, that is also called as spherical and quadrangle model (or Squad for short) is the spherical cubic equivalent of the Bézier curve. This curve is defined as a curve of a 3rd order, where two additional points define a tangent for each control point. A special case occurs when the tangents coincide in the control points. This results in differentiability in the control points presented Squad and then proved the continuous differentiability of Squad at control points [15].

$$Squad(q_i, q_{i+1}, s_i, s_{i+1}, h) = Slerp(Slerp(q_i, q_{i+1}, h), Slerp(s_i, s_{i+1}, h), 2h(1-h)) \qquad \textbf{(2,9)}$$

where $h \in [0,1]$ and $s_i = \frac{q_i \exp(-\log(q_i^{-1} q_{i+1}) + \log(q_i^{-1} q_{i-1}))}{4}$. $\qquad \textbf{(2,10)}$

Figure 8 Comparing performance of Slerp(orange) vs. Squad(blue)

Performance of both presented quaternion models can be compared in the *figure 8* from [16]. We can see a shortest path on a surface of a sphere created by Slerp vs. polynomial curve created using Squad.

2.4.3. *Machine learning*

Neural network might be far too complex solution withing this thesis and lay beyond the scope of our studies. Nevertheless, since it has already shown quite a lot of promise in adjacent content creation spheres, in most basic way I would like to mention techniques that might be a suitable solution for this project.

→ *Generative Adversarial Network*

As it is apparent from the name this network is designed to *Generate* something. Being *Adversarial* means it consists of two separate networks fighting against each other. One's improvement forcing another to improve as well, ultimately training the generator to create needed data.

Basic principle is better explained with a help of a block diagram (*figure 9* [3]) where we can see two main components:

- The *generator* learns to generate plausible data. The generated instances become negative training examples for the discriminator.

[3] Image and info source: https://developers.google.com/machine-learning/gan/gan_structure

- The *discriminator* learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

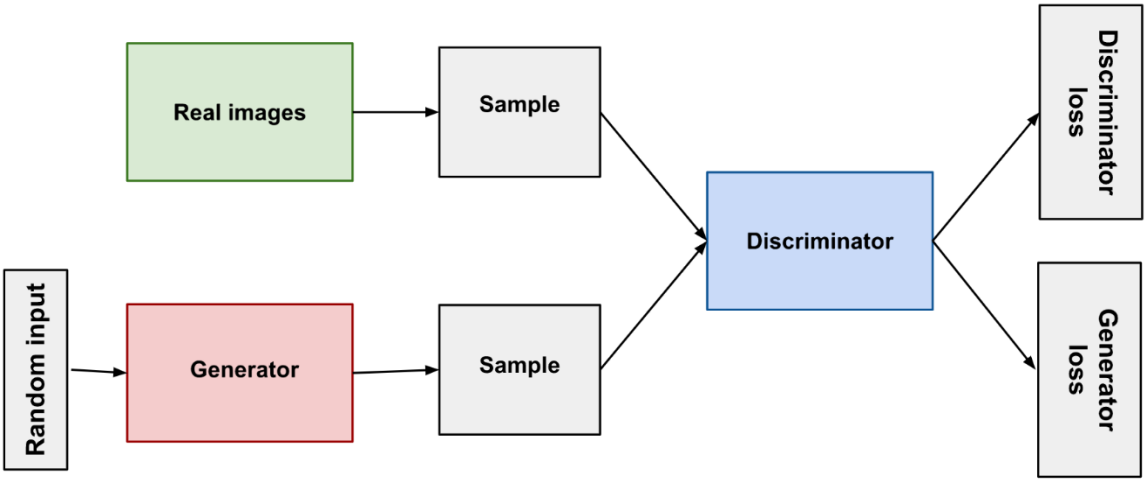

Figure 9 Principle behind GAN algorithm

At the beginning of the training phase, noise is being passed through generator to create obviously fake data. That data is then passed to discriminator. Every time discriminator compares received data with reference and claims data is real – generator has succeeded, and vice versa. But until discriminator is not fully trained it returns random verdicts on whatever data is real of fake.

Every time it called data fake – discriminator has succeeded at sending data back to generator. Over some time and multiple iterations of the loop networks get better and generated data is getting close to the reference.

This type of NN is commonly used in image generation but can be fitted to process coordinates or rotation matrixes recorded using motion capture.

→ *Recurrent Neural Network*

A recurrent neural network is a special type of an artificial neural network adapted to work for time series data or data that involves sequences. RNN is already in use in robotics to teach robots how to move arms to pick up objects. Application that is close to our goal of creating a movement of a rigid body withing a 3D space.

As described in [18]: ordinary feed forward neural networks are only meant for data points, which are independent of each other, and only work with pre-determined sizes (fixed-size inputs produce fixed-size outputs). However, if we have data in a sequence such that one data point depends upon the previous data point (for example, data for machine translation), we need to modify the neural network to incorporate the dependencies between these data points. RNNs have the concept of

'memory' that helps them store the states or information of previous inputs to generate the next output of the sequence. Making it possible to create a many-to-many or many-to-one network.

Just like a GAN option, RNNs work on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.

Back-feeding loop-based networks provide all of the necessary advantages for a task such as generating motion capture keyframes.

- Handle sequence data.
- Handle inputs of varying lengths.
- Store or 'memorize' historical information.

Withing multiple hidden layers generated data being looped a stage back and compared to the required inputs, creating weights and biases, based on which algorithm is updated. The Recurrent Neural Network will standardize the different activation functions and weights and biases so that each hidden layer has the same parameters. Over training time multiple hidden layers are ultimately going to be merged into one, creating an efficient and fast generating algorithm.

2.4.4. *Related Research*

As a last sub-chapter, I would like to present miscellaneous research findings that can be relevant for the future of the project.

- During final stages of this project a relevant article [24] came out taking on Missing Marker problem approaching it from perspective of data optimization. Explored optimization parameters included the restrictions between markers' distance and their movement dynamics in a set of empirical principles and equations. Unlike our work on Sing Language database, where we encounter genuinely arisen mocap artifacts, study [24] applied artificially generated gaps to simulate active loss of data. However, fully optimized algorithm has shown very promising results withing a specific dataset and would be further tested. Results include not only interpolations withing marker timelines, but also extrapolations on initial and final frames. Taking into consideration how much manual effort goes into correcting errors in recordings fully automatic approach might be of an essence for a project like mocap based sign language.

- Another research taking on a similar problem was published just a couple of month ago. In [25] Missing Markers problem is approach using neural network. They have tested *Feed Forward Neural Network* and *Recurrent Neural Networks* in different variations. Proposed approach varies from typical NN procedure in a training of an algorithm. Training data is not being feed in a massive amount in advance to form a predictive model. Instead, each sequence is treated separately

to try to reconstruct the gaps in individual motion trajectory on the basis of its own data only. This makes sense as long as the marker motion is correlated and most of the sequence is correct and representative enough. Paper states that neural networks perform worse when dealing with short gaps and outperform classic methods when facing longer ones. For now, given approach does not incorporate skeletal information, but there are plans to include it in the future.

- [26] is a research on using Deep Motion Interpolation Network for human skeleton animation. Which makes a lot of sense, since typical rule-based interpolation methods view every joint of skeletons as an independent variable and does not considerate the influence and interrelation of connected joints. In this research they view joints, rotations, and positions of human motion as motion images and complete missing area by DMIN. This network works on a very similar principle as GAN described in 2.4.3. improving it with a so-called Hourglass block, which takes multiscale information into account., and where features are processed across all scales and consolidated to best capture the various spatial relationships. This method requires quite a bit of compute power, but does produce convincing results, and by subjective evaluation of [26] authors algorithm generates more natural movement, that classis methods, such as Slerp.

Unfortunately, none of the listed studies works with exact problem that this thesis does, and it is hard to predict, how these algorithms would perform with more narrowly targeted job of reconstructing a Sing Language. Especially, there is no data if it can produce any results working specifically with finger or face animation.

Chapter III
Experiment

Here we would like to describe in detail an experiment with motion capture data, that took place in December 2021. With performance from a native Sign Language speaker a small dataset of SL has been captured using passive optical motion capture system from Vicon.

3.1. Harvesting data

Database of signs was recorded over one session under a supervision of an employee at CTU's Faculty of Biomedical Engineering.

Acquisition of motion capture data requires three simple preparation steps: marker set placement, camera placement and camera calibration.

3.1.1. Marker set

Most typical marker set is so called Plug-in-Gait Marker Placement Model as illustrated in the *figure 10-A* (For full description of a model look [13])

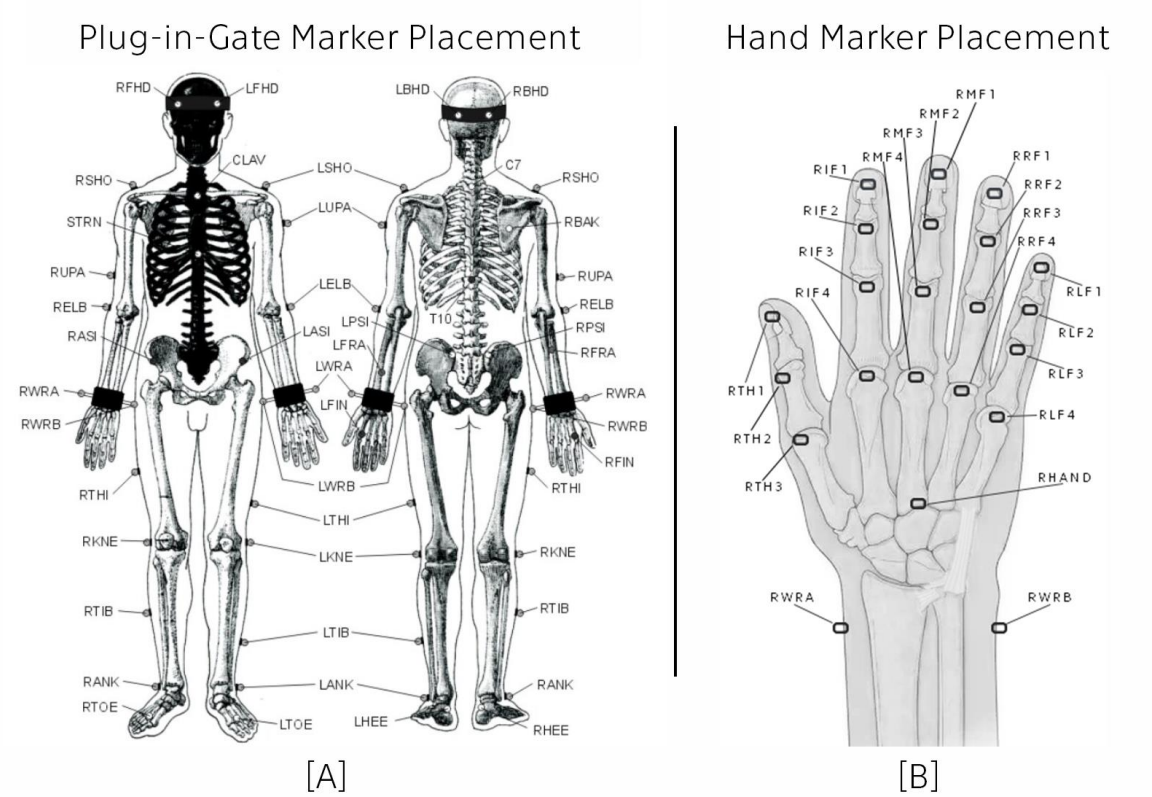

Figure 10 Standardized marker placement models

This marker set is suggested and supported by Vicon to be used best with their equipment and software. This model, however, does not include specifications for palm and face capturing. Finger marker placement can be seen in the *figure 10-B* (prefix R means Right hand, and is replaces for L for Left hand respectfully).

Apart from all of the issues related to facial animation and comprehension process of acquiring facial mocap data introduces further complications. There is no marker set models that could capture all of the nuances of facial expressions or track of lips or tongue. Even though tongue doesn't play a main role in sign language, it is still preferable to have it tracked at stage of completion of this project. Marker based motion capture system simply doesn't have enough resolution for this task.

According to models in the *figure 9* over all 41 markers has been placed. 3 head markers, 9 body markers and 14 on each palm. Since on theory stage it already has been decided to focus this project mostly on hand motions and manual components face capture will include only 1 jaw marker placed on the chin. Just as shown in the *figure 11*. Unfortunately, mistakes were made, and we did not add any hip markers, crucial for the transferring of motion data onto virtual avatar. More on how this problem can be solved in *processing* section.

Figure 11 Used marker set on the actor

It is important to know these placements to be able to navigate through next visual steps.

3.1.2. Camera setup

To record motion data were used Vicon professional optical system available at Kladno CTU department. System is based on passive optical principle and consists of 7 IR cameras (*figure 13*) placed as shown in the *figure 12*.

With all cameras positioned in the way so they cover as much signing space as possible we were able to see each marker position on seven 2D maps with different level of image quality.

As a next step each camera must be manually focused to deliver the clearest picture (basically task is to avoid blurry images and make sure no marker is missing on each camera). Calibration dials are circled red in the *figure 13*.

Although, Vicon system has masking algorithms to negate parasitic light and reflection, it could also be helpful to remove external sources of light or objects reflecting infra-red spectrum.

Now, before 3D tracks can be triangulated it is crucial to let Vicon software know, where exactly each camera is located in the world space coordinates. This could be done by recording a number of images of points whose space location and relation we already know. In other words, we need to wave a specialized Vicon Wand tool inside of the captured space.

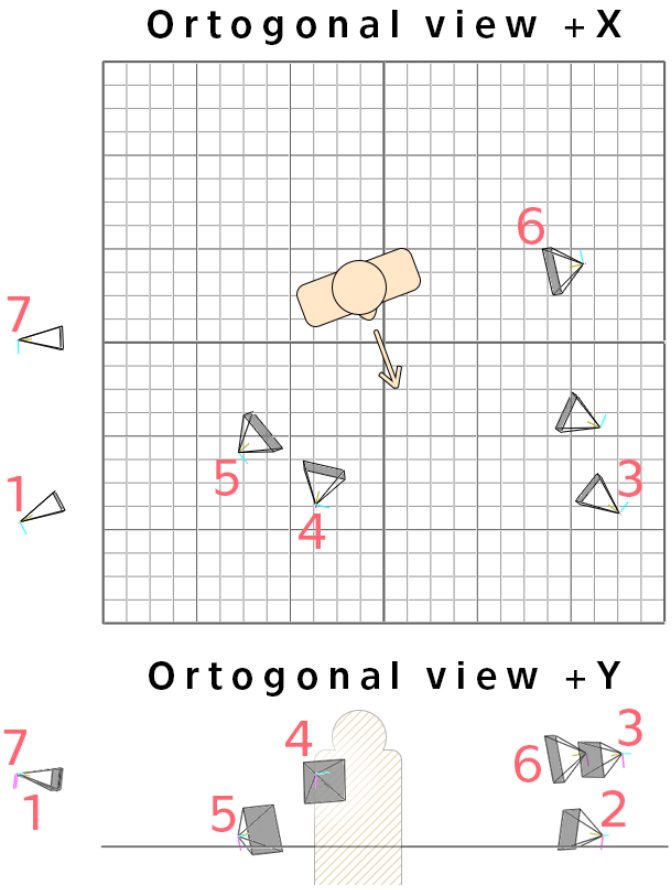

Figure 12 Camera placement

Figure 13 Vicon Bonita camera

Wand has 5 markers strategically placed in a T shape withing a known distance from each other. Process of tactical waving should continue until Nexus software gives feedback, that cameras has been placed into 3D capture space.

As a last step we have to define the 3D coordinate system (zero out the axes). It is achieved by placing same wand tool where we would like our system to start.

Now frame rate can be set and proceed to recording the data. For a regular mocap session (movie production for example) usually 30-60Hz is enough but working with some types of bio data, like for example a Sign Language, it is suggested to keep fps at least 100Hz.

This is the step where second mistake was made. While virtual camera setup is identical to real world setup, camera offsets and angles were not properly registered. As can be seen in *figure 14* actor's figure does not look at the right angle and is clipping through the floor. In no meaningful way does it affect quality of recordings, but it does introduce a slight post-processing challenge, which we discuss in a corresponding sub-chapter.

Figure 14 Actor making a T-pose

→ As system is ready, capture phase can begin.

First recorded shot (especially with a new actor) must always be a so-called T-pose. That is when yet another mistake has been made. Our database does not include a typical T-pose. However, we did include a different calibration pose, that is far harder to work with, but it does not render recorded data useless. Further I would explain how it is possible to correct that mistake during post processing.

3.1.3. Database

After this point process is simple and straight forward. Record necessary shots for a database. Vicon Nexus does include a script to detect movement and automatically start and stop the recording.

After consulting with specialist in Sign Language studies Mgr. Lenka Okrouhlíková, Ph.D. it was decided to work with limited dataset of simple day-to-day words. All word come in pairs of an adjective and a noun. For example, Black Cat, Wooden Table, Pretty Mother (recorded in Czech SL: černá kočka, dřevěný stul and krásná matka, respectfully).

Only fixed signs were used. Any classifiers, modifiers or specifiers were avoided as well.

At first every word on the list was recorded separately. Then each adjective was paired with each nous, making pairs that make sense. This way interpolated data can be evaluated in comparison with natural signing movement.

Each word or pair is recorded with a start and an end in a zero position – hands down close to thighs, head straight, mouth closed.

Full list of words and pairs can be found in a spread sheet in *attachment #1.* On a side note: every word pair is recorded as noun + adjective. One of the specifics of a Czech Sign Language is interplacebiity of a part of a sentence, but noun being first is more common.

This concludes stage of recording the data, but database is far from being ready to be used in any tests. Next sub-chapter tells us about every caveat this project has encountered.

3.2. Preparing database

Next part of the project is working with prerecorded 3D data to make them suitable for further testing. In this part we would discuss how to correct errors of recordings caused by technology and capture process imperfections, as well as how to compensate for mistakes that was made in previous stages. I will always try and include two ways each problem can be solved: scripting and visual approach.

Giving the fact that we are working with very sensitive data, that supposed to deliver information withing sometimes very subtle movements it sometimes proved better to process data "by hand" one by one, rather than just running whole set trough the scrip. But more on that later.

3.2.1. Reconstructing

First thing that we can see after loading our recorded data is nothing. That is due to the fact that data is still in a separate 7 containers, each representing 2D view from single camera, as shown on example in the *figure 15*. This is being resolved simply by running a reconstruction pipeline right in the Nexus interface.

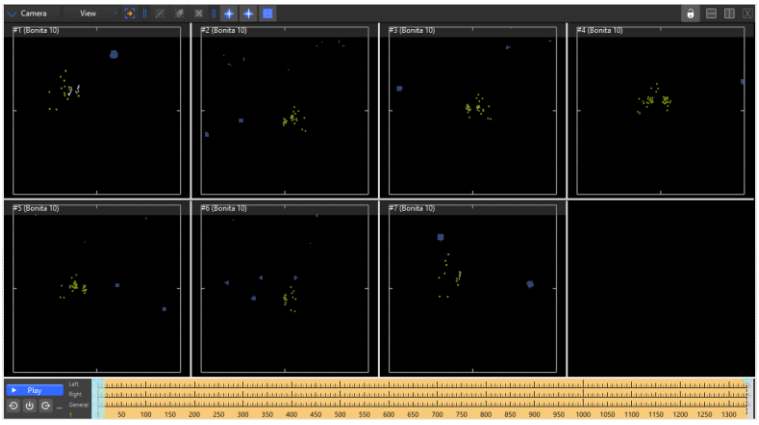

Figure 15 Multi-Camera 2D marker view in Nexus

3.2.2. Labeling and segmentation

Reconstructed data appears to be just a set of unlinked markers as shown in the *figure 16*. Among these white dots human figure can barely be identified. But visual aesthetics is not the only reason to label and separate segments. It is needed to be able to follow each single marker track later. First for debugging purposes, and then for application on the 3D model. In that plain form we cannot even export 3D data yet, as it would be just N random markers on M frames.

Figure 16 Reconstructed 3D tracks model in Nexus

First, we create new subject withing the project in Nexus. Then add marker names used in recording according to models shown in a previous chapter.

Created markers can be assigned to respectful marker on the subject's body. Following this step segments can be created. *Segments* can be predetermined per model, but usually they just follow simple rule of being a cluster of markers that does not change position toward each other. For example, Head or upper arm would be its own segment. Advised not to use more than four markers per singe segment.

Prepared segments must be linked with each other determining hierarchy from top to bottom. Finished model can be seen in the *figure 16* (segment links are invisible for better view). Best to give each segment its own color so it is easier to work later on.

This process in one-and-done, at least per actor. Now structure can be saved as a separate file and used later to be applied onto different recordings withing a session.

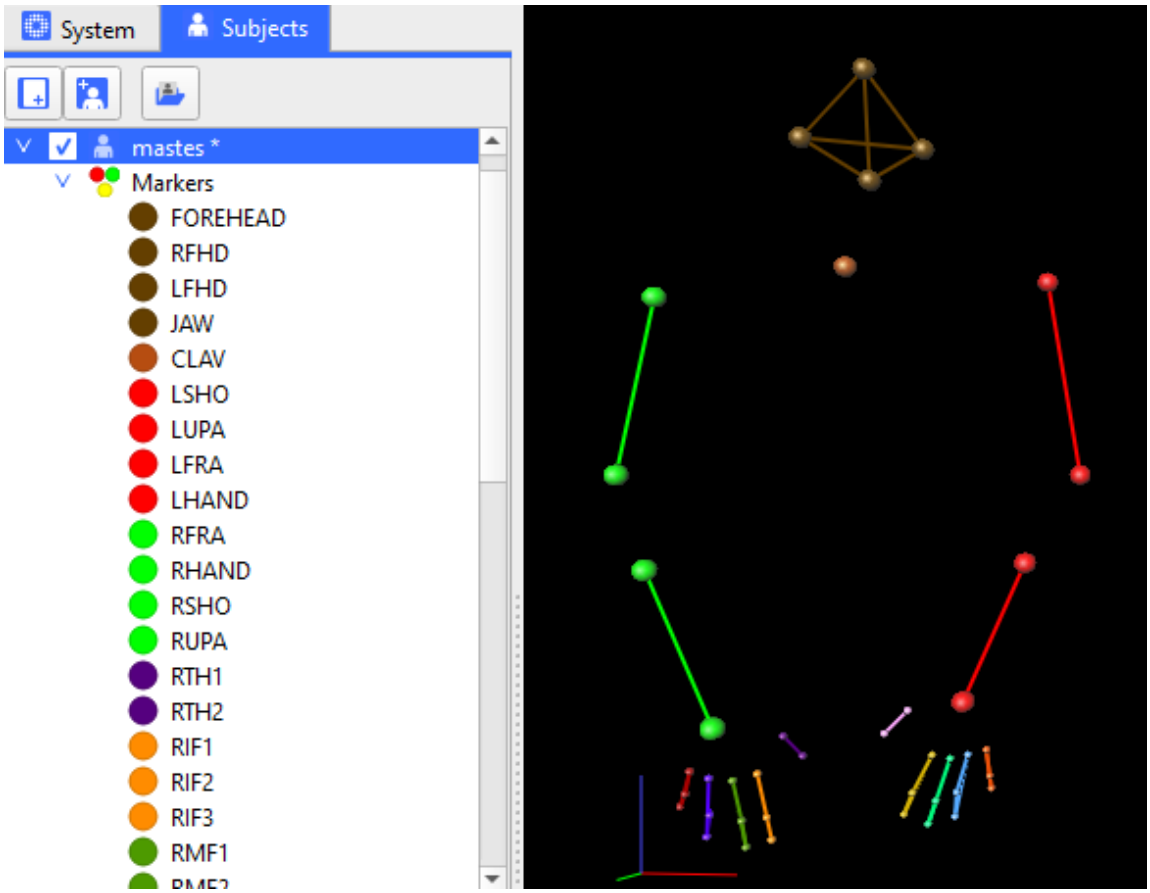

Figure 17 Example of a subject's structure: labeled, segmented, and colored for convenience

3.2.3. Fixing mislabeled markers

First imperfection of technology that this project has encountered is mislabeling withing a recording. What is meant by that is some markers are being lost by reconstructing algorithm and either not being given a label at all or mistaken for another marker. Perfect example of both can be seen in the *figure 18*.

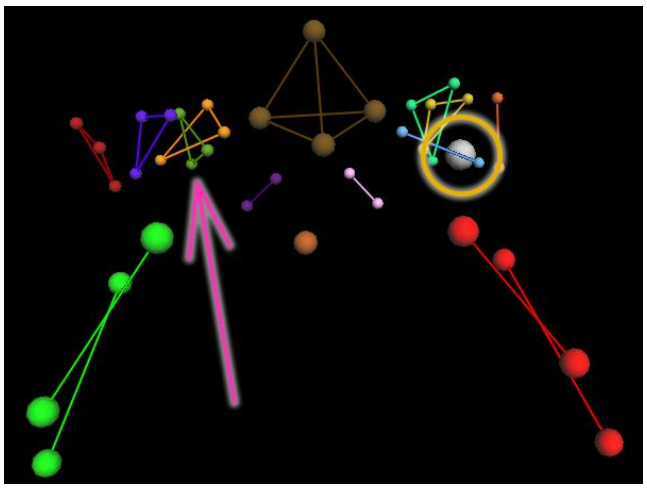

Figure 18 Mislabeled marker circled yellow; Misplaced marker pointed purple

This happens because of two main factors:

- Simple miscalculation withing algorithm
- Insufficient coverage of signing space

The root of both of these elements most likely lies withing the fact that finder markers are places withing very small distances from each other. Meaning there is no workaround this issue. Unless another capturing method or system is used. It was suggested by supervisor from FBMI CTU that ultimately this project would require at least twice as many cameras to cover whole signing space. Not only markers are too close to each other, but some of them are also being invisible to multiple cameras while actor perform specific signs. These signs being either obstructed by another body part, or simply facing away from the camera (basically covering itself). The worst-case scenario is when sign requires hands to move very close to each other, mixing every finger marker with another one.

Neither of these problems can be solved and has to be dealt with after the fact. This is probably the most frustrating and time-consuming part of the project. It is done by scrubbing through the footage and relabeling every faulty marker. Colorized and linked segments do help a lot with this process, because every misplace is instantly visible.

3.2.4. Gap interpolation

Once all of the markers are flowing through the tracks they supposed to, we can begin restoring missing data. Once again, these errors are caused by technology and approach imperfection and cannot be avoided unless we use different setup or capturing system. Number or missing tracks can be found in Nexus in quality tab, as *figure 19* shows. Here we can see total among of gaps and length of each individual gap. Number of gaps is color coded (from green to red) for each marker. Right underneath is a timeline, where missing part for selected marker is labeled yellow.

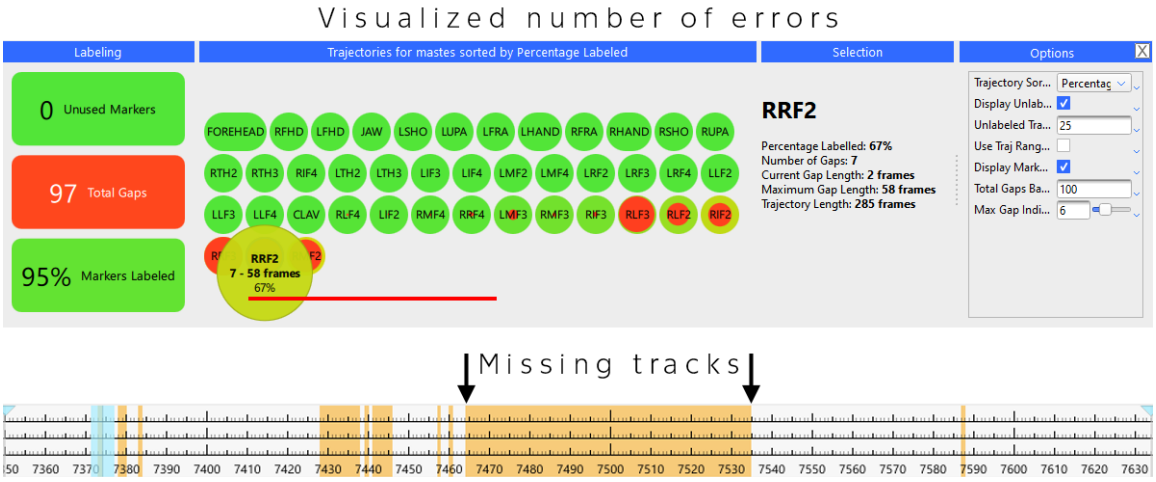

Figure 19 Markers before corrections and gap fillings

In this project we explore two different approaches to interpolate the track gaps.

3.2.4.1. Matlab

Scene by scene 3D data can be exported into C3D container directly from Nexus, no plugins required. In case of exporting data in bulk further separation would be required.

In order to open exported .c3d data in Matlab requires an additional toolbox. Options to use are as follows:

→ *BTK Toolkit*

The Biomechanical ToolKit project started in 2009. Internally, the code was based on the C3D-like container and a pipeline mechanism to process the data. Unfortunately project it stuck in early stage of version 0.3. Direct Matlab toolbox has not been updated for more than a decade now and does not operate with Matlab versions 2018 and newer (read as operate but with all the same hiccups as any pre-2015 code on newer versions). Python version was updated over 5 years ago.

→ *ezc3d*

It is an enthusiast made free toolbox based on python. This toolbox does not have precompiled .m files, but with a knowledge of python it can be compiled from authors souse code.

→ *c3dserver*

Exists as a standalone software, works with C++ and visual basic, but on multiple devices I failed to make it run without constant crashes.

Stable run can be achieved accessing c3dserver through Matlab. Once standalone version was installed, `c3dserver.dll` path needs to be added into Matlab's directory and now sever can be called via console as `>> test_variable_name = c3dserver`. List of available functions can be called with a command `>> methods(test_variable_name)`. The M-files can now be used as typical Matlab functions.

Or as a simpler approach we can load limited set of functions compiled by users (files .m can be found in attachment 5)

Now, with everything installed we can operate on .c3d containers almost as any other imported files.

Data can be loaded using:

```
[Markers, VideoFrameRate, AnalogSignals, AnalogFrameRate, Event,
ParameterGroup, CameraInfo, ResidualError] = readc3d(FullFileName);
```

We are going to be working with only one variable [Markers] from that list, but we should preserve most of the remaining parameters as they are needed to create a functioning .c3d container on export. Markers are now stored as a m-struct [frames $_x$ marker $_x$ axis].

Isolating one marker's timeline we can preview 3D data stored as separate values for X Y Z axis (as shown in the *figure 20*), interpolating is viewed as 1D task.

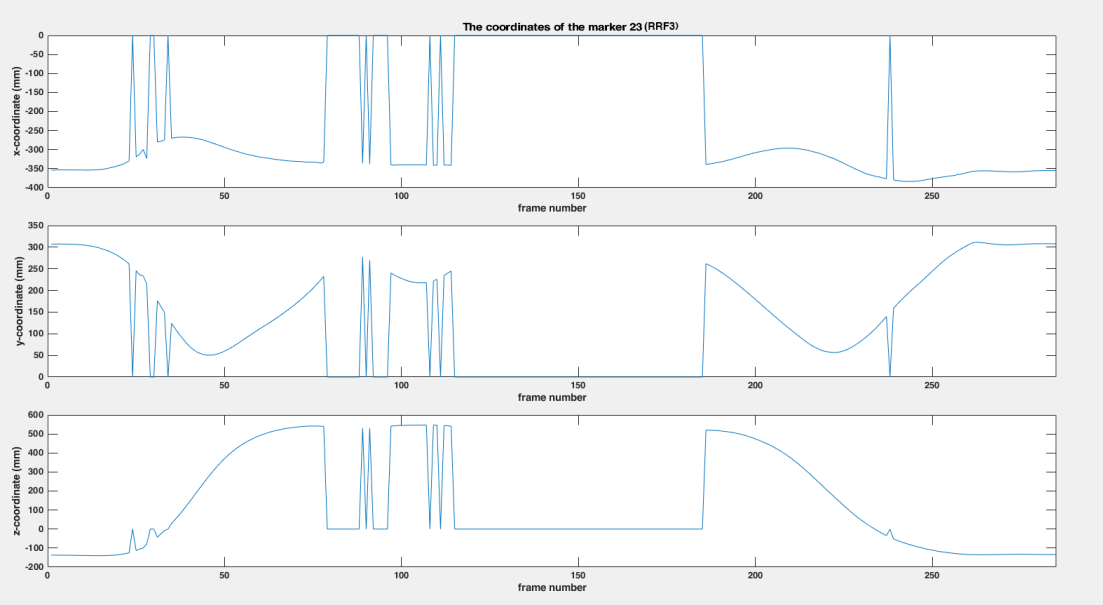

Figure 20 One marker timeline with multiple gaps (single movement viewed from 3 axes X Y Z)

As missing marker points are stored as zeros (with residual tag -1), first step is to locate and isolate all non-zero values (or values without residual tag). Then create interpolation vector and run 1-D data interpolation function. Then faulty timeline can be replaced.

```
tempM = Markers(:,n,m);
temp_l = 1:length(tempM);
x = find(tempM ~= 0);
v = tempM(tempM ~= 0);
New_M = interp1(x, v, temp_l, 'spline')';
```

Now procedure can be applied to every track in a set. Where *n* number of the marker as they were assigned in Nexus, and *m* from 1 to 3 for X Y Z respectfully.

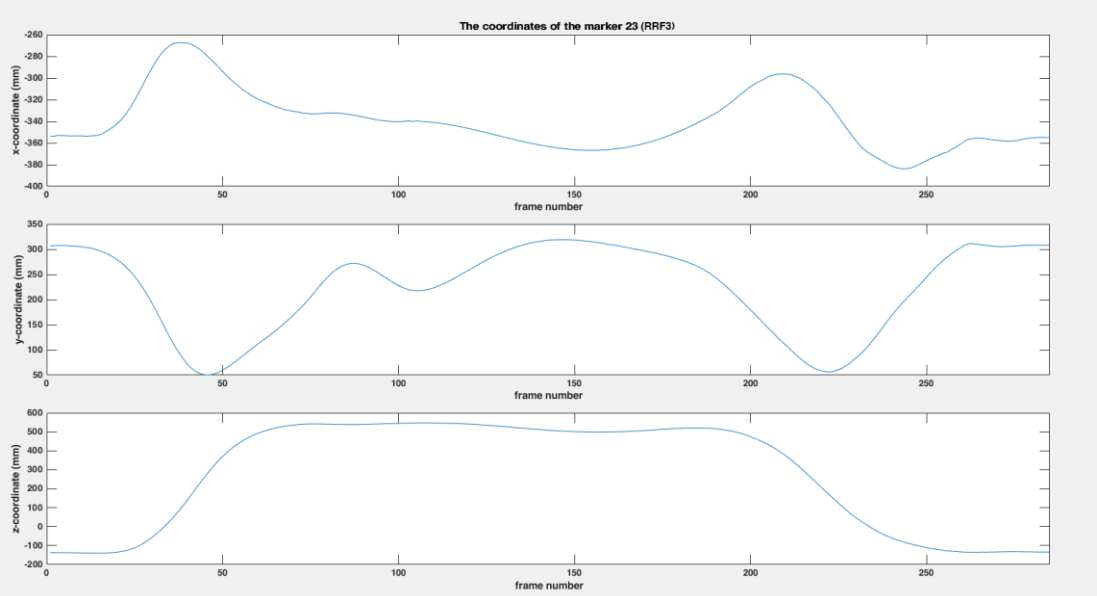

Figure 21 One marker timeline figure above (14) timeline with gaps removed using spline (single movement viewed from 3 axes X Y Z)

For this task three main methods were tested. Linear, cubic and spline interpolation. Comparison of the methods can be seen in *figures 22,23,24* Linear and cubic as expected are not precise enough for longer gaps. However, being less demanding on computing power it can be useful to run these interpolations for gaps of 6 frames and shorter. Especially if we account for the further denoising and smoothing procedures that we would apply. Spline on the contrast shows very promising results. But let us review every result it produces.

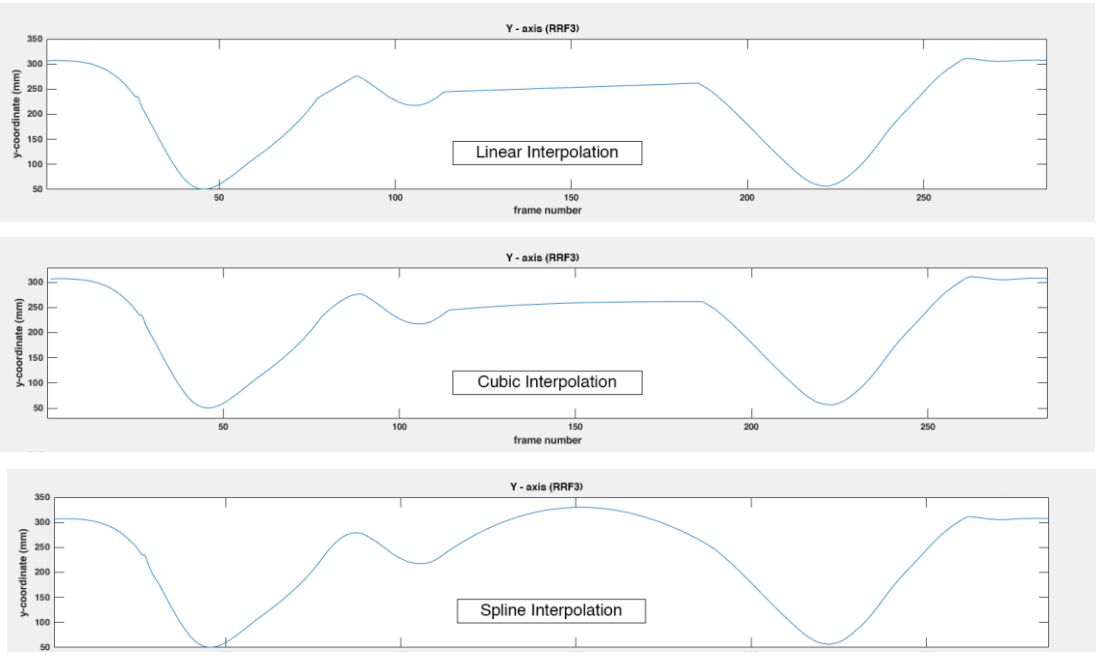

Figure 22,22,23 Marker trajectory within Y-axis filtered using linear, cubic and spline interpolations.

As seen on the next *figure 25* sometimes it causes faulty lines. In this particular case it is most likely given by an artifact circled red, which can be mitigated by removing local outliers, using for example function:

```
[cleaner_tempM,pos_outlier] = rmoutliers(tempM_local);
```

Or this might as well be cleaned after the fact during filtering stage. This approach results in clean graphs and smooth marker movement, which unfortunately do not always correspond to reality. Meaning, the more inattentive scripted processing is applied to the data, the less it is usable in our goals. Just a reminder, we are trying to preserve as much of original actor's finger movement as its job is to deliver information.

At first, trying to save time and running all of the files trough script I ended up spending more time coming back, verifying quality and correcting results. That is when second approach of using onboard Nexus features comes in handy.

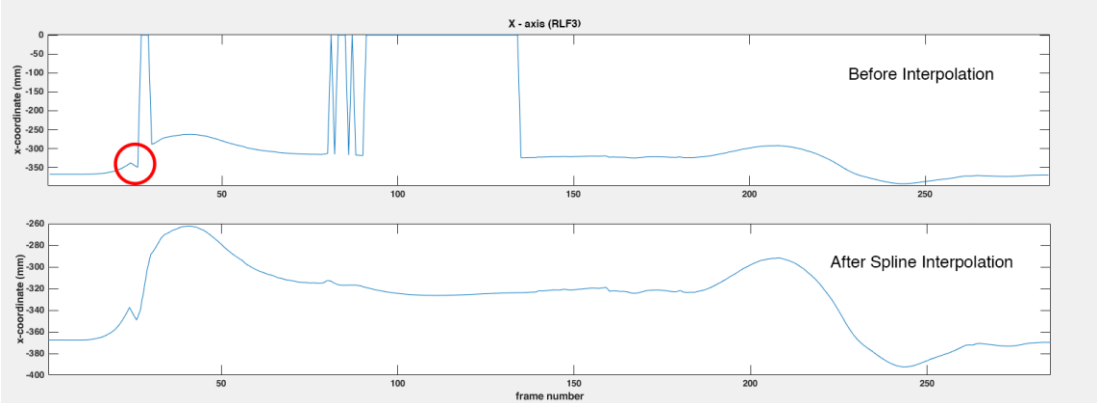

Figure 25 Interpolating gap on the plot with an artifact

3.2.4.2. Nexus

Using built-in feature dedicated to gap removal we can constantly keep an eye on the quality, even though that seems like a longer approach. It has three main methods, exact specification of which Vicon does not disclose. Example of graph correction withing Nexus seen in the *figure 26*. Corrections are also previewed in real time in 3D marker space.

- Woltring Quintic Spline
- Pattern – filling gaps using a donor trajectory
- Rigid Body – using relations between markers withing a segment

Once pipeline is set shortest gaps can be field almost 1-click for each marker using spline method. Same as in Matlab those correction do not need a lot of attention and work flawlessly almost every time.

Method for larger gaps depends on location and movement. For example, knuckle markers are best filled with a Rigid body between Hand and following knuckle markers. Or Little and Rear fingers can be a good Pattern source for each other, especially when not a lot of movement involved. This stage sometimes requires going back and forth with fillings, testing, and finding best match for Pattern or Rigid body until markers in 3D space behave in a natural way.

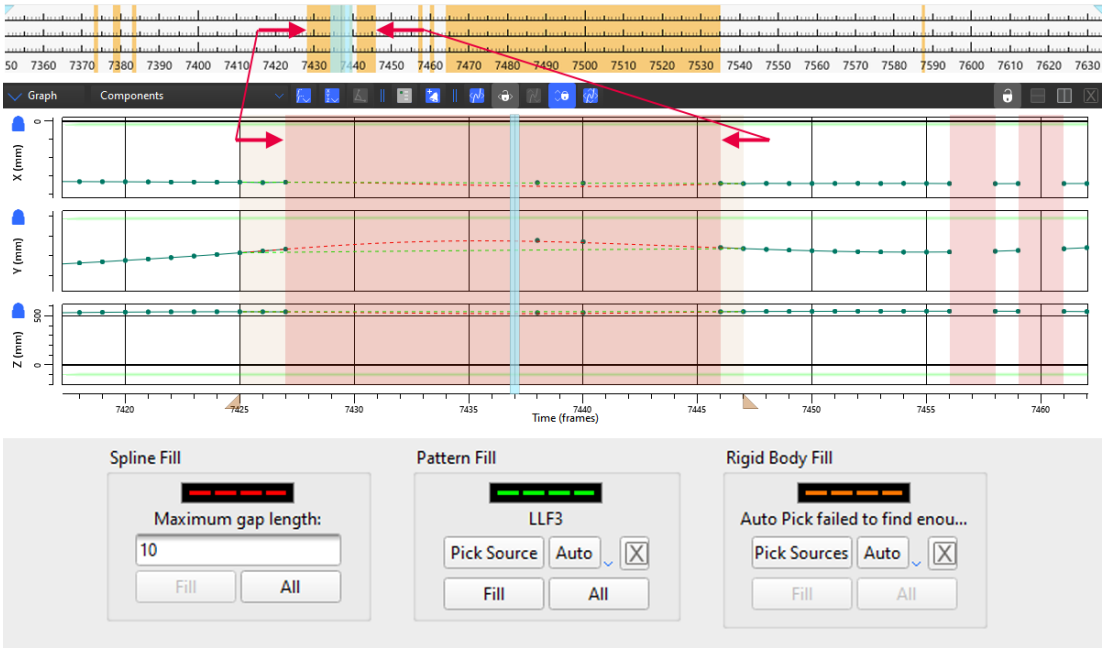

Figure 26 Automatic gap interpolation using Nexus: Spline (red) vs Pattern fill (green)

This concludes stage of gap interpolation. Markers are all set in place and move just like actor performed it. Yet some spike artifacts are still left, and overall performance has jitter. Removing those is explored in next sub-chapter.

3.2.5. Denoising

Working with optical mocap data we encounter two types of noise that needs to be filtered.

- *High frequency jitter*
  Common problem for most optical mocap system caused by either imperfect lighting or small amount digital noise on camera sensor that adds up during triangulation of 3D position.
- *Spikes*
  Rarely encountered problem caused by miscalculation of 3D position (usually given by marker being hidden on one of the axes). As illustrated on the *figure 26*.

Just like previous step denoising and smoothening can be performed via two different approaches.

3.2.5.1. Nexus

Denoising using Nexus features is simple and straightforward. Pick a track that needs to be cleaned. Choose one of three available methods:

- *Butterworth* (high/low pass of 3rd or 4th order are available)
- *VCM Spline* (no specifications provided)
- *Woltring* (Quintic Spline in GCV or MSE modes)

Then set a cut-off frequency. This is dictated by character of data, and for Sing Language typical jitter cut-off is about 6-10 Hz.

However, after all of the gaps were properly removed and markers action in realistic fashion, procedure of removing noise does not affect performance in any visible manner. Therefore, automated scripted approach is more suitable and time efficient.

3.2.5.2. Matlab

First step is to remove spike artifacts, so they do not affect further smoothening computations. This is best performed using a Hampel filter. This filter removes outlying values using Hampel identifier. For each frame Hampel block of `H_len` length calculates a median value m_i and a standard deviation σ_i of *k* frames before and after a current frame x_i ($k = \frac{\text{H\_len} - 1}{2}$ from each side). If current value varies more then set threshold n_σ value of $\times m$ *(1 is sufficient)* sigma such that $|x_i - m_i| > n_\sigma \sigma_i$ then this value is changed for median.

In Matlab applied as follows:

```
tempM = Markers(:,n,m);
hampel_M = hampel(tempM,k,nsigma);
hampel_temp = hampel_M;
N = length(tempM);
    for n = 1:N
        if (abs(hampel_M(n) - tempM(n)) < 0.1)
            hampel_temp = x(n);
        end
    end
Markers(:,n,m) = hampel_temp;
```

With spikes removed (as it shown in the figure 27) we can proceed to the jitter noise.

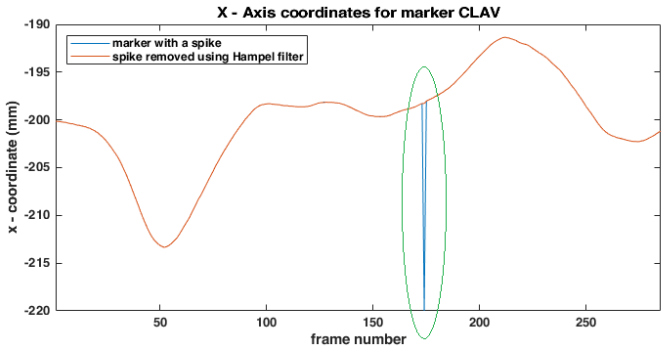

Figure 27 Example of a spike artifact and its removal

Two main options to remove jitters are Butterworth and Inverse Chebyshev filters. While results of both filters appear to be almost identical, Butterworth presents less computational complexity [19]. Where Chebyshev of 2nd order works withing stopband and Butterworth with high pass. Butterworth produces frequency response that is maximally flat in the passband and monotonic overall. Increasing order of the filter we are increasing steepness of the cut-off. For purposes of this project orders 3 to 5 can be used.

To apply a Butterworth filter in Matlab we first create filter response of a desired order, normalized cutoff frequency (set as `Wn = 2*cutoff/fs;`), and a type. Then extract zero-pole-gain filter parameters to transfer function form, and apply it to the signal.

```
M_in = Markers(:,n,m);
[z,p,k] = butter(b_order,Wn,'low');
[b,a] = zp2tf(z,p,k);
M_out = filtfilt(b,a,M_in);
```

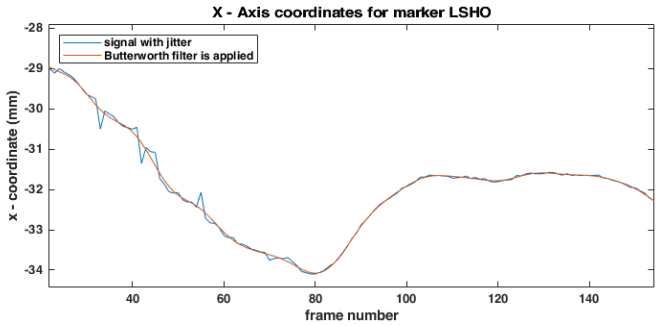

Figure 28 Marker signal denoised using Butterworth filter

As data was finally fixed and cleared it is almost ready to be applied onto 3D model all that is left is to correct human related data flaws mentioned at 3.1.1 and 3.1.2, that have been made during recording sessions.

3.2.6. Adding hip markers

Relatively easy but crucial procedure if we ever intend on applying this .c3d data onto 3D avatar. Animation building software requires pelvis markers to be set as a zero position while retargeting motion capture data onto a virtual actor. Without those markers model simply would not work, and there is no way to avoid that.

Luckily, we are working with a character that is not moving through space and stationary markers could be added in post. Activating an actor requires at least 3 markers from Plug-in-Gate set (figure 4): LASI, RASI, T10.

With a little bit of trial-and-error and finesse 3D coordinates for markers were selected approximately where those would sit on a real actor. And now those markers can be added using Matlab (extending original m-struct) or better with a help of C3Deditor software. In edit mode we pick New > 3D Marker. That creates an empty marker with X Y Z positions at 0. First 5 to 10 frames can be filled with desirable coordinates and file should be saved.

As a last step we have to interpolate the rest of empty frames using Patter fill with chest marker CLAV as its source. Now those markers can naturally move with the rest of the body without causing model mesh unrealistically stretching.

3.2.7. Clipping floor

Clipping floor issue, as well as actor facing off axis, does not affect data performance. But having markers being on the right level and facing the right direction makes is so much easier to apply onto a virtual actor.

Whereas .c3d files do not allow direct access to the data, adjunct .xcp files containing camera and system information can be viewed as a regular txt file. And after that it is a trivial task of finding `POSITION` and `ORIENTATION` parameters and setting it for a desired offset. In our particular case only 700 units was added to each camera position in Z-axis.

3.3. Interpolation

Having dataset ready we can try to connect individual segments. Results presented in this part are for the signs for *stranger's mother* (mother + stranger's; in Czech SL: cizí matka).

First task at hand is to prepare connection points, meaning we have to determine start and end of each recorded sign.

First idea of an automatic approach was to establish a Signing Space (SS) around a virtual actor as some sort of a cartesian box of coordinates. Then to track hand markers (RHAND, LHAND) and observe if those leave a Signing Space boundaries (`if` $|x_{hand}, y_{hand}, z_{hand}| > |x_{SS}, y_{SS}, z_{SS}|$).

But, not only, this approach works only from the assumption that all of the data was captured with a specific goal of creating a sign language database where every sign is starting and ending in some zero-position. But also, it does not actually determine a boundary of a sign.

To tell more, while working with this data set no specific tell of a pattern on when sign begins, or ends were spotted. And since it cannot be spotted with a naked eye for not automatic approach cannot be applied. Approximation of frames signifying end or finish can be found in attachment 1, listed next to corresponding sign.

Segments are going to be prepared by carefully studying naturally recorded sing combination and sliced at approximate frame. In this project this is done within Nexus software, modifying export parameters to include only necessary set of frames.

One regularity, however, was noticed. With a very rare exception, most movement in between two signs takes under 30 frames and travel around 10cm in real 3D space. Not only these are even shorter gaps that we were correcting using simple methods in sub-chapter *3.2.4. Gap Interpolation*. But also, giving that data was recorded using 100Hz system, motion under 30 frames would be performed in under third of a second. Which gives us a big hope for *1D approach* being just simple yet sufficient enough to satisfy project at this stage.

For the purpose of this work number of necessary frames can be approximates from distances between markers. Distance, of course is calculated using Pythagoras:

$$r(x,y,z) = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \qquad (3,1)$$

For this dataset (actor size and signs specifics) it was approximated to have 30 frames gap per travel of $r = 200$ units.

3.3.1. Back to the Nexus.

As built-in features already proved to be useful, it worth a while to test how they perform in sign concatenating.

Unfortunately, there is no option to merge segments withing software itself, so here is a fast work around.

First, we need to compensate for an error caused by Nexus exporting truncated data preserving original frame position. Open exported files using *Mokka* tool, mentioned in 2.1.1.1. Find *Acquisition options* and pick *Reframe from one*. It was discovered through multiple frustrating trial and error cycles, but without this step data will simply not work within some of the software. There is no particular explanation, but presumably first frame of a sequence has to be labeled as #1 and not by the label it had as a part of complete recording.

With that out of the way, precut ant reframed .c3d file can be opened in C3Deditor where first we add empty frames at the end of the sequence and then second sign can be appended at the end of the sequence as well.

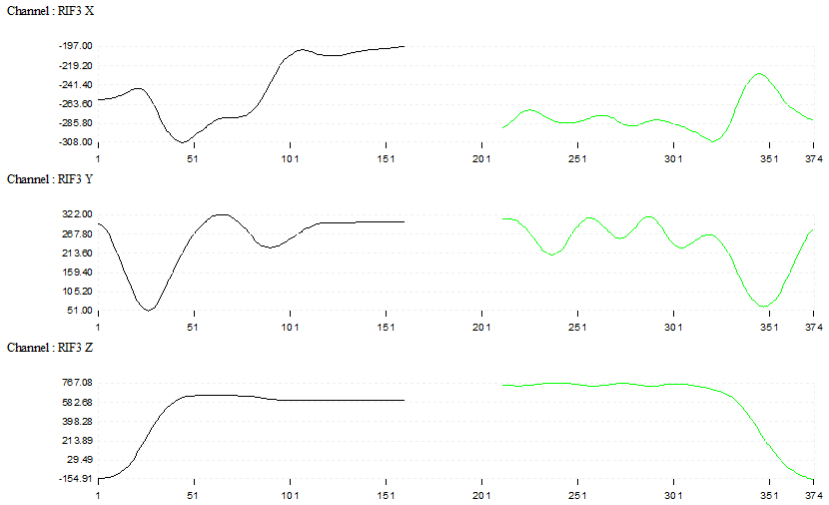

Figure 29 Two signs merged before interpolation.

Since Nexus does not have an import feature for its own .c3d format, before experimenting with interpolations newly created file just hast to be placed in a same folder as .xcp and other system files.

Outputs of all tested methods will be presented later alongside with other.

Using Matlab, however, is a more time efficient approach. Where both files are loaded at once by a function `readc3d.m`. Then length of a necessary gap is either calculated from a distance between last frame of a first sign and first frame of a last, or simply pre-determined from the knowledge of a dataset. And then stitched together with a zero vector in the middle, same way you would any other vector/matrix:

```
[first_word ; zeros(gap_length) ; second_word];
```

Process of interpolation itself mostly uses built-in Matlab functions, and is very analogous to the process describes in *Gap Interpolation* sub-chapter *3.2.4.1.*

Different .m files can be found in *attachment #2,* where most of the used methods have its own main.m directory.

To test linear, cubic, and spline methods same program as in Gap Interpolation was used. Zero values are isolated creating a singular vector for each marker, that is later is being run through the `interp1` function with a specified respective method of interpolation.

There is also a separate file for a spline function interpolating using precalculated arch from control points. This version also processes three axes together.

Script for the Catmull-Rom method in a same fashion loads 2 sign segments and merges them. Interpolation itself uses a slightly modified version of a function to calculate Cardinal Spline from Dr. Murtaza Khan, which was found as an open source.

Akima script is based of `makima.m` function and is being realized in a similar fashion as spline.

3.3.2. *Research of a 3D approach*

Methods described in a part *2.4.2.* regardless angular approach are commonly used in modern engines for 3D content creation, like for example a *Unity game engine.* Therefore, Euler and Quaternion interpolations can be tested directly inside of an engine.

While researching this topic were found multiple modern studies comparing these exact methods.

An extensive comparison was executed in [20] regardless the performance of three methods: *Lerp*, *Slerp*, and *Squad*. Broad range of upper body movements was animated and shown to a broad test group. It was found that animation generated by Squad interpolation was perceived as significantly more natural than that generated by Lerp or Slerp, scoring 3.5/5 on naturalness (higher is better). This conclusion holds not only for audiences with different levels of expertise in animation (novice or professional) and different gender groups (male or female), but also for different gesture types.

It also was stated that while lerp and slerp has radically different mathematical approach, any difference in performance is unperceivable, both reaching result of 2.5/5. Meaning, just like with our tested *Coordinates* approach higher continuity reaches better results.

Withing a testing group, however, only trained in 3D related sphere professionals assigned radically lower scores for linear methods. What can induce an assumption that for most of the customers it might suffice.

Unfortunately mentioned study does not go into comparing demand for compute power, but from a general knowledge of algorithms behind tested models, it is reasonably to guess demand scales with each method Lerp < Slerp < Squad. Creating a classical tradeoff dilemma between performance vs speed. Nonetheless, it is safe

to assume that being less demanding while performing the same – Lerp can be preferable method if we keep gimble lock in mind, of course.

More about computation time we can read from Avishkar Kolahalu on his GitHub post[4] dedicated to comparing three mentioned models. As predicted time to complete takes as follows 1 for Lerp < 6 for Slerp < 18 for Squad in normalized time units.

There we can also see animations of models in action. As stated by Avishkar:
although it is computationally more expensive, the smoothness of the motion is nearly identical to the input in most cases, and makes it worth the cost.

In study [21] researchers as well took on comparing Euclidian with Quaternion interpolations. Unfortunately, no animation was provided there, but on paper results further confirmed already mentioned outcomes.

[4] https://github.com/avikola/motion-capture-interpolation#1-input--bezier-euler

3.4. *Data implementation*

It is impossible to tell naturalness of data just by looking at infinite graphs depicting markers trajectories. So as a bare minimum we need to see it being implemented on a segmental skeleton created within Nexus. Outcomes are presented in a *Chapter 4: Results*.

Besides that, I would do my best to apply captured motion data onto 3D character. This procedure is done using Autodesk's Motionbuilder.

As a first step we need to either create or find a character model that satisfies parameters listed in *2.2.5.* Process was tested on two 3D models. Female character model provided by a faculty (CWom0013-M4). Female character found online as a royalty-free asset (Claudia).

After loading chosen character into Motionbuilder, in order to Characterize our 3D model, we first put it into a perfect T-pose using translation and rotation tools (in global mode). Once T-pose is ready, skeleton is needed (Character > Define > Skeleton > Biped). And now it is just a question of assigning 3d model parts to their respective bones as shown in the *figure 30.*

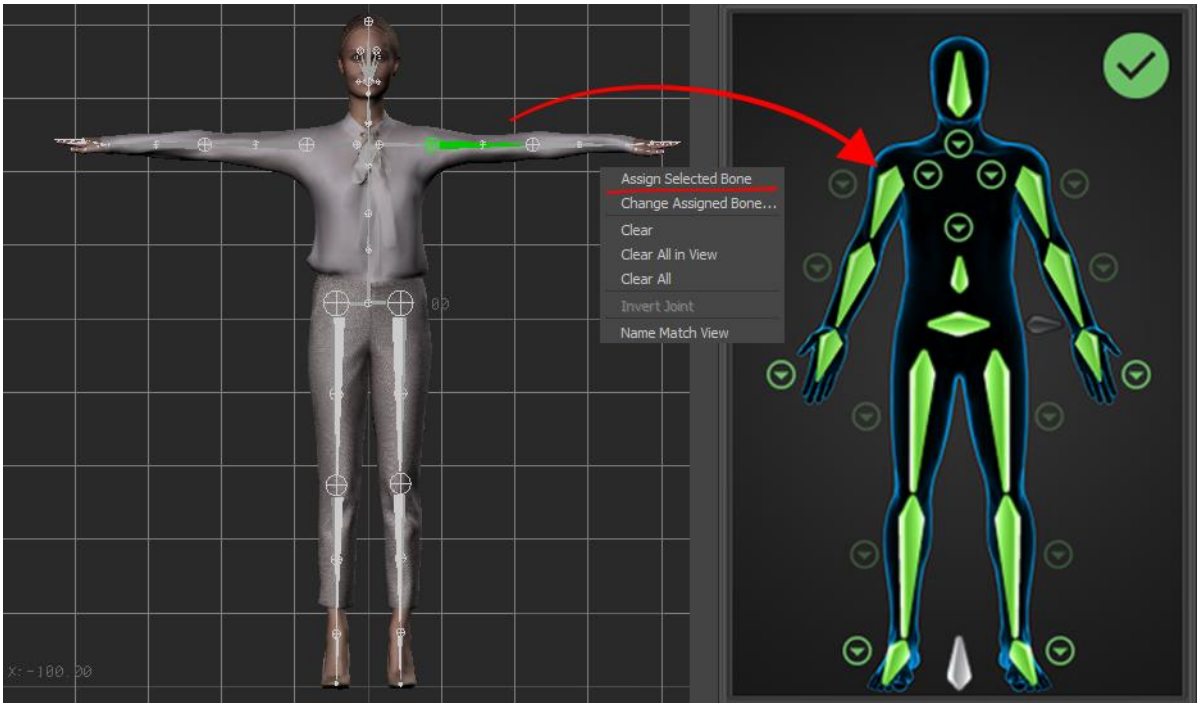

Figure 30 Character with a visible skeletoal structure being assigned to a virtual skeleton

If 3D model was created correctly and all assigned bones light up green without any warnings, software will be able automatically Characterize skeleton with a push of a corresponding button found on a Character Definition panel, and model can be saved as a .fbx file for the later use.

Now we can import motion data with our marker positions. C3D format can be opened directly by motionbuilder but does not import a structure we built. Therefore, it requires a pretty meticulous middle step process of rigging a special puppet tool called Actor. It needs to appear just as puppet would wear markers during recording session.

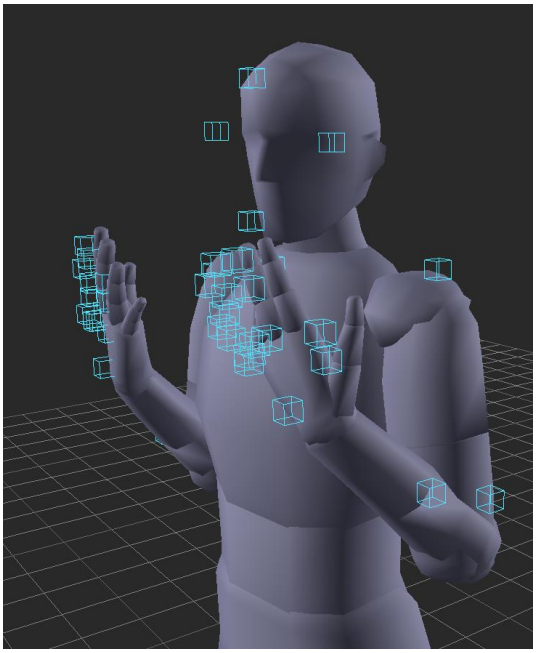

Figure 31 Actor puppet while assigning .c3d data

By default, actor puppet appears in a T-pose and needs to be shaped into motion data geometry. Here is where a lack of T-pose included in a dataset comes into relevance. Unconventional actors shape not only hard to achieve, but it may also cause inconsistencies during application.

With a help of a method described in *3.2.6.* and a lot of time on one's hands T-pose can be artificially created. Important part it to preserve distances between markers. Those distances are easy to measure using *(3,1).* Resulting number is a distance alongside X axis. This method, nonetheless, also leads to some distortions. Meaning each process has its tradeoff.

In presented results we managed to achieve satisfactory synchronization with a puppet using pose from calibration file. Example is seen in figure 31.

Blue cubes represent markers. Each marker needs to be added into newly created marker set and assigned to respective part of the body to be controlled. All that is left is to click activate. This is the point where hip markers created during 3.2.6. come into play. Without those markers actor will just assume a fetal position and will not be controlled by motion data.

As a very last step files containing motion driven actor and 3D character can be merged, and actor has to be selected as a motion source for the model.

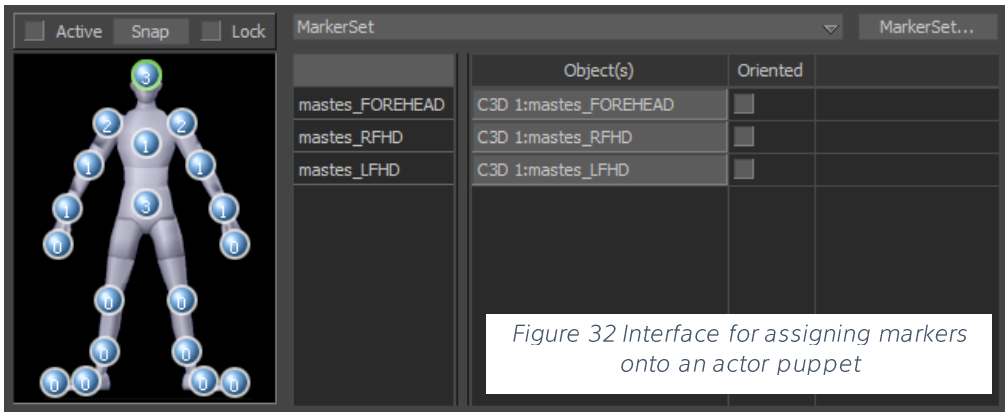

Figure 32 Interface for assigning markers onto an actor puppet

Chapter IV
Results

In the last chapter, we present visual results of the test and then verify if original hypothesis, that simple methods such a spline can produce sufficient enough results.

Unfortunately, 3D moving objects, especially in a form of marker sets, is not easy to present or particularly pleasant. Therefore, first we interpret the graphs. In the *figure 33* we can see an output from Matlab: same image from 3 different angles (right, middle, left). Graph shows recorded markers of an actor in the idol pose. Red lines connecting markers represent skeletal structure. Magenta colored pyramid connects markers on the head.

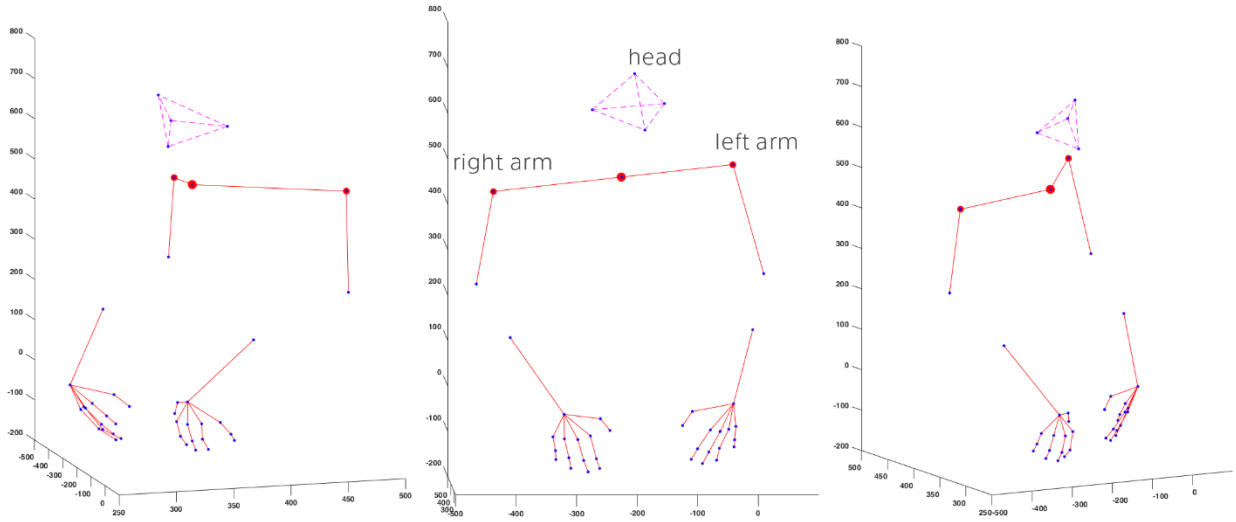

Figure 33 3D representation of a marker set

To distinguish moving parts, portion of a skeleton is hidden (*figure 34*). Arm movement is going to be represented by 2 frames: before and after. Just as it is seen in the next *figure 34.* Lower position represents frame #1 and higher one frame #150. Cyan dotted line shown movement through space for a particular marker. Attached recordings of a moving skeletal structure can help with explaining.

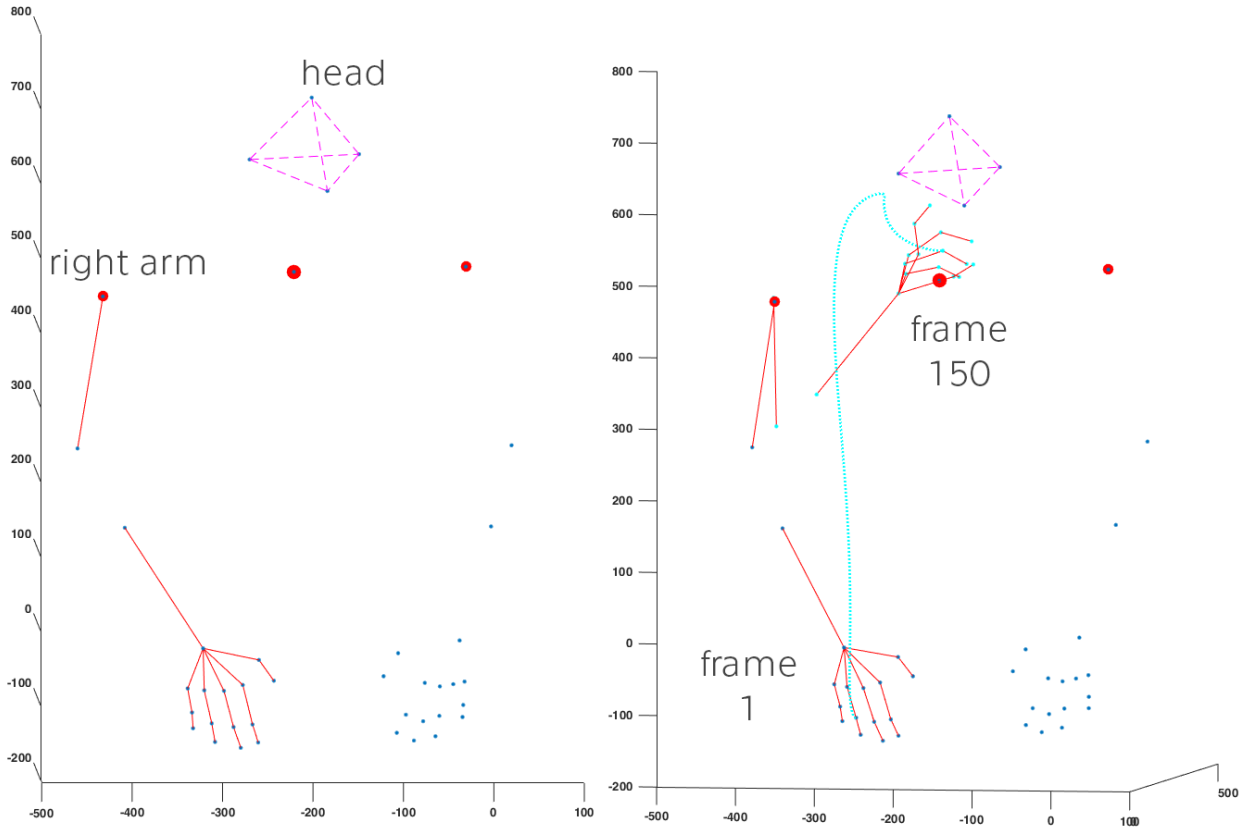

Figure 34 Right hand moving from frame 1 to frame 150

Knowing that we can proceed to view the results of an experiment.

Every graph presented here can also be found in *attachment #3* saved as a .fig file.

But since we are not comparing only graphs, every video output is contained withing *attachment #4*. Folder contains video of every method in 100% and ¼ of a speed. Two files comparing videos are included as well

4.1. Visual output

First, we should see a *figure 35*, how native sign movement looks like. It is presented by a very natural, long, arched curve. As seen on a video with a respective name individual signs are evidently readable, and connection is clear.

Following method is Linear (*figure 36*). This operation produced the least convincing result, for obvious reasons. What is not apparent from graph, however, movement looks unnatural not only because of the strict right angles, but also because of unnaturally even speed of movement

Next up is Cubical spline. It does appear in the *figure 37* to be similar to simple linear function, but it is mostly caused by length of the gap between symbols. On video it appears as much more natural. Still having a problem of moving on a straight line, but preserving a somewhat natural speed, thanks to slightly bend curve.

Same problem as with cubical arises withing different methods as well. Spline1 (*figure 38*), Nexus (*figure 39*), and Extra (*figure* 40) splines appears to look linear in a graphical form due to the short length of a gap, but do appear as somewhat natural on a video form.

Nexus here obviously referring to a built-in interpolation within Vicon's software. Movement looks pretty smooth, however, Nexus take a lot of creative liberties with the algorithm and interpolated signal loses some of the micromovements. Nevertheless, that does not affect the final result, as part of micro movement is visible only in a marker form and disappears while being stretched on a 3D model.

Graph labeled as Extra (*figure 40*) presents build in interpolation process from .c3d software developer [9]. They do not disclose anything about the algorithm, but by its performance it is safe to guess that this app applies cubical spline. Results are very similar and while being a bit straightforward it preserved a natural speed.

Hypothesis, that akima spline is going to perform the best has not been confirmed. As seen in a *figure 41*, connections, that are being produced are way to steep, resulting in an angular movement. That is being supported by the video as well.

Presented in a *figure 42* Catmull-Rom function looks almost the same as other splines on the graph. On video however it is apparent that it is not suitable, which should have been apparent from a description, calling this function, best perform with evenly spaced data.

As a last option, shown in a *figure 43* Spline with a precalculated arch performs the best. It produces the closes to the original curve. This version is best looking in a video form as well. However, algorithm is causing minor jitters before and after a connection point, which can be finely tuned with a denoising procedure, for example.

Presenting all of the graphs on a same picture is unfortunately impossible because every take, that is recorded using mocap has different coordinates, as actor moves through the scene in between takes.

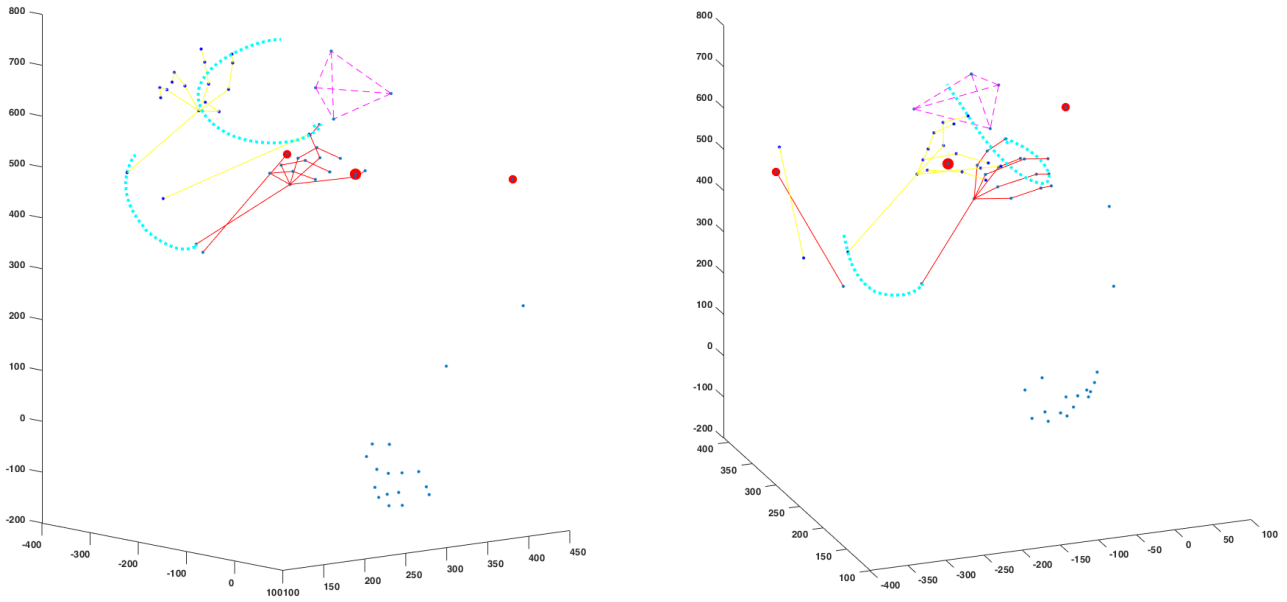

Figure 35 Original signs performed by an actor

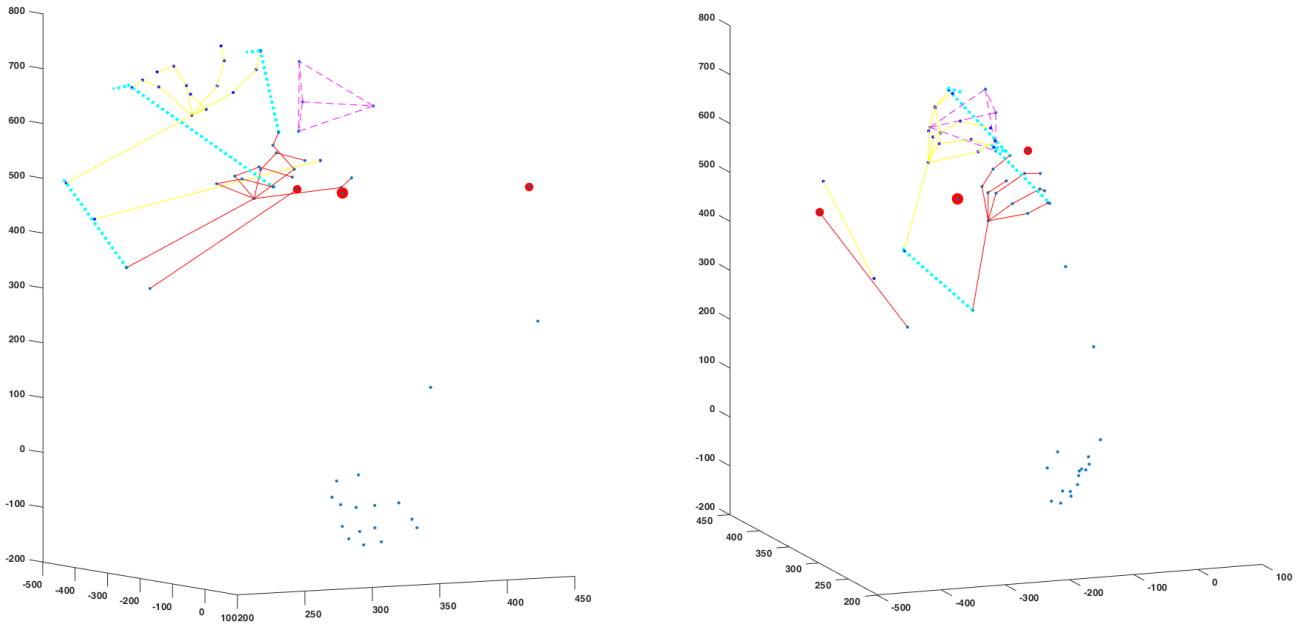

Figure 36 Signs connected using linear interpolation

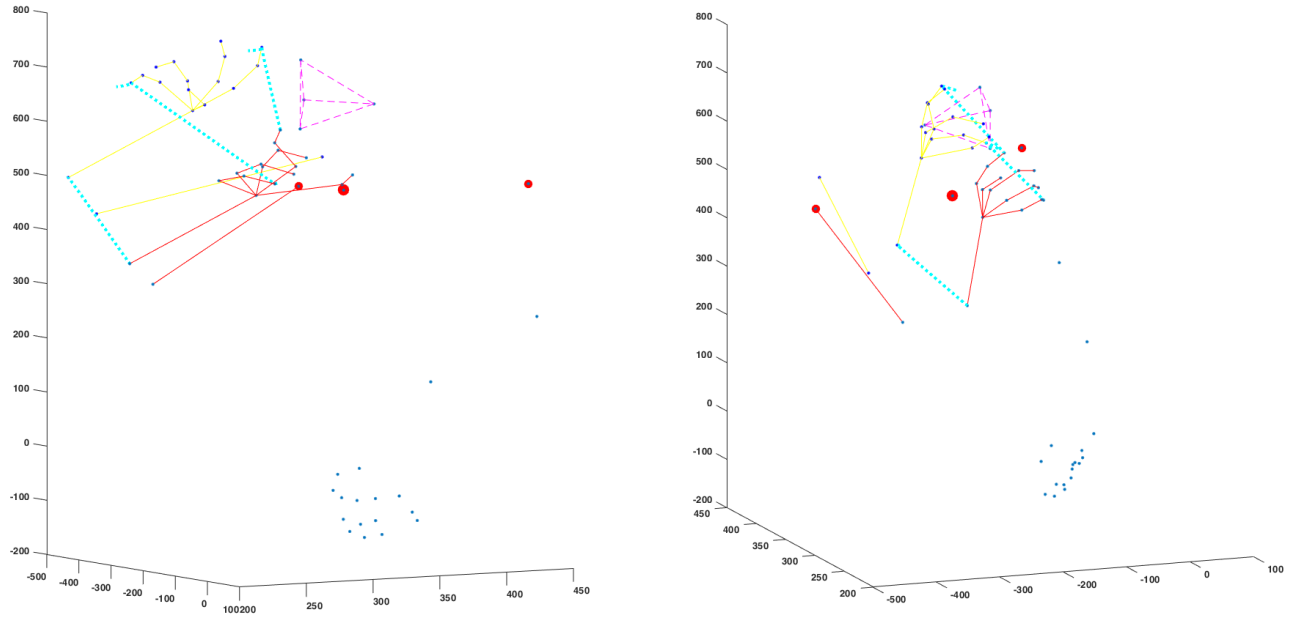

Figure 37 Signs connected using cubic interpolation

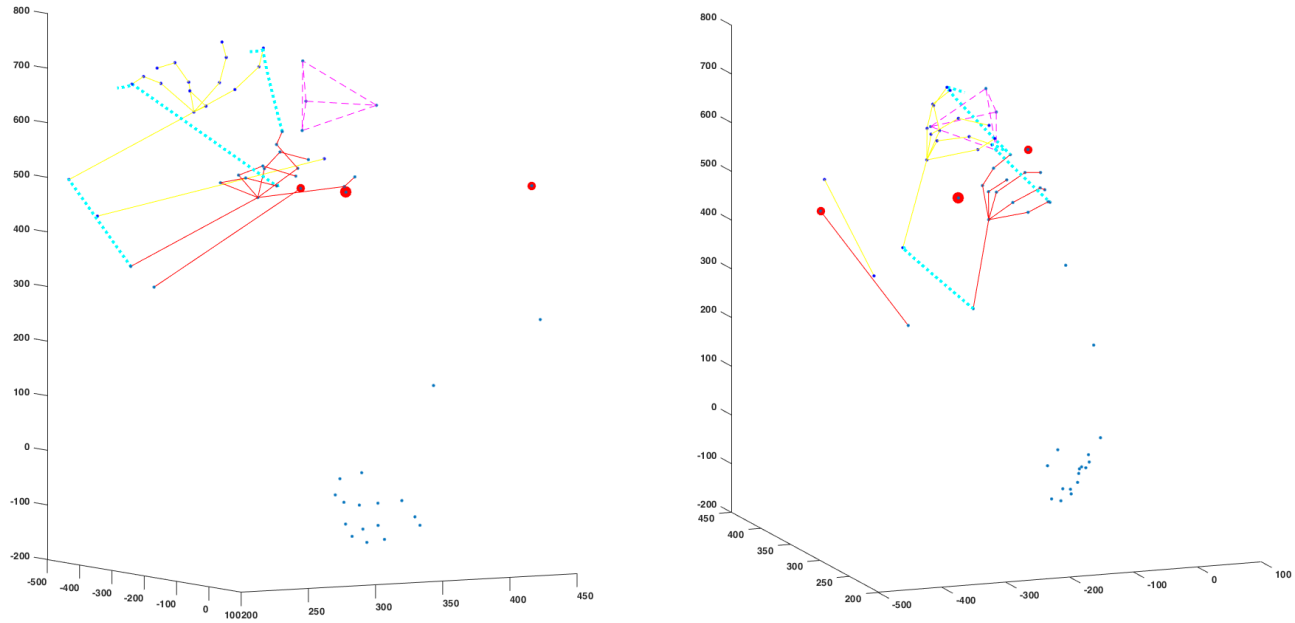

Figure 38 Signs connected using spline interpolation

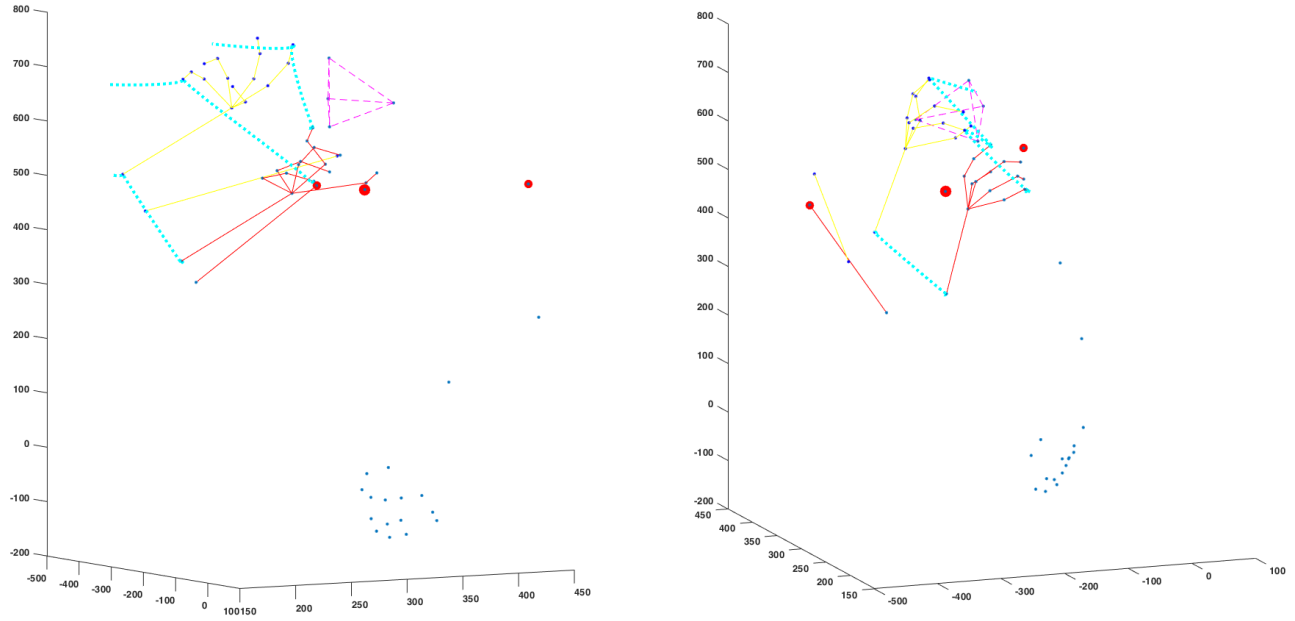

Figure 39 Signs connected using built-in nexus algorithm

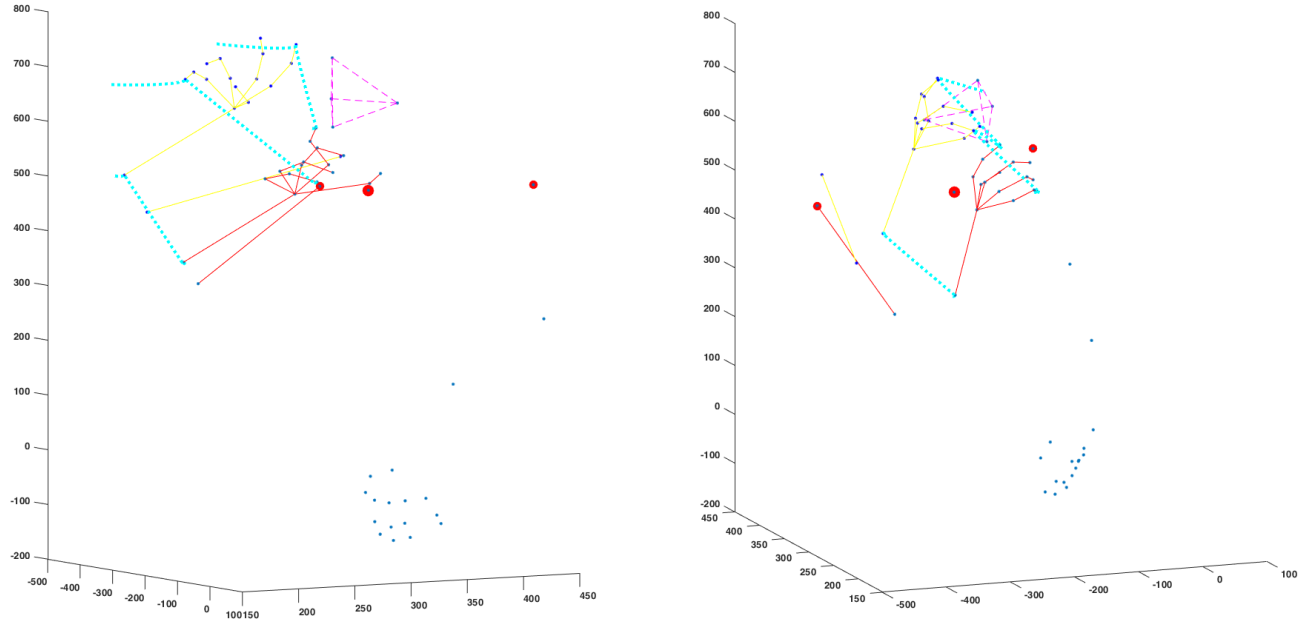

Figure 40 Signs connected using c3d algorithm

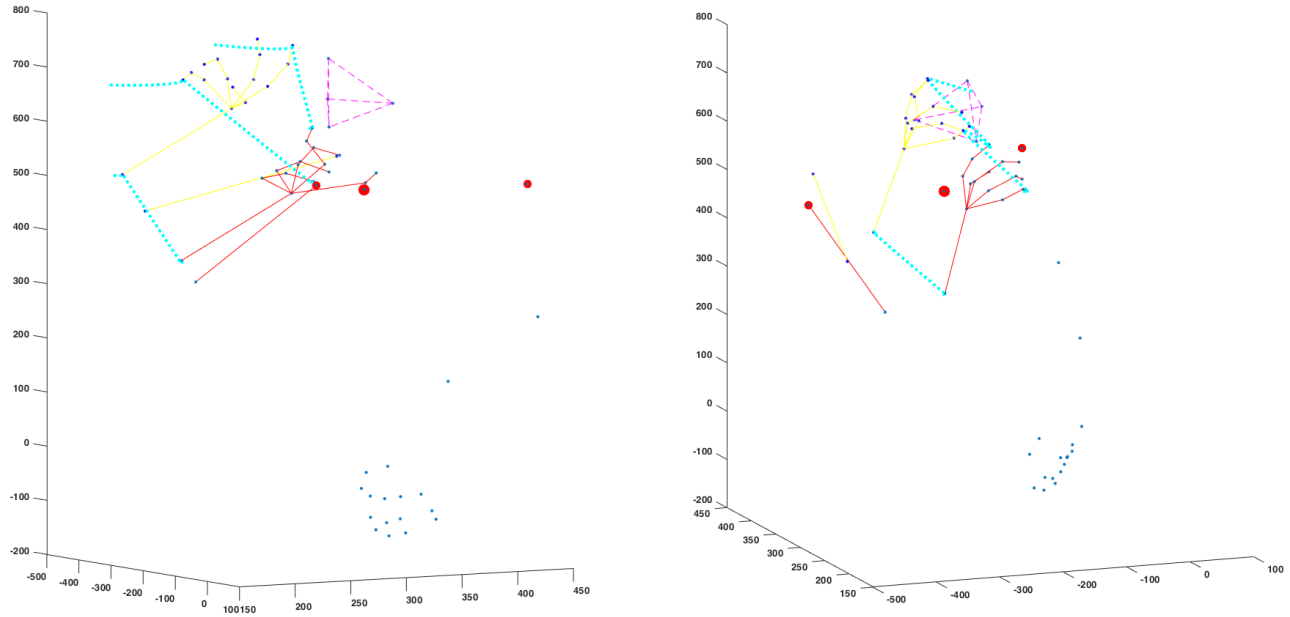

Figure 41 Signs connected using akima interpolation

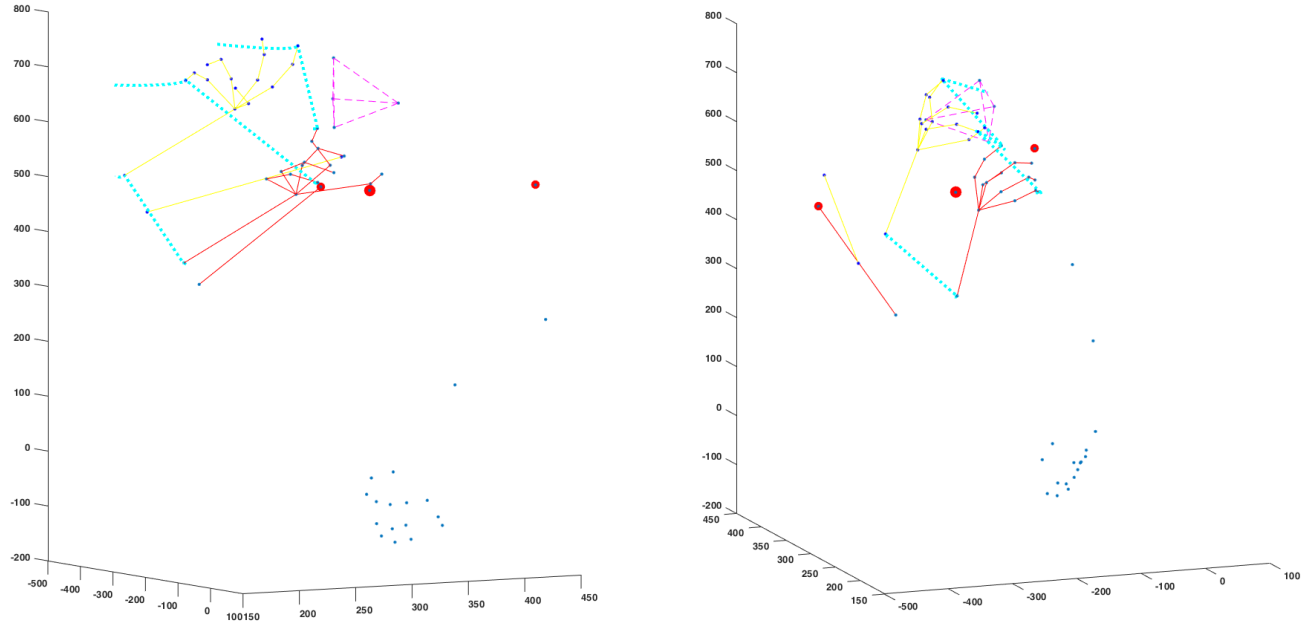

Figure 42 Signs connected using Catmull-Rom interpolation

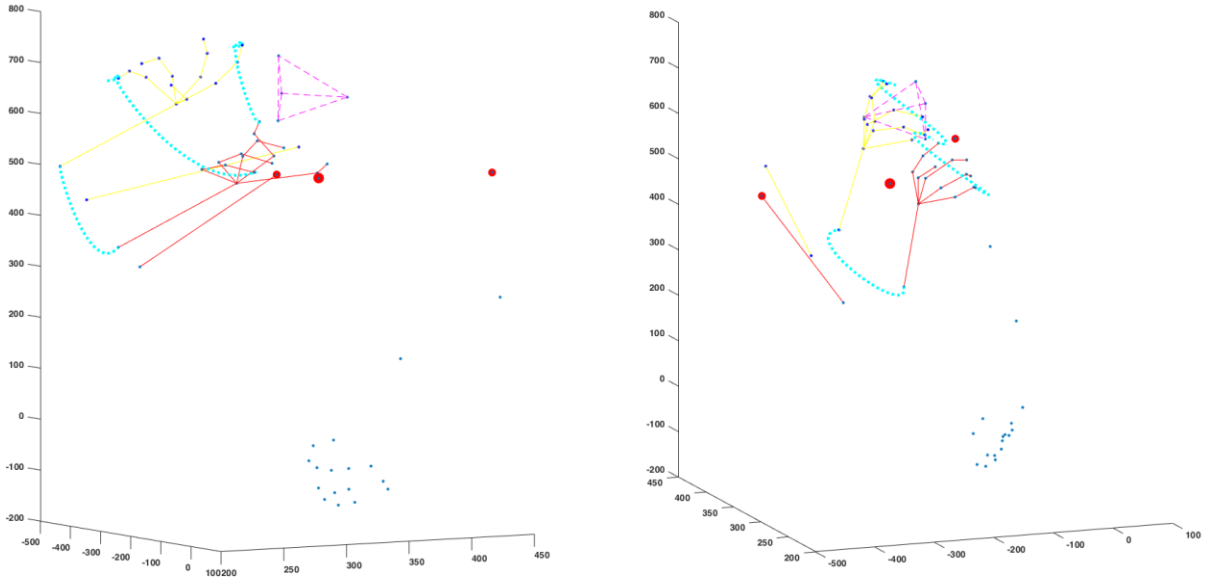

Figure 43 Signs connected using arched spline interpolation

Overall results from different methods looks almost identical. One of the exceptions is a linear interpolation, that was use just a base to compare with other procedures.

Another exception is an arched spline, that produced the most convincing result, especially given how simple the algorithm is.

Not only patter of a motion contributed to naturalness of a sign. Preserving speed of a potion has shown to be equally important. These results are better compared looking at the slowed down footage presented in attachment #4.

Knowing that these methods were tested with an assumption, that minor data irregularities would not be as noticeable when applied on a rigid 3D model, we can say that result of an experiment is somewhat satisfying.

It is true, that produced result does not look as something a native Sign Speaker would be pleased to see as an interpreter on a TV. Nevertheless, giving simplicity of computing, these methods can be used as a first stage of developing automized Sign Language translator. Such a translator can possibly be used in places without a possibility of having an interpreter on site at all times, such as a post office or a bank, providing basic mean of communicating day-to-day needs.

It is fair to mention, that algorithms performed so similarly because of the length of a gap we were working with. But this can be considered a specific of a Sign Language animation. Gaps between individual signs are naturally short. Applying same methods onto more conventional type of animation that works with big motion and long intervals in between keyframes would not reach same results.

At a such an early stage there might be no need to overcomplicate computations to be exact and perfect. Knowing that simple datasets at first would most likely work with only Fixed sign, and avoid elements described in *2.1.* sub-chapter of this thesis.

On top of that, each method described here can be combined with a simple follow up procedure, like a local jitter removal for an arched spline. Or cutting into a sign earlier, producing rounder curve.

Additionally in *attachment #7* can be found video files with a mocap data applied onto a 3D character. There we can see each of the signs being performed on its own, then original performance from a Sign Language speaker, and then we compare two interpolations that stood out during previous assessments: Linear and Arched Spline. Even though model does not look perfect, and we can observe mesh warping, we can also clearly identify produced signs.

All that being said. The whole process from start of the recording, to a graphical output, need assistance and far away from being automatic. Yet that approach still has potential to save time and funds if we compare it with real live interpreter or a computing machine that would be capable to run a full game engine to produce realistically looking sign interpolations.

4.2. Last word

Here are some thoughts regardless overall approach.

Optical motion capture presents a lot of challenges before we can even get to the stage of concatenating sign language. Mechanical suit can be a good solution for a better quality, but a costly one. Combined system can be the answer. For example, full body motion capture with mechanical gloves to record finger movement. Fingers being hardest to process from the optical data, while carrying the most information, definitely need a special approach.

Most of the research mentioned in this thesis to have decent and natural looking results works with motion capture data in a 'skeleton' format. Such a structure is for example supported in .bvh format. Animation creation also does always start with building a skeleton, so it only makes sense to try and change direction. BVH seems to be more suitable for transferring rotations as well. Having a system that controls a character from the inside sounds more reasonable than system of marker. Markers rather sit on top of the skin puppeteering model like a marionet. Rigid body defined within a skeletal structure also has potential to be more precise.

It is understandable, that there is no optical motion capture system that record data in such a format. And there is no direct way to convert one format into another, but handmade approach is possible. Using software like blender, for example. Writing an algorithm for blender to automize conversion might be worth considering.

Chapter V
Conclusion

Main goal of this thesis was to explore the process of creating a dataset of Sign Language elements to start working on an automatic translator that can simulate live interpreters; and then investigate algorithms helpful in creating new connections withing this dataset.

This work extensively covers the entire procedure from the choice of signs, to recording process, to practical application onto a virtual 3D avatar.

Most importantly, practical part describes the use of standard interpolating algorithms and shows their efficiency in a visual format. Summarizing achieved results, we can say that most of the listed methods perform similarly. Exception was a spline interpolation, when used with additional arch curve placed in between interpolated segments. Outcome of this method is surprisingly convincing, giving how easy it is to implement. It does not produce a perfectly natural result, but with minimal tweaks it can be used as a first stage of developing automized Sign Language translator. Such a translator can possibly be used in places without a possibility of having an interpreter on site at all times, such as a post office or a bank. Results can be found in attachment folder in a video format.

Besides practical part, thesis covers research of more sophisticated algorithms of animation interpolation, mainly used in game engines or professional 3D software.

In this thesis we also went over some drawbacks of a motion capture system as a whole. And discussed if different approaches are possible. Main concern was number of errors introduced into an information sensitive data. Current error correction methods were explored.

Overall results are satisfying, mostly proving proposed hypotheses.

Farther, these paper tries to navigate future students on the way of improving project of Sing Language synthesis, and its practical part can be used as a guide on how to work with sign language dataset.

Bibliography

[1] Rose, S. (2006). Monitoring progress of students who are deaf or hard of hearing. National Center on Student Progress Monitoring

[2] Nespor, M. & Sandler, W. (1999). Prosody in Israeli sign language. Language & Speech 42, (Cross-reference from [22]) (pp. 143–176).

[3] Woll (2001). The Hands are the Head of the Mouth, chapter The sign that dares to speak its name: Echo phonology in British Sign Language (BSL), (pp. 87–98).

[4] Gutemberg, G. F. (2005). Optical Motion Capture: Theory and Implementation. At Computer Vision Laboratory, Center for Automation Research. University of Maryland (pp. 1-2)

[5] At Next Generation. Imagine Media (1995). Motion Capture: Optical Systems

[6] At Next Generation. Imagine Media (1995). Motion Capture: Magnetic Systems

[7] Tanveer, S. (2014). Motion Capture Technology. At SlideShare, seminar report 33462669 (pp. 12-15)

[8] Xsens 3D motion tracking. A history of motion capture. https://www.xsens.com/

[9] C3D developer page. https://www.c3d.org/index.html

[10] Fläckman, J. Roos, J. (2006). Animations Anpassad mjukvarupieline för Motion Capture leverantörer.

[11] Nedel, L. P. (1998). Simulating Virtual Humans. Universidade Federal do Rio Grande do Sul (UFRGS). École Polytechnique Fédérale de Lausanne (EPFL). (pp. 10-20)

[12] Baerlocher, P. (2001). Inverse Kinematics Techniques for the Interactive Posture Control of Articulated Figures. At Département D'Informatique et de Recherche Opérationnelle Fédérale de Lausanne (pp. 40-41)

[13] Božić, P. (2016). Plug in-gait marker placement. At SlideShare (pp. 1-4)

[14] Hiroshi, A. (1970). "A new method of interpolation and smooth curve fitting based on local procedures." Journal of the ACM (JACM) (pp. 589-602)

[15] Dam, E. & Koch, M. & Lillholm, M. (1998). Quaternions, Interpolation and Animation. At Department of Computer Science University of Copenhagen, Technical Report DIKU-TR-98/5 (pp. 5-33,38-39, 42-56, 77-84)

[16] Riaola, G. (2019). Towards an Intuitive and Iterative 6D Virtual Guide Programming Framework for Assisted Human-Robot Comanipulation (pp. 10-17)

[17] Smołka, J. & Skublewska-Paszowska, M. (2014). Comparison of interpolation methods based on real human motion data. At Lublin University of Technology, Electrical Engineering and Computer Science Faculty. (pp. 226)

[18] Saeed, M. (2021). An Introduction To Recurrent Neural Networks And The Math That Powers Them (web article: https://machinelearningmastery.com/an-introduction-to-recurrent-neural-networks-and-the-math-that-powers-them/)

[19] Kubíček, A.(2018). Motion Capture Data Denoising. At Faculty of Electrical Engineering, Czech Technical University in Prague. (pp. 34-37)

[20] Lei, X. (2020). A Comparation of Interpolation Methods for Virtual Character Upper Body Animation. At Department of Computer Graphics Technology, Faculty of Purdue University (pp. 33-36, 48-49)

[21] Haarbach, A. & Birgal, T. & Ilic, S. (2018). Survey of Higher Order Rigid Body Motion Interpolation Methods for Keyframe Animation and Continuous-Time Trajectory Estimation. At 2018 International Conference on 3D Vision (3DV) E-ISBN:978-1-5386-8425-2 (pp. 7-8)

[22] Zatloukal, P. (2014). Motion Data Processing in Sign Language Synthesis. At Faculty of Electrical Engineering, Czech Technical University in Prague. (Cross-referencing [1], [2], [3]) (pp. 7-8, 10-11, 34)

[23] Sukhomlina, K. (2014). Humanoid Motion Capture. At Faculty of Electrical Engineering, Czech Technical University in Prague. (pp. 16-18)

[24] Gomez, D. & Guimarães, V. & Silva, J. (2021). A Fully Automatic Gap Filling Approach for Motion Capture Trajectories. At Faculty of Engineering, University of Porto

[25] Skurowski, P. & Pawlyta, M. (2021) Gap Reconstruction in Optical Motion Capture Sequences Using Neural Networks. At Department of Graphics, Computer Vision and Digital Systems, Faculty of Automatic Control, Electronics and Computer Science, Silesian University of Technology

[26] Zhou, C. & Lai, Z. & Wang, S. & Li, L. & Sun, X. (2021). Learning a deep motion interpolation network for human skeleton animations. At Virtual Human Group, Netease Fuxi AI Lab. Special issue paper John Wiley & Sons, Lt

List of attachments

#1 Spreadsheet with a dataset recorded during this project. It includes list of individually recorded signs, as well as list of pre-recorded combinations. Each word has an assigned frame from start and finish of the sign, so it can be located withing a recording.

#2 Folder with Matlab script, necessary functions and .c3d containers of sign language segments that were connected.

#3 Folder with graphs and Matlab figures in .fig.

#4 Videos showing results of an experiment. 2 signs separately, 1 original performance and interpolated connections with respective names. Each video comes in 100% and 25% of a speed.

#5 Folder with Matlab function to work with C3Dserver, compiled by users. Could also be found on C3D developers web [9].

#6 Recorded database in original format ready do use in Nexus.

#7 Video output of real and interpolated data being applied onto a virtual 3D avatar.