This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement n: 761999

EasyTV: Easing the access of Europeans with disabilities to converging media and content.

D3.7 Sign language capturing technology final version

EasyTV Project

H2020. ICT-19-2017 Media and content convergence. – IA Innovation action.

Grant Agreement n°: 761999

Start date of project: 1 Oct. 2017

Duration: 30 months

Document. ref.:

Disclaimer

The EasyTV Consortium consists of the following partners:

	Partner Name	Short name	Country
1	Universidad Politécnica de Madrid	UPM	ES
2	Engineering Ingegneria Informatica S.P.A.	ENG	IT
3	Centre for Research and Technology Hellas/Information Technologies Institute	CERTH	GR
4	Mediavoice SRL	MV	IT
5	Universitat Autònoma Barcelona	UAB	ES
6	Corporació Catalana de Mitjans Audiovisuals SA	CCMA	ES
7	ARX.NET SA	ARX	GR
8	Fundación Confederación Nacional Sordos España para la supresión de barreras de comunicación	FCNSE	ES
9	Unione Italiana dei Ciechi e degli Ipovedenti	UICI	IT

PROGRAMME NAME:	H2020. ICT-19-2017 Media and content convergence. convergence. – IA Innovation action
PROJECT NUMBER:	761999
PROJECT TITLE:	EASYTV
RESPONSIBLE UNIT:	CERTH
INVOLVED UNITS:	CERTH, UPM
DOCUMENT NUMBER:	D3.7
DOCUMENT TITLE:	Sign language capturing technology final version
WORK-PACKAGE:	WP3
DELIVERABLE TYPE:	R
CONTRACTUAL DATE OF DELIVERY:	30-09-2019
LAST UPDATE:	30/09/2019
DISTRIBUTION LEVEL:	PU

Distribution level:

PU = *Public,*

RE = *Restricted to a group of the specified Consortium,*

PP = *Restricted to other program participants (including Commission Services),*

CO = *Confidential, only for members of the LASIE Consortium (including the Commission Services)*

Document History

VERSION	DATE	STATUS	AUTHORS, REVIEWER	DESCRIPTION
v.0.1	24/09/2019	Draft	Kiriakos Stefanidis, Kosmas Dimitropoulos, Petro Daras (CERTH)	
v.0.2	27/09/2019	Review	Nicolamaria Manes (MV)	Partner review
v.0.3	30/09/2019	Final	Kiriakos Stefanidis, Kosmas Dimitropoulos, Petro Daras (CERTH)	Final version after review

Definitions, Acronyms and Abbreviations

ACRONYMS / ABBREVIATIONS	DESCRIPTION
API	Application Programming Interface
GUI	Graphical User Interface
CNN	Convolutional Neural Network
DOF	Degrees of freedom
DoA	Description of Action
FPS	Frames Per Second

Table of Contents

List of Figures

Executive Summary

This document is the final version of the deliverable (D3.7) concerning the development of the EasyTV capturing technology. This technology will be used for capturing the motion of the signers, i.e., persons speaking Sign Language. The main points of difference with the preliminary version of the deliverable are the following:

- Multi-lingual support of 5 languages (Spanish, Catalan, English, Greek, Italian).
- Replacement of Microsoft's Kinect sensor with Intel's RealSense depth sensor.
- Newest version of keypoint detection software with a new body skeleton model of 25 keypoints.
- Implementation of 3 data filtering techniques as an automatic post-processing step for enhanced motion refinement.
- Exporting C3D and BVH files.
- Support of Avatar VR Gloves for enhanced hand tracking accuracy (Optional).

1. INTRODUCTION

Motion capture systems are technologies that are used for capturing the motion of a person. The result of this process is data that encodes motion information and is used to animate digital character models in 2D or 3D computer animation. Such technologies have been successfully used in cinema, video games, sports and military training. Recently, there have also been some efforts to use motion capture for the playback of signing avatars.

The EasyTV capturing technology aims to provide accurate motion capture for Sign Language translation tasks. These tasks will be defined by the moderator of the EasyTV crowdsourcing platform (T5.2) and distributed to Sign Language experts in order to fulfill them. Since speaking Sign Language involves not only hand gestures but also facial expressions, the technology has to be capable of capturing all necessary information for any sign and thus include an accurate hand tracking technology, as well as, detectors for the face and body of the signer. The results exported by this module include text, video, and most importantly, motion files. This data will be uploaded to the EasyTV crowdsourcing platform.

The present document is organized as follows:

- **Chapter 2** presents the architecture of the capturing technology conforming to the EasyTV DoA.
- **Chapter 3** presents the changes in the capturing setup that properly fit the requirements of Sign Language.
- **Chapter 4** describes the detection phase where 2D keypoints are detected and 3D data is generated.
- **Chapter 5** gives an analysis of the refinement process where errors in the motion data are eliminated and values are restored.
- **Chapter 6** discusses the formats exported by the capturing module, as well as, their utilization by the crowdsourcing platform and other EasyTV services such as the crowdsourcing platform and the realistic 3D avatar.
- **Chapter 7** presents changes in the UI of the application.
- **Chapter 8** outlines the conclusions of this work.
- **Chapter 9** lists the references.

2. ARCHITECTURE

This chapter outlines the aspects concerning the architecture of the EasyTV capturing technology. These include the interconnection with other EasyTV services (i.e., external architecture), as well as, the internal components that make up the capturing technology (i.e., internal architecture).

2.1. Input & Output

The input and output of the capturing technology is defined in document D1.4 “Final release of the EasyTV system architecture” which describes the overall EasyTV architecture and the interconnections between its modules and services. According to this document, the input of the EasyTV capturing technology is given by an expert signer, that is, a person who has excellent knowledge of Sign Language and is capable of correctly signing in front of the capturing sensor. The input acquired by the capturing software concerns different types of image data (i.e., RGB-D) that capture the requested sign or signs, but also, text annotations that describe the meaning of each sign. As it will be described latter in this document, these two actions are performed in a sequential manner with the acquisition of the image data performed first and the user annotation following next. The visual content is processed by detection algorithms in order to extract the necessary information to compose the motion information of the signing.

The output of the capturing module is a collection of files that are uploaded to the crowdsourcing platform. The uploaded data will be stored properly into repositories and be available for use by other EasyTV services and components.

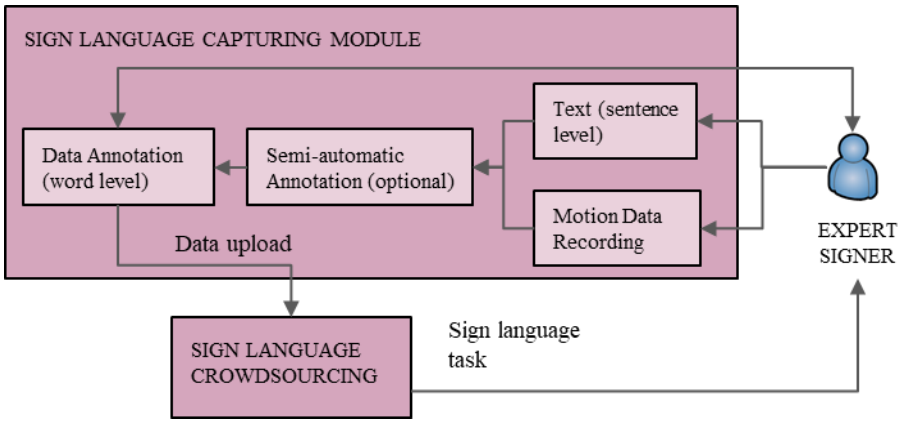


Figure 1: Schematic representation for the input and output of the capturing module.

A schematic representation for the input-output of the capturing module is given in Figure 1. In this figure, the crowdsourcing platform issues a task for the signer, the signer makes the requested signs and the capturing module records the images and extracts the motion data. As we can see, there are different types of annotation procedures: the expert signer initially annotates sentences but can also break the sentences into words in a later stage. Moreover, a semi-automatic annotation mechanism can be used for making the process easier for the signer by automatically recommending the right translation for a given sign. At the end of the procedure, the data is uploaded to the Sign Language crowdsourcing platform and stored into repositories. The crowdsourcing platform can generate additional tasks to the same signer and the whole process is then repeated.

2.2. The Phases of the Capturing Process

The EasyTV capturing technology consists of a number of phases that meet the requirements of the EasyTV project for Sign Language motion capture. These phases form a pipeline architecture

where the output of each phase is given as input to the next one. The architecture includes phases for the acquisition of images containing the signs, the annotation of each sign, the detection of keypoints for the hands, body and face, the generation of 3D data, the correction of motion data, and the formatting of motion files. A graphical representation of this interconnection between the phases of the capturing process is given in Figure 2.

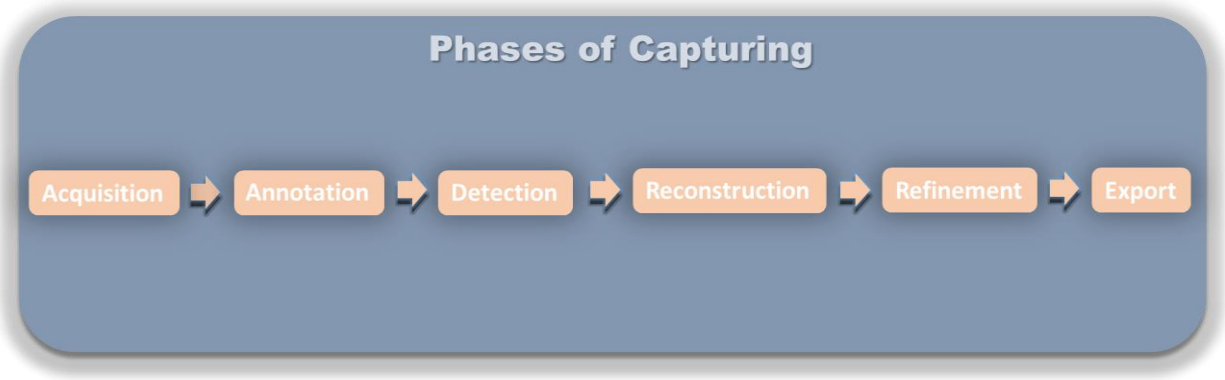


Figure 2: Pipeline diagram demonstrating the six phases implemented in the EasyTV capturing technology.

More specifically, a brief description for each phase is given below:

- **Acquisition:** The signer makes the requested signs and the capturing software acquires images from the sensor. There are two types of images captured by the sensor: those containing RGB information and others containing depth information. Both types though are necessary for the generation of 3D motion data. Also, color and depth videos can be generated in order to display the captured content to the user (e.g., admin or signer) in the annotation phase.
- **Annotation:** The admin or the signer reviews the captured content and annotates video. The annotation concerns the mapping of visual content to text (i.e., subtitles). This process can involve the annotation of the whole video with a single phrase or the trimming of the video, that is, the video is divided into segments with each segment representing a sign and annotated with text.
- **Detection:** Algorithms detect keypoints on frames. Keypoints are points of interest on the captured frames that identify the important visual content on the image. In our case, the important parts of the images are the hands, face and body of the signer. The keypoints detected can be either 2D or 3D keypoints. Currently, we use a detection procedure for 2D keypoints. The number of the keypoints and their position depends on the training of the detection algorithms. In the current state-of-the-art, such detection algorithms are usually deep neural networks that are trained on annotated images.
- **Reconstruction:** This is the method for generating 3D keypoints. Since our detected keypoints are two-dimensional, we also need to infer the depth information in order to compose 3D keypoints. The 3D keypoints are necessary for generating motion data that can be imported into the EasyTV realistic Sign Language avatar (T2.4).
- **Refinement:** Concerns a method for automatic error correction in the detection process. This phase is required due to noise in the detected data or misdetections.
- **Export:** Data that is exported in proper file formats for compatibility with other EasyTV services. The most important files exported are the motion files supported by 3D animation software and avatar technologies.

A more detailed analysis of each phase is given in the following chapters of this document.

3. NEW CAPTURING SETUP

This chapter provides a brief overview of the existing sensor technologies used for hand tracking, as well as, the overall setup that is considered to be suited for the purposes of EasyTV and the capturing process of Sign Language.

3.1. Integration with Intel's RealSense Depth Sensor

We have made the necessary modifications to the capturing module in order to replace the Microsoft Kinect sensor with Intel's RealSense depth sensor. More specifically, we use the Intel RealSense D435 sensor which offers the widest field of view of all other Intel cameras, along with a global shutter on the depth sensor that is ideal for fast moving applications. The depth camera is a stereo tracking solution that offers good quality depth estimation. Having a range up to 10m and a wide field of view, the depth sensor is very good for our motion capture application on Sign Language, where seeing as much of the scene as possible is vitally important in order to capture full body motion.

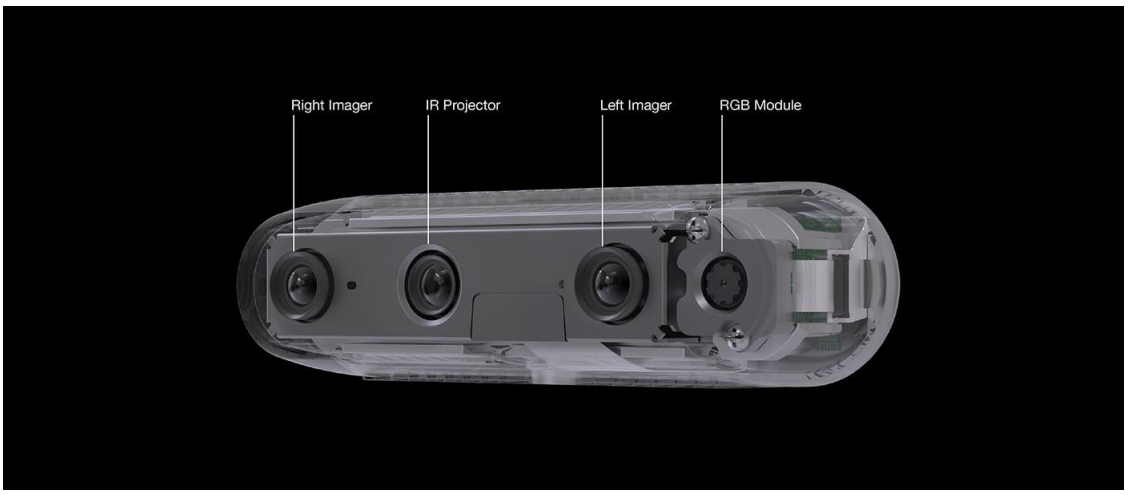


Figure 3: The Intel's RealSense D345 set of image sensors.

The RGB image is aligned with the depth image in a way similar to the Kinect sensor's coordinate mapper. We position the D435 sensor on a tripod and place it in front of the signer in a distance that allows for any extension of the hands to be visible, although not too far away so that the quality declines. An example of a capturing session using the RealSense sensor is illustrated in Figure 4.

Figure 4: Integrating and testing Intel's RealSense depth sensor.

3.2. Enhanced Hand Tracking with Avatar VR Gloves

In order to reduce the levels of noise when recording fingers, a new implementation with tracking gloves was introduced and integrated into the capturing module. More specifically, we can provide enhanced hand and finger tracking using the Avatar VR Gloves which are a top-notch upper-body tracking technology that is based on IMU sensors tracking the movements of arms, hands and fingers with pinpoint accuracy. The gloves have a built-in 600 mA Li-Po battery to ensure from 5 to 8 hours of continued use. Recharge is done by micro USB port. Finger Tracking is performed by using a set of IMU. Thumb uses two IMU in combination with a flex sensor, and one IMU for rest of the fingers. IMU technology is considered to be the best solution available, as it simulates full motion with all degrees of freedom. These gloves also support Bluetooth 4.0 Dual Mode with low latency custom firmware. An avatar playback using the gloves is presented in Figure 5.

Figure 5: Enhancing the robustness of finger motion on a 3D avatar using the Avatar VR gloves.

4. KEYPOINT DETECTION

This chapter provides an analysis of the methods that were used to detect keypoints on the acquired images. In the current state of the capturing technology, we decided to use 2D keypoint detectors on RGB images and then infer the depth values in a following step.

4.1. Keypoint Detection & Changes on the Skeleton Model

There are many definitions for keypoints in literature. In fact, keypoints are usually referred to as interest points within an image. They are spatial locations, or points in the image that define what is interesting or what stands out in the image [1]. In other contexts, though, they can be regarded as a set composed of (1) interest points, (2) corners, (3) edges or contours, and (4) larger features or regions such as blobs. The special characteristic about keypoints is that no matter how the image changes, i.e., whether the image rotates, shrinks/expands, is translated (i.e., if affine transformations are applied to the image) or is subject to distortion (i.e., a projective transformation or homography is applied to the image), one should be able to find the same keypoints in this modified image when comparing with the original image.

Keypoints are found by specific algorithms that process the image. Such types of algorithms that detect keypoints are called *detectors* and find application in a wide area of computer vision tasks, such as, image processing and analysis, object recognition and classification, and also, motion capturing. Some desirable properties of a keypoint detector are:

- Accurate localization.
- Invariance against shift, rotation, scale, brightness change.
- Robustness against noise, high repeatability.

The newest version of the body keypoint detector detects a set of 25 points instead of 18 included in the previous version. The new set of keypoints fits better the skeletons that are regularly used by

most avatars. The differences between the 2 skeleton models can be seen in the following figure.

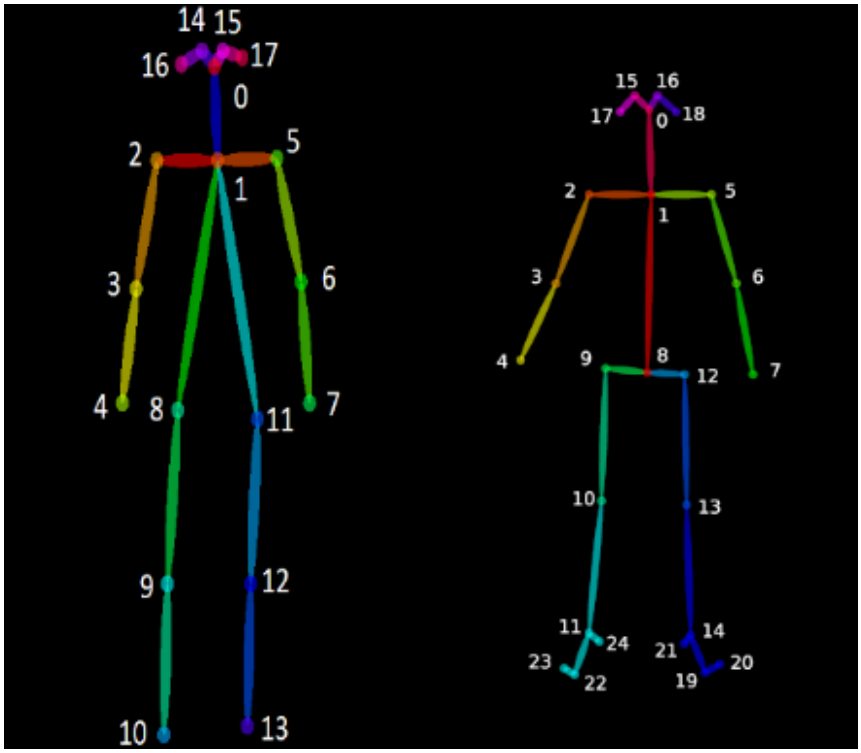


Figure 6: Differences between body detection models.

4.2. Detecting 2D Keypoints

Sign Language detection is a very demanding process requiring the motion capture of gestures, as well as, body movements and facial expressions. For that reason, we need to detect keypoints not only for the hands of the signer, but also for the body and the face. In order to accomplish that, three distinct keypoint detectors need to be integrated with the capturing module. In the current state of the module, we decided to use the *OpenPose* library [4] as our preferred solution for the implementation of the detection phase. Our research indicated that this is the state-of-the-art solution for unobtrusive hand detection, and hence, optimal for capturing the motion of Sign Language. *OpenPose* is also a practical fit for the EasyTV capturing technology because it only requires RGB images as input for detecting keypoints. Thus, the software can work with any camera available. Except for hand detection, it also includes robust keypoint detectors for the face and the body, making it a complete keypoint detection suite that doesn't require any extra third-party software to integrate with. In fact, *OpenPose* can detect a total of 137 keypoints on the acquired RGB images, which are sufficient for accurately capturing the motion of the signer.

Each keypoint detector in *OpenPose* is a deep neural network and, more specifically, a special type of CNN called *Convolutional Pose Machine* (CPN). CPNs have the ability to learn long-range dependencies among images and multi-part cues, and also, inherit a modular sequential design. These features combine with the advantages afforded by convolutional architectures, thus making the networks capable of learning feature representations for both image and spatial context directly from data. In the first stage, the convolutional pose machine predicts part beliefs from only local image evidence, while the convolutional layers in the subsequent stage allow the classifier to freely combine contextual information by picking the most predictive features. More comprehensive information about the architecture of a CPN can be found in [2].

In order to avoid the problem of annotating databases for hand detection, the training process of a CPN is done using a technique called *Multiview Bootstrapping*. While a thorough analysis of this

method is provided in [3], we will also mention some important aspects of it for the completeness of presentation. Multiview bootstrapping is an approach that allows the generation of large annotated datasets using a weak initial detector. More specifically, the weak detector is trained on a small annotated dataset in order to detect subsets of keypoints in the so called “good views” which are the views for a certain frame achieving the highest scores as evaluated by a heuristic scoring policy. A robust 3D triangulation procedure is then used to filter out incorrect detections. Images where severe occlusions exists are then labeled by reprojecting the triangulated 3D hand joints. The inclusion of the newly generated annotations in the training set, iteratively improves the detector, and thus, in each iteration we obtain more and more accurate detections. With this approach we generate geometrically consistent hand keypoint annotations using constraints from the multiple views as an external source of supervision. In this way, we can label images that are difficult or impossible to annotate due to occlusion.

More formally, if $d(I)$ is a keypoint detector on an image I, its output would consist of a set of tuples of position vectors x_p and confidence values c_p, with each tuple corresponding to a certain keypoint,

$$d(I) \mapsto \{(x_p, c_p)\ for\ p \in [1 \dots P]\}$$

An initial training set T_0 consists of annotated images with a predefined set of N_0 keypoints, i.e.,

$$T_0 \coloneqq \{(I^f, \{y_p^f\})\ for\ p \in [1 \dots N_0]\}$$

We train an initial detector d_0 by using stochastic gradient descent on dataset T_0,

$$d_0 \leftarrow train(T_0)$$

This detector is then used to produce a new dataset of labeled images, namely T_1. The bootstrap technique suggests that an improved detector can be generated if we construct a new training set by concatenating datasets T_0 and T_1,

$$d_1 \leftarrow train(T_0 \cup T_1)$$

This procedure continues until we find a keypoint detector that achieves high accuracy results on a given test set.

As we mentioned above, the detectors are trained on datasets containing only “good views”. This is an important point to note because the inclusion of erroneously labeled frames in the training dataset will lead the iterative process to failure. While the selection of the valid frames could be realised by using uniform temporal subsampling on the videos, another technique is used which segments the video into windows of W frames (e.g., W=15 or W=30) and picks only the best frame from each window. By “best” we mean the frame that has the maximum sum of detection confidences for the different views. The mathematical formula with which scores are given to frames, is:

$$score(\{X_p^f\}) = \sum_{p \in [1 \dots P]} \sum_{u \in I_p^f} c_p^u$$

Finally, the frames of the training set must be checked for errors. This validation is done manually by visual inspection of the top 100 frames in the training set.

As we mentioned earlier, there is a detector for each part of the signer's body. The hand detector finds 21 keypoints on each hand of the signer, that is, a total of $2 \times 21 = 42$ keypoints are detected (Figure 7). These include keypoints on the tips of the fingers, other articulation points on the arches, as well as, one point on each wrisp of the signer.

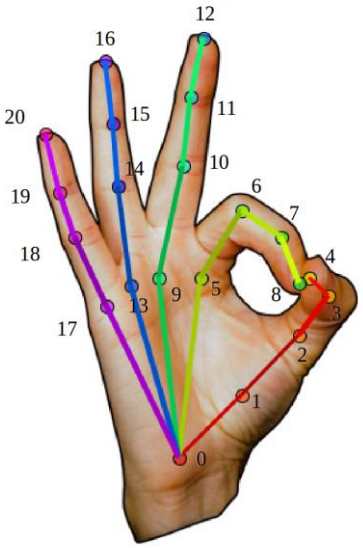


Figure 7: Hand keypoints.

As we mentioned in the previous paragraph, the body detector now detects 25 keypoints instead of 18 in the older version (Figure 16). Concerning the arms of the signer, keypoints for the wrisps, the elbows and the shoulders are detected. Keypoints for the hips, the knees and the feet are detected for legs. Also, keypoints for the neck and the head of the signer are detected.

Finally, the face detector detects 70 keypoints (Figure 8). These keypoints include the contour of the face, the eye orbits and eyelids, two layers of points for the lips and also some points for the nose of the signer.

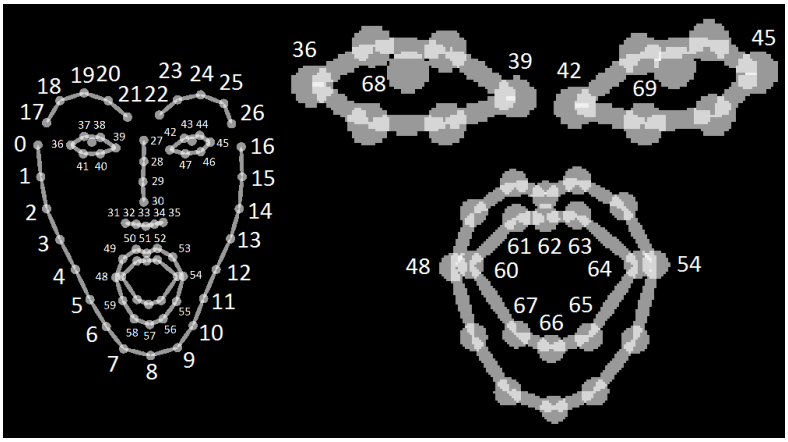


Figure 8: Facial keypoints.

In order to retrieve all the keypoints that are necessary for the accurate capturing of Sign Language motion, all three detectors must be executed upon each acquired frame. Considering that these detectors are deep neural networks and computations of just one deep network are heavy even in its inference phase, it is evident that the process of the overall keypoint detection is a very computational demanding task if done in parallel. Therefore, there is the option of executing the detectors in a sequential manner, one after the other, or executing only two detectors in parallel. For example, we can execute the hand and body detectors first, and then the face detector by its own. As an example, we see in Figure 9 the hand and body detectors running in parallel and finding keypoints on a signer.

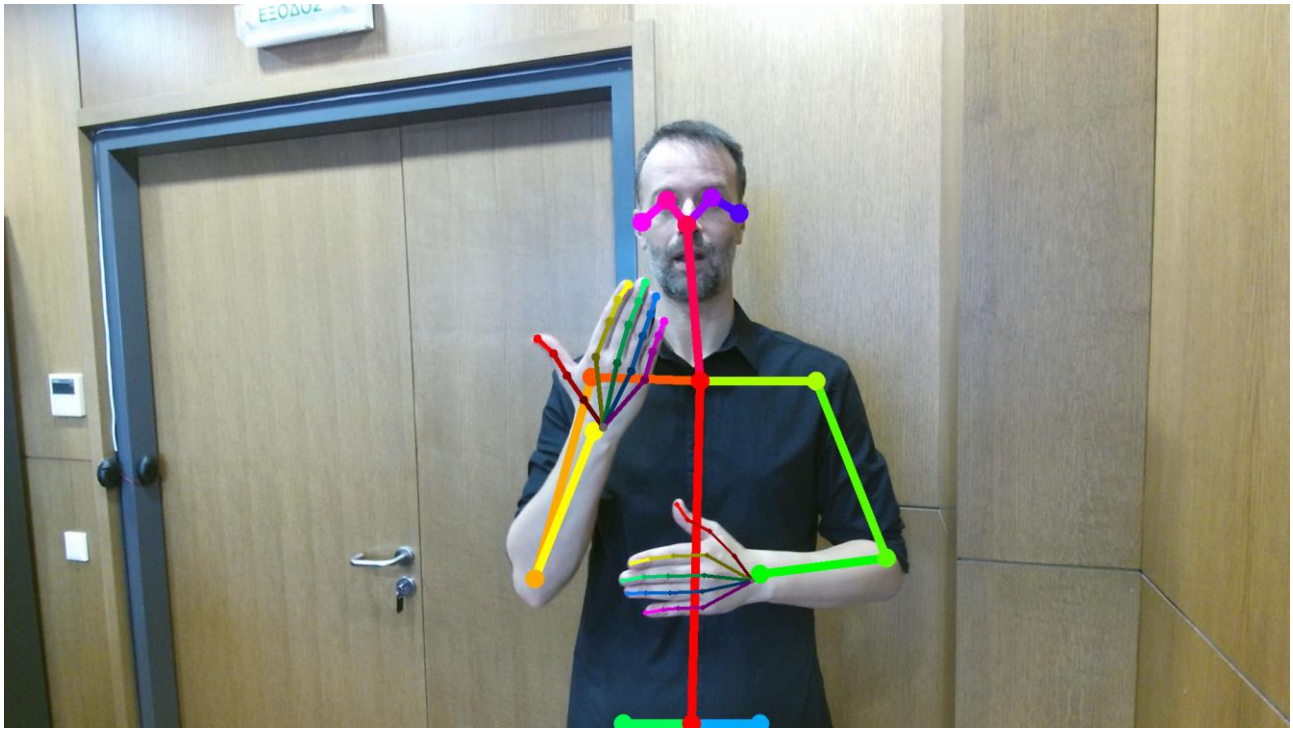


Figure 9: 2D keypoints detected with the new skeleton model for hands and upper body of a signer speaking Sign Language.

4.3. Generating 3D Keypoints

In order to provide avatar playback, the keypoints extracted from the captured images have to be mapped to specific control points on the avatar. Using a 2D detection algorithm, we only get the values corresponding to the width and height of an image which is not appropriate for controlling a 3D avatar. We, therefore, need a technique to infer the depth dimension and, by using it, construct 3D keypoints.

As we mentioned earlier in section 3.2, it is more practical and reliable to use the aligned depth frames given by an RGB-D sensor over a multi-view 3D reconstruction solution. What we do is simply extracting the values from depth frames for each X and Y coordinate, and then merge all three values and store them in a single file.

Another issue that we have to consider is that in case the avatar has fixed control points, the generated 3D keypoints from the detection phase need to map the positions of those control points. For example, if an avatar has a fixed control point on the hip, then the detection algorithm

needs also to find that specific keypoint on the acquired images. If a misalignment exists between the avatar control points and the detector keypoints, then we either have to re-train the detection algorithm or to perform mathematical transformations in order to re-position the skeleton points properly.

Figure 10 shows how the 3D keypoints look if we draw them using OpenGL. A signer made Sign Language gestures and images were captured using an RGB-D sensor. In the detection phase, we initially found hand and body keypoints from RGB data using detection algorithms, and then, the depth values from the aligned depth frames were extracted to form 3D data. The result shows that in the given frame all keypoints were accurately detected without being affected by occlusions or noise. For this demonstration though, we did not use the face detector so no facial keypoints are show in the picture.

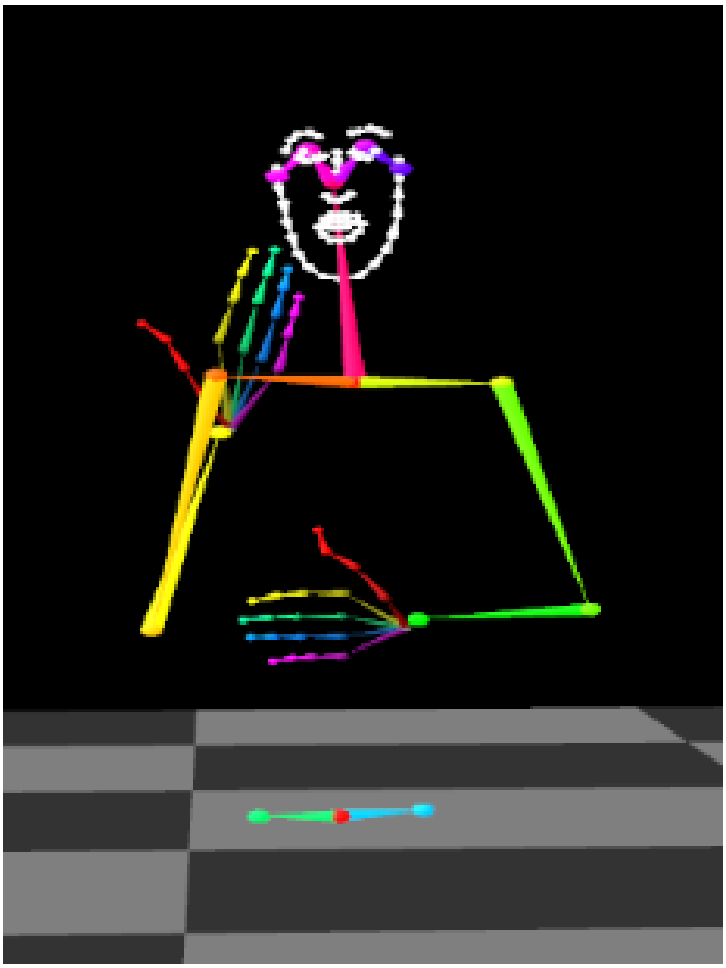


Figure 10: 3D keypoints drawn in OpenGL for Sign Language, formed by merging the 2D keypoints shown in Figure 9 with the corresponding values from the aligned depth frames.

5. MOTION REFINEMENT

In this chapter we describe the types of errors typically encountered in the detection process and the techniques that we apply in order to correct them.

5.1. Filtering Motion Data

Data from the detection process can contain errors. This means that either some keypoints are not detected by the algorithms, i.e., missing keypoints, or the algorithms produce erroneous values for some keypoints, i.e., misdetections. The first case can occur due to occlusions from other body parts while the signer makes gestures. For example, gestures made for a certain sign might involve hiding some fingers behind the other hand. When the acquired images are given to the detection algorithms, the keypoints of the hidden parts would not be detected. The second case can be encountered in the presence of noise. Such noise can be generated due to sensor specifications, room lighting conditions, or even colours of the clothes that the signer wears.

Noisy data can have negative effects when propagated in formatted motion data. Most motion file formats follow a hierarchical structure under which the position of each body part is defined as an offset from the previous one. Thus, errors in joint detection will affect other joints in the hierarchy as well. It should be apparent that under such a scheme, even a single error occurring on a keypoint value might disturb the skeletal structure and motion greatly and produce unwanted results when the motion file is imported for avatar playback.

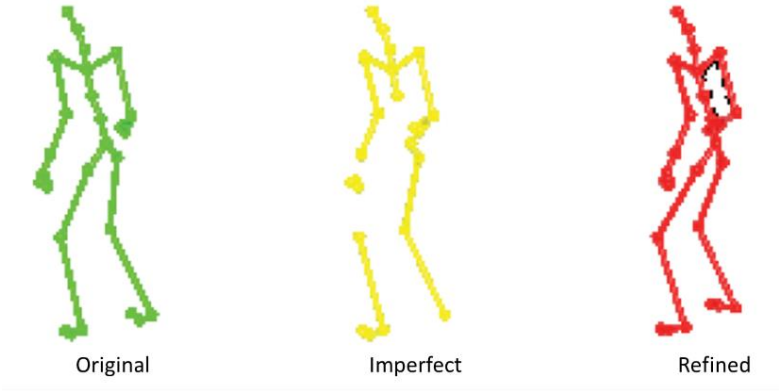


Figure 11: Green color: An ideal keypoint detection, Yellow color: An imperfect detection with missing keypoints, Red color: Keypoints after a motion refinement process.

In the process of filtering, certain components of a signal in either the time or the frequency domain are either reduced or amplified. For example, in the frequency domain filtering uses a transfer function to change the Fourier coefficients of frequency components. For cutoff frequencies f_1 and f_2, the transfer function of the filter has three frequency spectrum bands: pass $[0, f_1]$, transition $[f_1, f_2]$, and stop $[f_2, \infty]$. Frequency components covered by the pass band remain unchanged. On the other hand, the transition band decreases the power of frequency components. Finally, the stop band reduces or eliminates all the remaining frequencies. In the filtering process, the power spectral density is multiplied by the transfer function. The important factor of the process is the selection of an appropriate cutoff frequency since a high cutoff allows noise to pass while a low cutoff could eliminate part of the signal.

A well-known filter in field of signal processing is the low pass Butterworth filter. This filter is applied to a forward time series $< x_1, x_2, \ldots, x_n >$ in order to obtain a filtered time series $< y_1, y_2, \ldots, y_n >$. The filter is then reapplied to the reverse filtered time series $< y_n, y_{n-1}, \ldots, y_1 >$. The order of the Butterworth filter indicates the sharpness of the transfer function. A second order filter is found to

work reasonably well with motion data. Extrapolation techniques are required in order to create data points near the edges of the signal since each filtered data point is a function of previous and future data points.

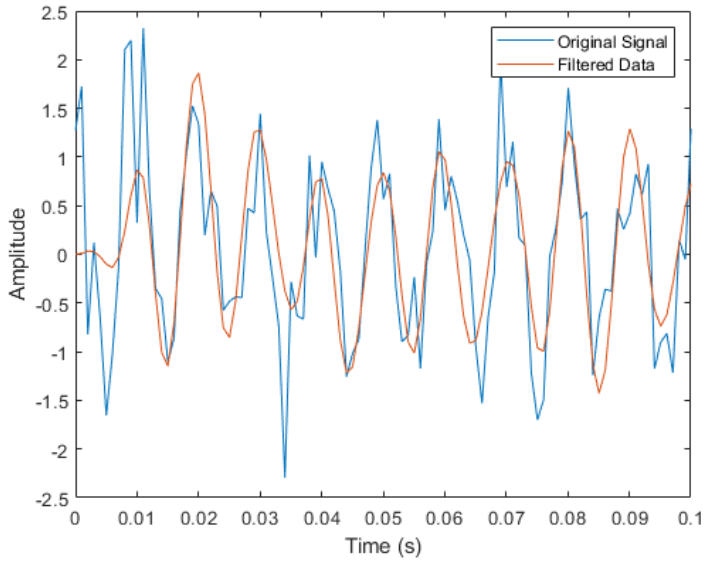


Figure 12: Filtering a signal in the time domain.

5.2. Reducing Jitter with a Moving Average Filter (MAF)

5.2.1 Filter description

We can describe a moving average filter (MAF, also known as rolling average or running average) in different ways. In general, this type of filter calculates the running weighted sum of a time series.

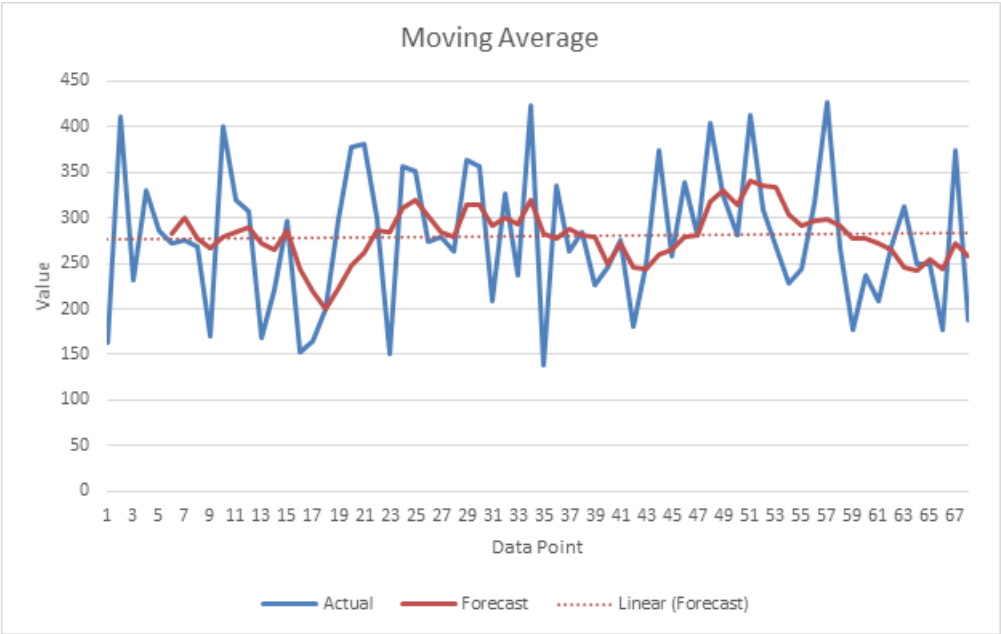


Figure 13: An example of moving average filtering.

More specifically, a MAF is a simple Low Pass FIR (Finite Impulse Response) filter that is commonly used for regulating an array of sampled data (i.e., signal). It takes M samples of input at a time and calculates the average of those to produce a single output point. As the length of the filter increases, the smoothness of the output increases, whereas the sharp modulations in the data are made increasingly blunt.

This type of filter is normally used with time series data to smooth out short-term fluctuations and highlight longer-term trends or cycles. The threshold between short-term and long-term depends on the application, and the parameters of the moving average will be set accordingly. For example, MAF is often used in technical analysis of financial data, like stock prices, returns or trading volumes. It is also used in economics to examine gross domestic product, employment or other macroeconomic time series. Mathematically, a moving average is a type of convolution and so it can be viewed as an example of a low-pass filter used in signal processing. When used with non-time series data, a moving average filters higher frequency component without any specific connection to time, although typically some kind of ordering is implied. Viewed simplistically it can be regarded as smoothing the data.

MAF smooths data by replacing each data point with the average of the neighboring data points defined within the span. The whole process is equivalent to lowpass filtering with the response of the smoothing given by the difference equation:

$$y_s(i) = \frac{1}{2N+1}(y(i+N) + y(i+N-1) + \cdots + y(i-N))$$

where $y_s(i)$ is the smoothed value for the i^{th} data point, N is the number of neighboring data points on either side of $y_s(i)$, and $2N+1$ is the span. So, in this way, we can describe a moving average filter as a type of low-pass filter (LPF) that doesn't have any control over its bandwidth for a fixed number of coefficients.

There are some differences between standard low-pass filters and a MAF. For a finite impulse response (FIR) filter, the output signal $y[n]$ is given in terms of the input signal $x[n]$ and the filter coefficients $h[n]$:

$$y[k] = \sum_{n=0}^{N-1} h[n]x[k-n]$$

The filter length in this case is N coefficients. A moving average filter has coefficients which are all equal:

$$h[n] = \frac{1}{N}, \qquad n = 0, 1, \ldots, N-1$$

whereas in general, a low-pass filter can have different values for each coefficient. This allows controlling of the frequency selectivity of the filter.

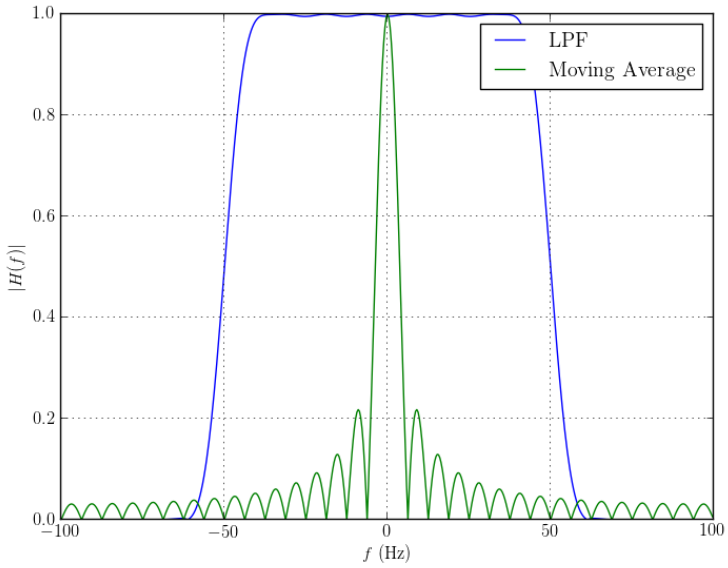


Figure 14: Difference in coefficients of a LPF and a MAF.

Within the context of image processing we can describe the moving average filter as an algebraic operation performed on local image neighborhoods according to a geometric rule defined by the window. Given an image f to be filtered and a window B that collects gray level pixels according to a geometric rule (defined by the window shape), then the moving average-filtered image g is given by

$$g(n) = AVE[Bf(n)]$$

where the operation AVE computes the sample average. Thus, the local average is computed over each local neighborhood of the image, producing a powerful smoothing effect. The windows are usually selected to be symmetric.

Another point to mention is that a MAF has the desirable effect of reducing zero-mean image noise toward zero. However, the filter also effects the original image information. The moving average filter, which is lowpass will blur the image, especially if the window span is increased. Balancing this tradeoff is often a difficult task.

5.2.2 Types of moving average filters

SMA (simple moving average)

A simple moving average filter, denoted as $SMA(k)$, is a finite impulse response filter that for any moment t returns the average of the previous k values. An interesting feature of this type of MAF is that for any width k and time series of length N, its output can be efficiently calculated in $O(N)$ time (having no dependence on k).

CMA (cumulative moving average)

In a cumulative moving average (CMA) filter, the data points arrive in an ordered fashion and the output is the average of all of the data up until the current data point arrived. This type of calculation has many real-word applications, for example, an investor may be interested on the average price of all stock transactions for a particular stock up until now. When a new transaction occurs, the average price at the time of the transaction can be calculated as a function of all transactions up to that point, using the cumulative average. This is typically defined as the equally weighted average of the sequence of n values $x_1 + .. + x_n$ up to the current time:

$$CMA_n = \frac{x_1 + .. + x_n}{n}$$

The obvious way to calculate this would be to store all the data, calculate the sum and divide by the number of data points every time a new point arrived. However, it is possible to simply update the cumulative average as a new value x_{n+1} becomes available, using the formula:

$$CMA_{n+1} = \frac{x_{n+1} + n \cdot CMA_n}{n+1}$$

So, the cumulative average for a new data point depends on all previous calculations of CMA. When all data points arrive ($n = N$), then the cumulative average will equal the final average.

EMA (exponential moving average)

An exponential moving average filter, denoted as $EMA(k)$, is an infinite impulse response filter which is defined as a transformation of a time series according to the following formula:

$$EMA(t) = aX(t) + (1-a)EMA(t-1)$$

where a is the smoothing parameter and controls the effect of previous calculations on the current data point.

LRMA (linear regression moving average)

A linear regression moving average filter, denoted as $LRMA(k)$, is a finite impulse response filter which fits a straight line to a sliding window with width k. Although LRMA filtering costs more than simple or exponential moving average filters (its running time is $O(N \cdot k)$), its linear regression estimation tends to better follow the trend line.

5.2.3 Applying MAF on Sign Language motion data

We apply the moving average filter on our Sign Language recordings in order to reduce jitter coming from either Intel's RealSense depth sensor or any of the keypoint detectors we use. As we mentioned earlier, jitter is expressed as small fluctuations of keypoint positional values over time. During the reporting period, we have experimented on all MAF types (listed in section 5.2.2.) and finally concluded to the one which seems to give the best results for our capturing setup. More specifically, we have found the EMA filter to give better results than the rest of MAF types and effectively reduce jitter while preserving natural keypoint motion in most Sign Language recordings that we tested. The following diagrams showcase how motion with jitter looks like and how it can be resolved with the application of EMA filtering.

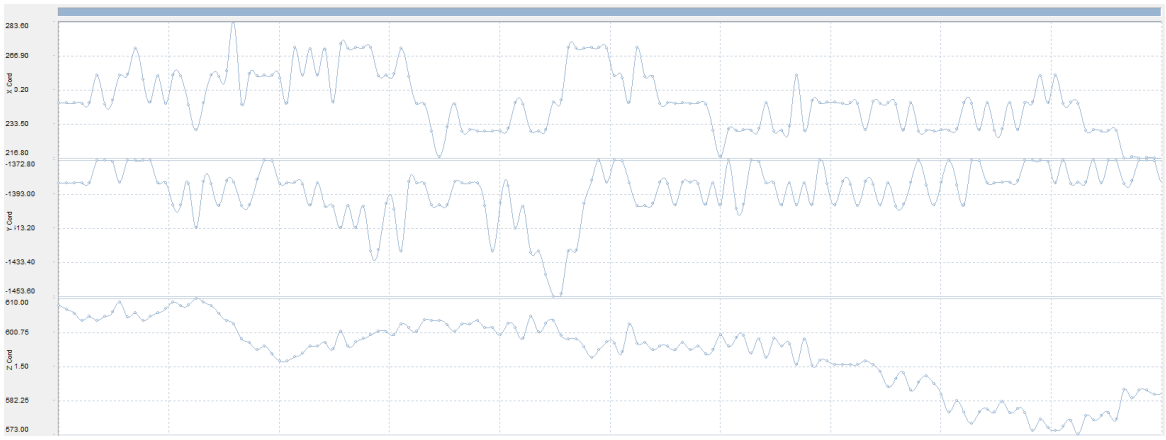


Figure 15: A keypoint with jitter (i.e., no filters applied).

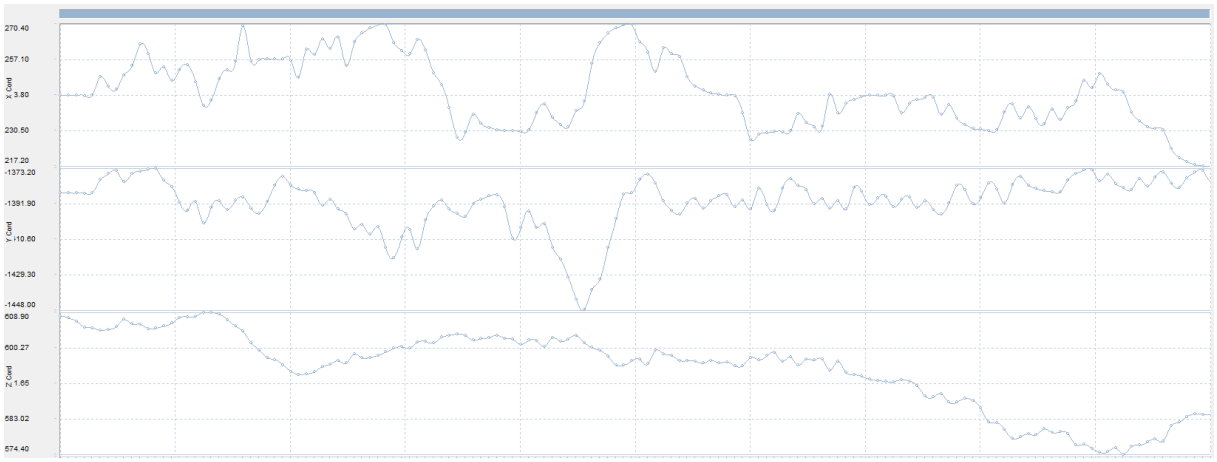


Figure 16: Applying EMA (smoothing parameter = 0.5).

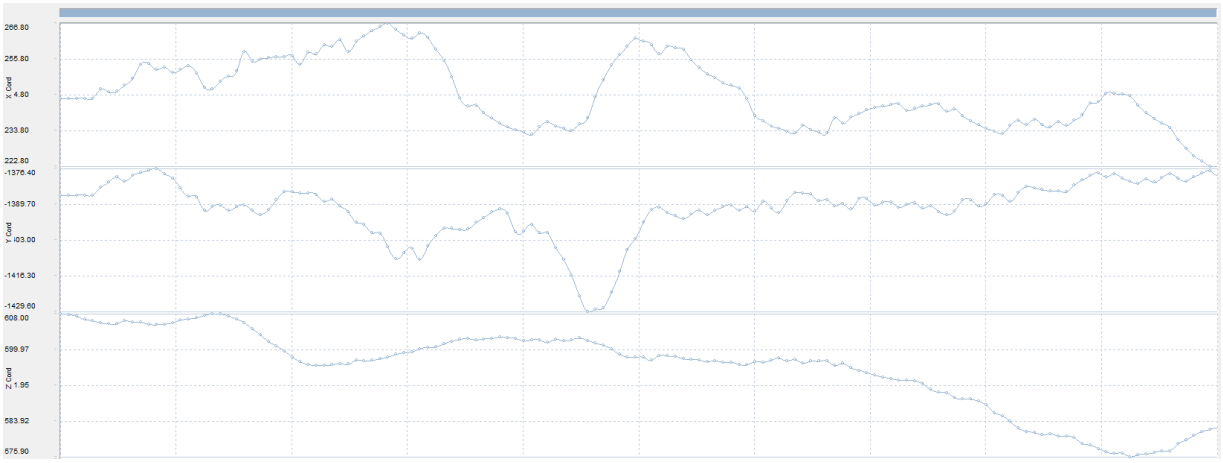


Figure 17: Applying EMA (smoothing parameter = 0.2).

As we can see in the figures above, keypoints might contain jitter in all 3 coordinate dimensions (see fluctuations of values through time in Figure 25). Application of MAF with a smoothing parameter of 0.5 (found experimentally) in each dimension is effective in reducing jitter and smoothing the motion of keypoints. If we further decrease the value of the smoothing parameter (e.g. 0.2), a lot of details in motion are lost and motion becomes 'over-smoothed out' (Figure 26).

5.3 Removing Spikes with a Median Filter (MF)

5.3.1 Filter description

The median filter (MF) is a non-linear digital filtering technique that is commonly used to remove noise from an image or signal. This type of filtering is a very widely used technique in digital image processing because it removes noise while also preserving edges. It is a common pre-processing step that improves the results of other tasks such like detecting edges on an image. The use of median filters was first suggested for smoothing statistical data. The filter has, however, found most of its applications in the area of digital image processing. Its edge-preserving properties can remove noise and speckles without blurring the picture. Removing artifacts from imperfect data acquisition, for instance horizontal stripes sometimes produced by optical scanners, is done successfully using median filters. They are also used in radiographic systems, in many commercial tomographic scan systems and for processing electroencephalogram (EEG) signals and blood

pressure recordings. Recently, it is also likely to be found in newer commercial digital television sets because of the very good cost-to-performance ratio.

The process of median filtering involves running through the signal entry by entry, replacing each entry with the median of neighboring entries. The set of neighbors is called “window”. Using this window, the filter slides entry-by-entry over the entire signal. For 1D signals, the most obvious window is just the first few preceding and following entries, whereas for 2D (or higher-dimensional) data the window must include all entries within a given radius or ellipsoidal region (i.e. the median filter is not a separable filter). So, in other words, a median filter is a sliding window of sampled data values, similar in concept to a moving-average filter. The difference is that a MAF computes the mean of the window values and is therefore affected by large-amplitude outliers, while MF selects as its output the median value. We can think of the process of median filtering like a sorting operation after each new element is introduced into the data window and then selecting the value which lies midway between all values in the window. This is not the only method for determining the median, but an adaptation of this approach provides one of the fastest possible hardware implementations. In a real-time setting, we have found it convenient to initialize the data window with zeros, although other values are possible. Ordinarily, an odd number of taps is used, often a horizontal window with 3 taps; occasionally, 5 or even 7 taps are used. Sometimes spatial median filters are used (for example, 3×3).

In many applications, the most difficult noise to deal with is what is referred to as “impulse noise”. What is meant by this term is noise that consists of relatively high-amplitude, narrow “spikes” (i.e. exhibit unwanted transients) that are uncharacteristic of the signal. Lots of applications are potential sources of impulse noise. Brushed DC motors are definitely a source of such noise, as anyone who has tried to maintain a “clean” power supply in the proximity of an operating vacuum cleaner can attest. As an example, consider the open-loop voltage across the input of an analog instrument in the presence of 60 Hz power-line noise. The sample rate is 1 kHz. Commutator brush noise created the spikes in the waveform. Attempts to remove these spikes with a linear filter are complicated by the fact that the rate of occurrence of the spikes is a function of the motor speed, which is rapidly changing during the transient. Hence, it may be difficult or impossible in many situations to select a (nonadaptive) linear filter cutoff frequency which removes the spikes without distorting the underlying edge. The tools for dealing with such disturbances aren’t necessarily the same as those for dealing with random noise. In such cases, a median filter is particularly good for removing impulsive type noise from a signal.

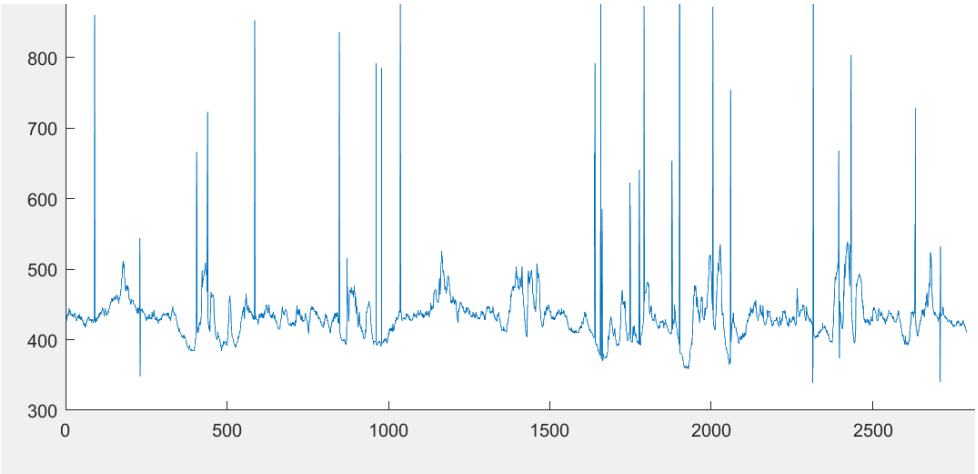


Figure 18: An example of 1D signal with spikes.

Frequently used in image processing, MF is a type of nonlinear filter that is very effective at removing impulse noise (i.e. the “salt and pepper” noise in an image). In this context, the principle

of median filtering is to replace the gray level of each pixel with the median of the gray levels in a neighborhood of the pixels, instead of using the average operation. More specifically, we specify the kernel size, list the pixel values, covered by the kernel, and determine the median level. If the kernel covers an even number of pixels, the average of two median values is used. Before beginning median filtering, zeros must be padded around the row edge and the column edge. There are a number of variations of this filter, and a two-dimensional variant is often used in DSP systems to remove noise and speckles from images:

$$y(n) = med[x(n-k), x(n-k+1), \dots, x(n), \dots, x(n+k-1), x(n+k)]$$

where $y(n)$ is the output and $x(n)$ the input signal. The filter "collects" a window containing $N = 2k+1$ samples of the input signal and then performs the median operation on this set of samples. The median filter itself is simple and in the standard form there is only one design parameter, namely the filter length $N = 2k+1$.

Studying the frequency properties of the filter is not typically one of the best ways, since the median filter is nonlinear. One successful approach is to consider how the median filter alters the local geometry or shape of a waveform. Any discrete waveform may be described as a sequential collection of constant neighborhoods, edges, impulses, and oscillations. Passing a signal once through a median filter will eliminate impulses and reduce oscillations. It has been shown that repetitive median filtering of a finite length signal will produce, after a finite number of repetitions, a root signal which is invariant to further applications of the filter. Such a root signal consists only of edges and constant neighborhoods. Hence, if a waveform consists of an underlying signal of edges and constant neighborhoods corrupted with noise which consists of impulses and oscillations, the median filter will remove or reduce the noise without modifying the underlying signal.

5.3.2 Applying MF on Sign Language motion data

Except for jitter manifesting in their motion data, Sign Language recordings might also contain spikes. This might happen due to failure in sensor readings or the keypoint detection process. In that case, a keypoint loses its position and receives extreme values to its coordinates, usually for a very short period of time before finally returning to its previous state. The following figures illustrate how the application of MF can effectively remove spikes from Sign Language motion data.

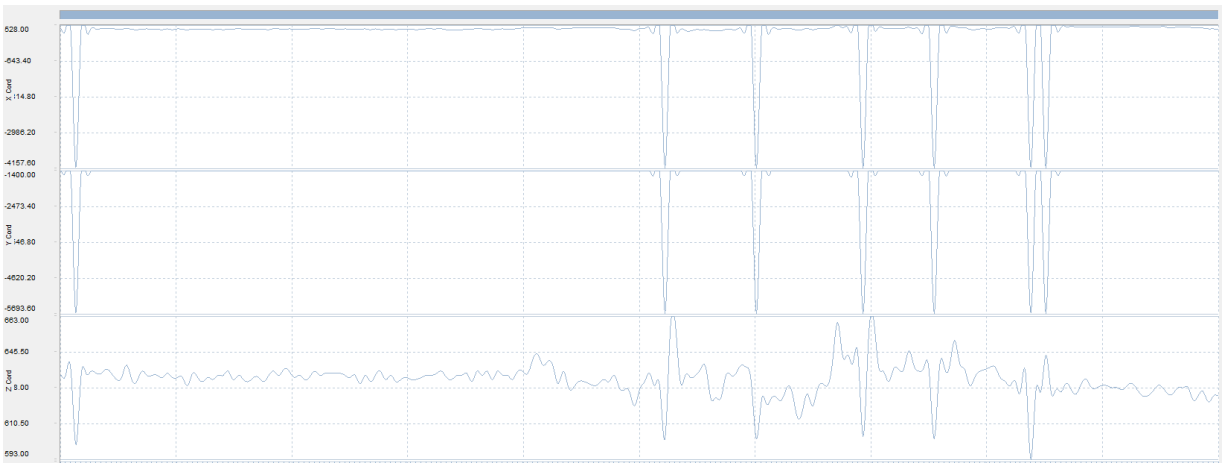


Figure 19: A keypoint with spikes (no filters applied).

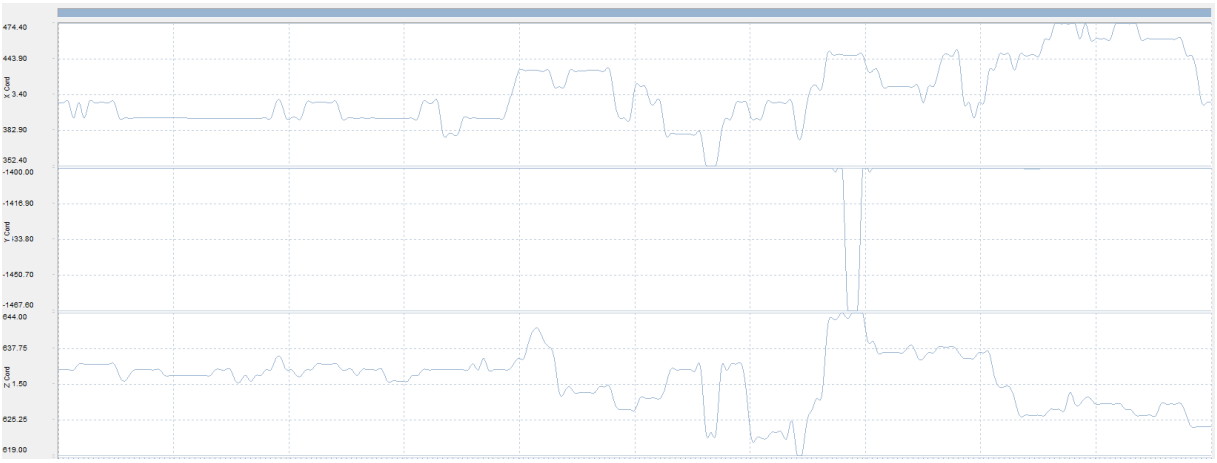


Figure 20: Applying Median Filter (MF) effectivelly removes spikes.

5.4 Eliminating Depth Occlusions

In an ideal motion capture recording, the 3D trajectory for a marker lasts for the whole sequence of frames, from the first frame till the last one. This is not the case for real-life motion capture sessions where trajectories are broken by occlusion which causes a point to be missing and lost in the tracking process. This is a significant problem in both marker-based and markerless motion capture systems. In a marker-based system, markers may be occluded by moving subparts of the actor or other objects in the scene. Self-occlusion occurs when rotation of a subpart makes a marker in this subpart invisible from the camera origin. Absence of a marker also happens when a sensor for some marker fails. When multiple cameras are involved, triangulation may become impossible if enough cameras do not see a marker, and thus, the 3D position cannot be calculated. Usually, increasing the number of cameras mainly reduces occlusion effects while also improves accuracy and coverage. In a markerless setting, the same issues occur for keypoint detectors trained to detect skeleton points on RGB and/or depth images.

Figure 21: Occlusions are very common in Sign Language. In this recording, left hand hides all right hand's fingers.

The problem of occlusion becomes even more noticeable in Sign Language recordings. In most signs, the motion of hands and fingers of the signer may hide other body parts and produce occluded keypoints. In our setup that consists of just one RGB camera and depth sensor, occlusion is evident on the depth values of keypoints. Therefore, we developed a simple rule-based occlusion elimination strategy that replaces an occluded keypoint's depth value to the last valid one and thus enhances the quality of our Sign Language recordings.

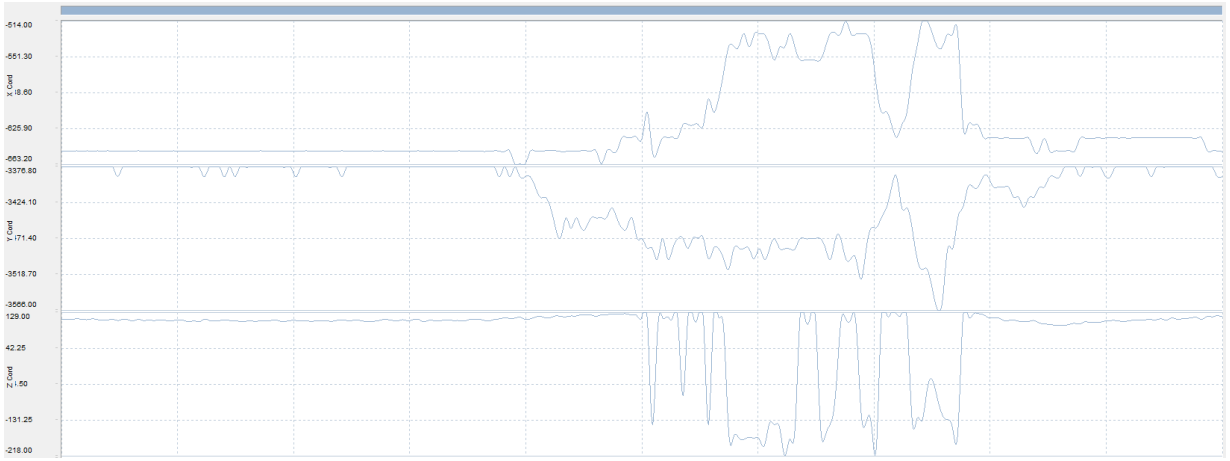


Figure 22: Keypoint with occlusion (distored depth).

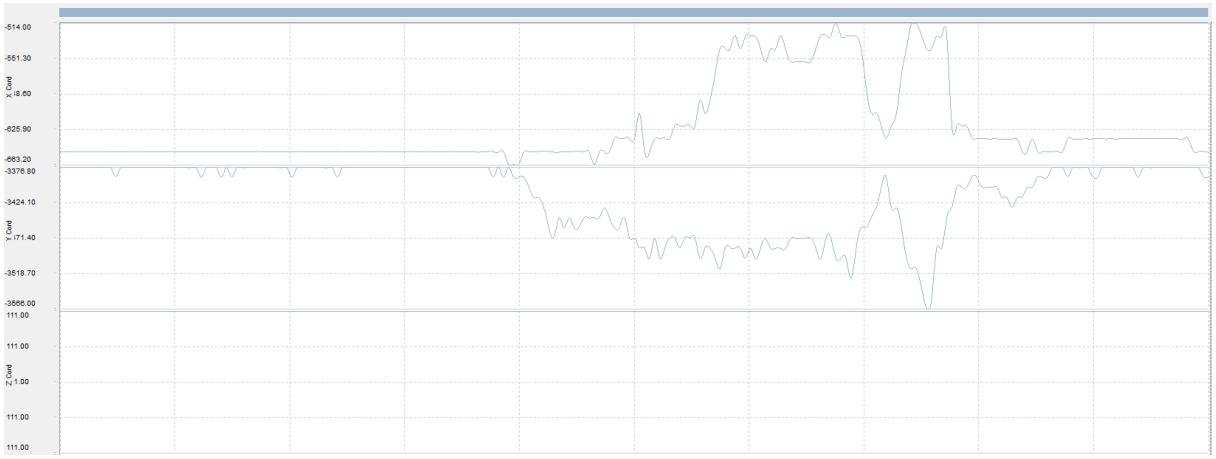


Figure 23: Occlusion elimination (stable depth).

5.5 Repercussions on Sign Language Recordings

Despite their obvious benefits, applications of refinement techniques on recorded motion can also produce some unwanted effects. Adverse effects of filter application can occur mainly due to over-smoothing effects and result in motion that tends to be unnatural and unrecognizable. In such cases, Sign Language recordings might contain unrecognizable motion even if it comes from an expert signer. Having said that, we have to properly adjust the filter's parameters in order to both eliminate noise in motion data and also preserve naturalness in Sign Language recordings.

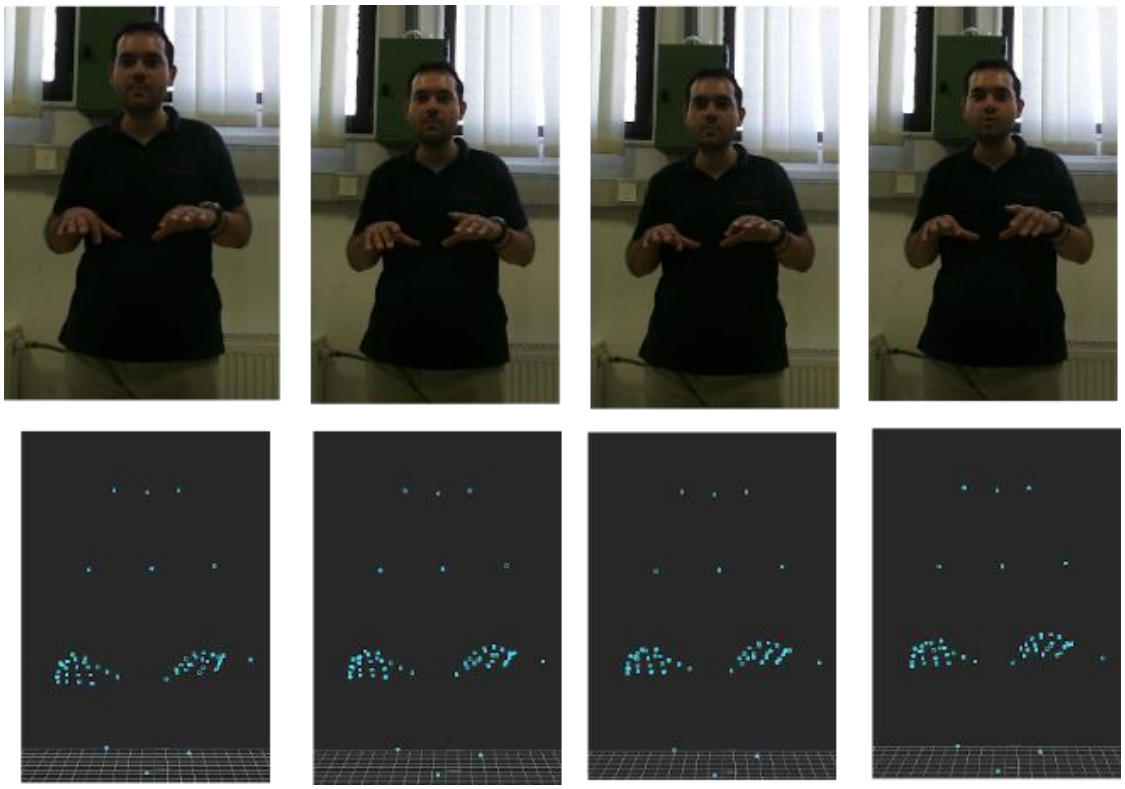


Figure 24: Capturing the sign 'computer' in English Sign Language. Post-processing with strong filter application for noise reduction results in loss of important data for signer's finger motion.

6. EXPORT

This chapter discusses the files that are exported by the capturing technology, their internal format, and the EasyTV services that use them.

6.1. Types of Files

The EasyTV capturing module generates a number of different file types. These files are essential to other EasyTV services and modules. Examples are the realistic 3D avatar, the multilingual ontology, as well as the crowdsourcing platform. The files are initially uploaded to the crowdsourcing platform and then stored into repositories in order to be accessible by these services and modules. Below are some types of files that are exported by the capturing module.

- Images

These files are the images containing the frames captured in a recording session. There are 2 types of images exported: RGB and depth images. Depth images are useful for extracting the 3rd dimension of depth for creating 3D data (as described in Section 4). On the other hand, all RGB images are used by the keypoint detectors to detect 2D skeleton points, but are also uploaded to the crowdsourcing platform in order to be given as input to the multilingual ontology module.

- Motion files

These files contain the motion data generated in the detection phase of this module, as described in chapter 4. The selection of the right file format affects the simplicity in calculations when creating the file and the availability of solutions when importing it for avatar playback. Considering the importance of these two aforementioned points, we devote the next sub-sections to this topic.

- Video files

These files concern videos containing the frames acquired by the RGB-D sensor. While there are two types of frames captured, RBG and depth frames, the capturing module generates a video only for RGB images because this is required by the multilingual ontology module. The file format of our choice is .mp4 (since .avi formats are larger than .mp4). These videos can also be used in the validation phase of the crowdsourcing process by the moderator of the crowdsourcing platform, as well as, in search options for database content retrieval.

- Annotations

These files contain annotations provided by the end users of the capturing module. For each recording session, one or more annotations may be given to describe the signs being captured.

- Description file

This file is a sum-up of all the exported files. More specifically, it describes the mapping between motion files, video files, and user annotations which ascribe the meaning of each sign. Currently, we decided to use XML as the preferred format for these files due to its simplicity and also its straightforward API inclusion by most programming languages.

6.2. Exporting C3D files

Most major motion capture software systems support the C3D (Coordinate 3D) file format for exporting motion. This format offers the ability to store 3D and analog data in an unprocessed form. C3D files can also store processed data but in its standard form the format stores raw 3D coordinate and analog sample data, together with information that describes the stored data (i.e. metadata). As mentioned in the official C3D website, the format provides the following features:

- Preserves information that describes the physical data collection environment of the laboratory such as EMG channels used, force plate positions, force plate types, marker sets, etc.
- Stores 3D and analog trial information relating to the circumstances of the test session such as the number of 3D points monitored, 3D and analog sample rates, EMG muscles recorded etc.
- Potentially stores subject information - name, age at trial, with physical parameters such as weight, leg length etc.
- Stores calculated analysis results such as gait timing, cycle information and related information.
- Extensibility - the C3D format provides the ability to store new information without making older data obsolete.
- The public specification and format description of the C3D format allows anyone to access data without depending on a manufacturer for information.

The C3D specification expects physical measurements to be one of two types, either positional information (3D coordinates) or numeric data (analog information).

- Each 3D coordinate is stored as a raw X, Y, Z data samples with information about the sample - accuracy (the average error or residual), and camera contribution (which cameras were used to produce the data).
- Each sample of numeric data can contain analog information from sources such as EMG and Force Plates etc. and is linked to the 3D samples so that it is easy to determine the correct numeric data values for any 3D sample within the file. If desired (for high analog rates etc.) The C3D format can store multiple numeric samples per 3D coordinate sample. As a result, many C3D files contain both analog and 3D data linked frame by frame which is a big improvement over the OEM formats that store analog and video data separately. Storing related information in a single file gives a greater degree of confidence in the data and makes it easier to retrieve the relevant data.

In our case, we only record positional data and thus our C3D files contain X, Y, Z coordinate

values. No analog information is exported within our C3D files.

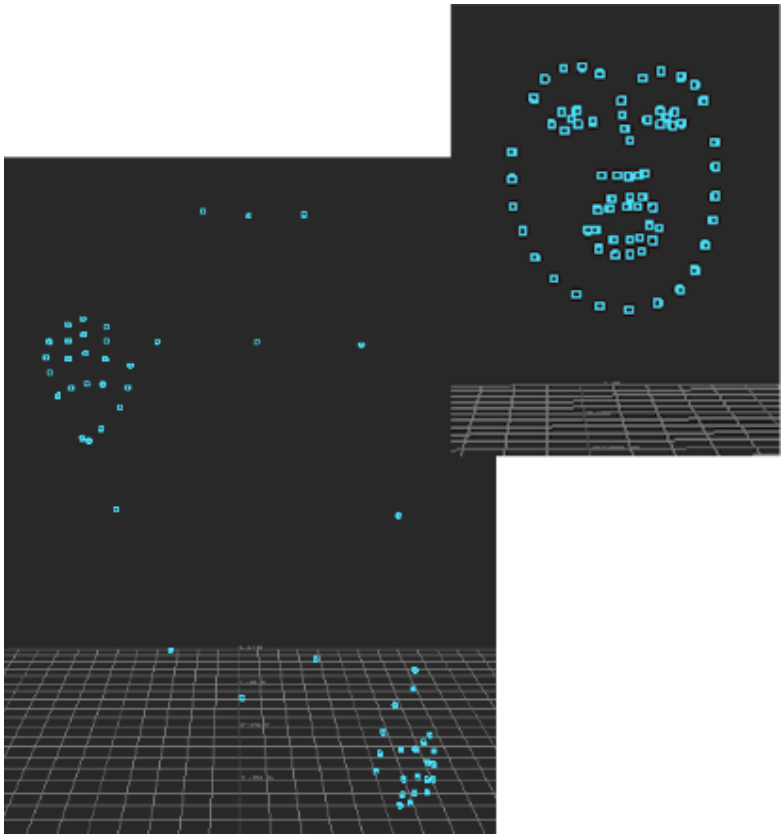


Figure 25: C3d files for the sign 'Hola!' in Spanish SL as shown in MotionBuilder. A full body mocap for the signer contains body, fingers and face keypoints.

The design of the C3D file format was originally driven by the need for a convenient and efficient format to store data. The format stores 3D coordinate and numeric data for any measurement trial, with all the various parameters that describe the data within a single file. This largely eliminates the need for motion-related data to travel around with additional notes and test information (which usually gets separated from the data at some point in its travels). The ability to store a multitude of information about the data is the feature that sets the C3D format apart from every other biomechanics format. As a result 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. All early C3D files included camera contribution data and a 3D calculation residual stored with each 3D coordinate, allowing the quality and accuracy of every recorded 3D data value to be continually monitored and evaluated. Simultaneously, multiple samples of analog data could be sampled during each 3D frame to guarantee temporal synchronization, each analog sample being recorded as the value directly read from an ADC and recorded as raw sample values.

Overall, the design of the C3D format enables all the data collected to be stored with a minimum of processing and allows the user to verify the quality of the raw and unprocessed 3D and analog data collected during an experiment. Also, adding parameter information to a C3D file is very easy. Since the C3D format is not tied to any specific manufacturer it can be freely adapted to store the information that the users require without making a commitment to any specific manufacturer. The C3D file format provides a means of storing all the raw data and other information required to interpret or analyze the raw data at a later stage. Data stored in the C3D format can provide a

means of standardizing the interchange of information and can enable multi-user studies across a wide variety of manufacturers hardware and software platforms.

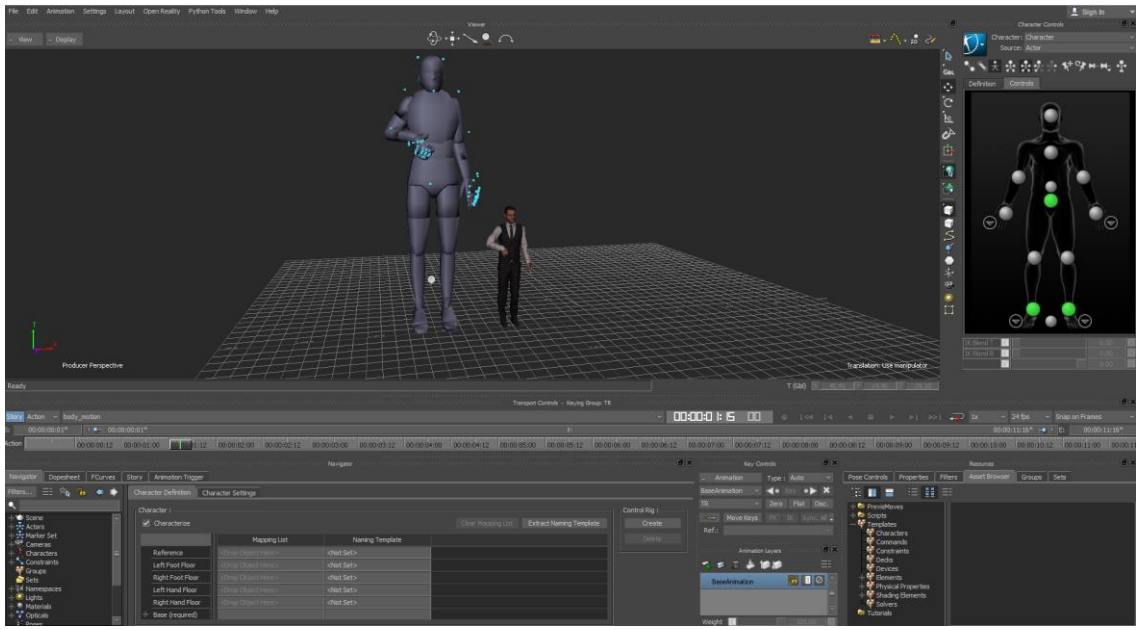


Figure 26: C3D motion files drive MotionBuilder's Actor and then motion is retargeted to a realistic 3D avatar.

It is recommended that the default configuration for any C3D file creation is to store unprocessed raw data. However, as data collection systems have evolved, many manufacturers have begun processing the raw sampled data before creating the C3D file and created their own additional data values in the file as pseudo-3D points to store processed and calculated data values. While this does not violate the C3D specification, it does mean that end users now have to implicitly trust their manufacturer's data collection and processing efforts because many C3D files these days do not contain any raw data as the manufacturer has removed the original "raw" data from observation. This simply means that any errors in data collection and processing before the C3D file is created, are effectively hidden from the end-user to the manufacturer's advantage, thus preventing a quality assessment of the data collection process. Towards that direction, using the C3D format to store processed data is an option that manufacturers can use without breaking the format but if all that is recorded is the processed data then the manufacturer has created an environment that is difficult to fault-find because it prevents the end-user from evaluating the raw data collected.

6.3. Exporting BVH Files

After obtaining 3D data of **motion-captured** human pose joints (body and hand) the purpose is to process them and produce rotational data – joints' relative rotations. This will enable the encoding of the recovered human pose of the 3D joint locations into standard **Biovision hierarchical data (B.V.H.)** format. The format is popular as it is supported by various 3D animation engines, which allows the animation scale-invariant retargeting of the poses to other avatar rigs. The basic characteristics of the BVH format (*more in D3.1):

- A root joint position.
- Relative angles of each joint.

- Fixed bone-length (encoded as offset between joints).

We could make a summary of the basic calculations involved in the production of rotational data in a number of steps as follows:

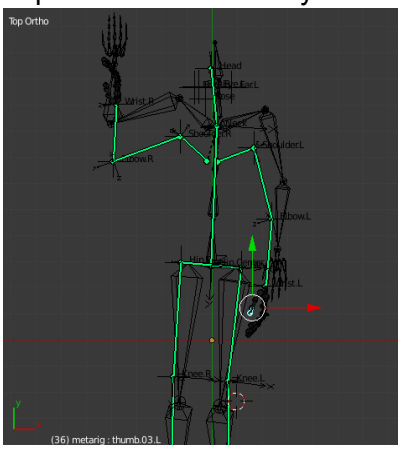
1. A local coordinate system is defined for each joint (according to human kinematics).
2. For each joint, calculate the **Rotation Matrix** which if applied, rotates the *child* bone (vector) and aligns it to the parent bone. This calculation is based on Quaternion math for robustness.
3. Calculate the **Euler Angles** from the rotation matrix (results in degrees).
4. Export the Euler angles in the BVH file according to the hierarchical specifications.

Considering our purpose of automating the generation of rotational data into our pipeline, it is possible to implement the previous calculations in an efficient manner using the Blender framework. The framework can be instructed to handle the low-level calculations to exporting various formats compatible for animating engines (like fbx files):

- An armature skeleton corresponding to our Mocap detectors' keypoints joints is created in prior in the blender workspace. This is essentially a 'stickman' avatar, which serves as the puppet model following the positional data.
- The per-frame raw data are imported as 3D geometry points and the armature's bones are instructed to follow them applying constraint rules.
  Example: in the snapshot, the right-side armature bone from shoulder to elbow is instructed to follow the keypoint tagged as 'Elbow.R'.
- A humanoid avatar, compatible for playback by other animation engines like Unity3D, copies the rotations to its own skeleton.
- The Blender engine can export the animation's joints relative rotations (Euler angles).

6.4. Uploading Files to the Crowdsourcing Platform

When the files are exported by the capturing module, they must be uploaded to the EasyTV crowdsourcing platform in order to be available to other EasyTV services and modules. The exported files are the result of a crowdsourcing task that the user has completed. The crowdsourcing platform contains a special form for uploading the files. This form is part of the UI that concerns the task that is appointed to this user. The different file types make it necessary to import one compressed folder instead of a number of different files. After this folder is uploaded, the crowdsourcing platform should decompress it, extract the files and proceed to the validation process. If the moderator validates the content of the files and accepts them as being the correct answer to the given task, the files are stored into the proper repositories. In the opposite case, that is, if the moderator rejects the motion capturing for the given task, the user has to repeat the task or the task may be assigned to a different user.



6.5. Input to Multilingual Ontology

The capturing module also sends information (via the crowdsourcing platform) of the recorded video to the multilingual ontology for the enrichment process being developed in the Task 3.2. A first proposal of an exchange JSON format is presented in Figure 27. The JSON file must contain information about the language of the recorded video, the oral language sentence that represents

the video and the sentence composed by the concatenation of signs, which cannot be the same as oral language. Moreover, information about each video segment is provided with the start and end in the video and their meaning. In addition, an URL where the video is stored should be provided.

The JSON file will be processed by the EasyTV-annotator library to enrich the multilingual sing language ontology (see document D3.2 Enriched multilingual ontology with signs in different languages preliminary version). The library processes the natural language sentence to retrieve linguistic aspects such as the tokens, the part of speech of the words, their lemmas and whether or not there is a named entity. This information is needed to find the suitable concepts in the ontology that represent each of the words that compose the sentence. Then, the class of the ontology is associated with the video segment received in the JSON file.

```
{
    "video": {
        "url": "http://.....",
        "nls": "the tree elephants",
        "sls": "elephant the three" ,
        "duration": "00:50" ,
        "language": "es" ,
                "segmnents": [  {
                "order": "1",
                "start": "00:00",
                "end" : "00:30",
                "content" : "elephant" ,
                }
                {
                "order": "2",
                "start": "00:31",
                "end" : "00:40",
                "content" : "the" ,
                }
                {
                "order": "3",
                "start": "00:41",
                "end" : "00:50",
                "content" : "three" ,
                }]
        }
}
```

Figure 27: An example of a JSON format for describing Sign Language content.

6.6. Avatar Playback

The visualization of the recorded signs will take place via a humanoid avatar. During the preliminary stage of development, a simple desirable character following some basic guidelines, especially regarding the quality of signs visibility, i.e., black shirt was designed in order to focus on collaboration regarding recording data. Therefore, 3D computer graphics application was used where the user can choose and modify character components, such as body parts or clothes, in real-time. Figure 27 illustrates the avatar creation process.

When the design is completed, a 3D software engine is in order to use the Rigging service. Rigging is a method to create a skeleton for a 3D model by constructing a series of bones so it can be animated and move. Each bone has a three-dimensional transformation (which includes its position, scale and orientation), and an optional parent bone and therefore they form a hierarchy. So, moving a shoulder-bone will move the rest of the hand too. The bones are connected with each other through joints. The rigging allows a 3D model to be animated in an articulated manner. An articulation is a rotation/translation of a joint which moves a connected bone. On the other hand, the pose is a set of joint articulations which results in positioning the articulated body. In the case of an avatar, the rigging is the skeleton that ties into the human posture. At this stage,

technologies which use machine learning methods to automate the steps of the character animation process where used, including 3D modeling to rigging and 3D animation.

In the same software a basic facial animation with blended shapes was created, which can be used for lip sync or other face expression. Generally facial expressions can be achieved either with bone transformation or blend shapes. In the first case, bones can be moved to specific directions and the combination of whole movements form an expression. Moreover, the method is based on controlling bone rotations when moving between different positions. On the other hand, blend shapes create the illusion that one shape changes geometrically into another in a natural-looking way. BlendShape is a technique of allowing a single mesh to deform in order to achieve numerous pre-defined shapes and any number of combinations of in-between these shapes. Again, the combination of blending shapes can animate an avatar's mouth to open or smile. The result of this procedure is a rigged 3D model with blend shapes, ready to be imported in Unity 3D.

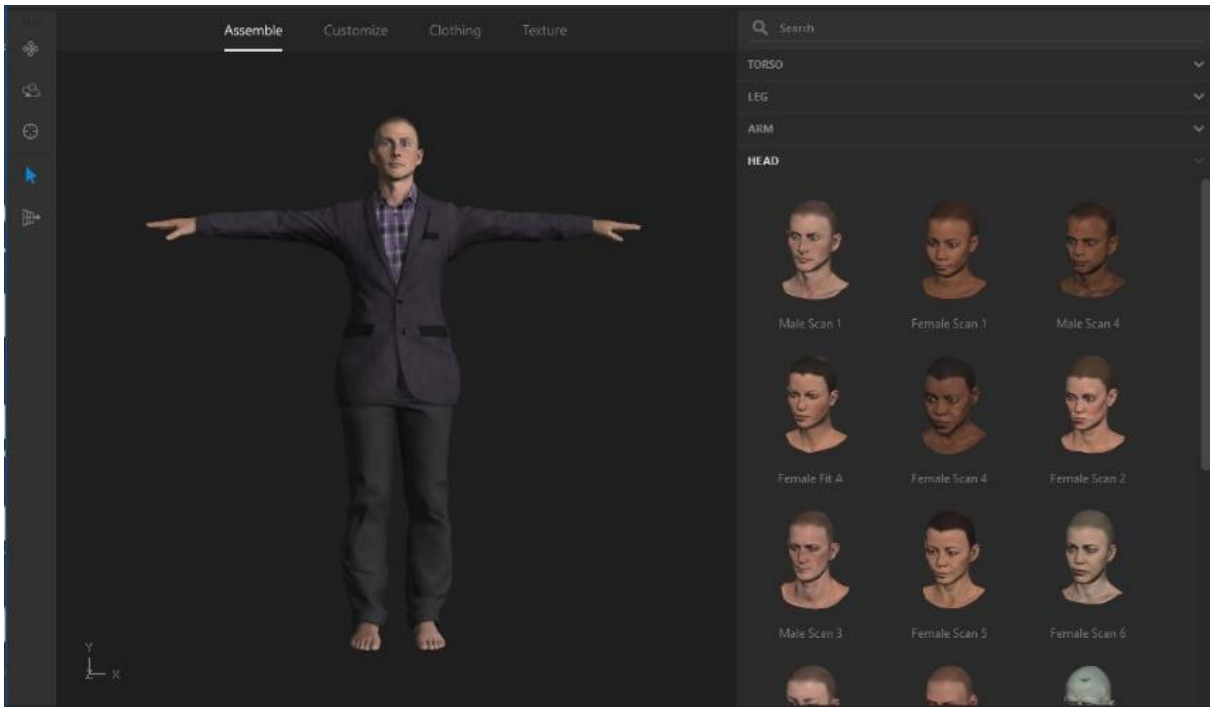


Figure 28: Adobe Fuse CC 3D Model.

The rigged 3D avatar is imported into Unity3D as an object, ready to be modified. Unity3D is a cross-platform game engine used to create both three-dimensional and two-dimensional games, as well as, simulations for desktops and laptops, home consoles, smart TVs, and mobile devices. Inside the editor of Unity3D the avatar will be modified in order to reproduce the recorded signs. Regarding avatar's rigging, few changes need to be made first in order to give a humanoid aspect to the character motion, i.e., definition of a humanoid animation type in the rig option. Files with positional data in JSON format are loaded into the avatar, pointing out target positions for each joint and specific timestamp.

Inverse Kinematics (IK) method is used at first in order to make the coordinates of each point reach a target configuration. Forward kinematics uses the joint parameters to compute the configuration of the pose, whereas inverse kinematics reverses this calculation to determine the joint parameters that achieve a desired configuration, having as main objective to reach the desirable position. The synchronization of frames per second between recorded data and reproduction is very important for a normalized movement. Even if the sync is perfect, an issue may occur regarding the physics, resulting in image jitter. In order to offer a smoother image and avoid jittering, interpolation must be applied between each different position. The whole process mentioned above consists of a group of C# scripts useful for the control for loading data in correct manner, smooth motion and

synchronization.

Despite the usage of JSON files, the main objective is to match each recorded keypoint of the body detector (as described in section 0) to each joint in the rigged body of the avatar. Generally, a rigged body in 3D modeling has a root joint which is the base of the structure. Usually for the humanoid avatars, the center of the hips constitutes the root joint. However, as it can be seen on Figure 7, there is not that kind of keypoint. Moreover, there are more keypoints recorded in the head which do not exist in the avatar. As a consequence, additional effort must be applied in order to avoid wrong reproduction of the recorded signs. An example of a signing 3D avatar is shown in Figure 28.

Figure 29: Signer avatar interprets the word "name".

In the next stages of development, improvements were made especially regarding the visual aspect of the avatar and the connection with the recording module. An updated and more human realistic texturing was introduced with the new avatar as seen in Figure 29 without affecting the final size. Furthermore, a new rigging was added matching with the updated skeleton on figure 5. So, every keypoint of the recording data in json file, as described in previous chapter, has a representative point on Avatar's rig. If there are additional bones can be ignored but also have to be tested due to their connection with texture. Figure 30 shows the rig of the avatar.

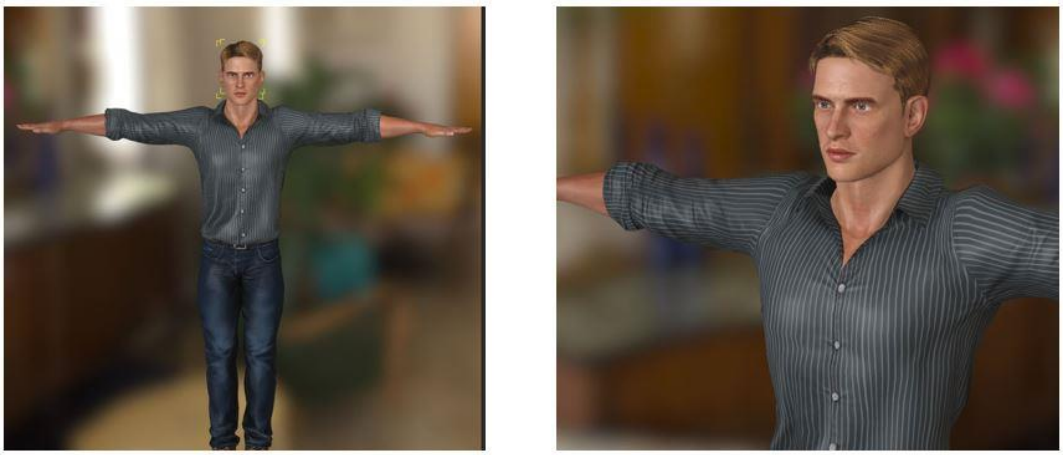


Figure 30: Updated Avatar.

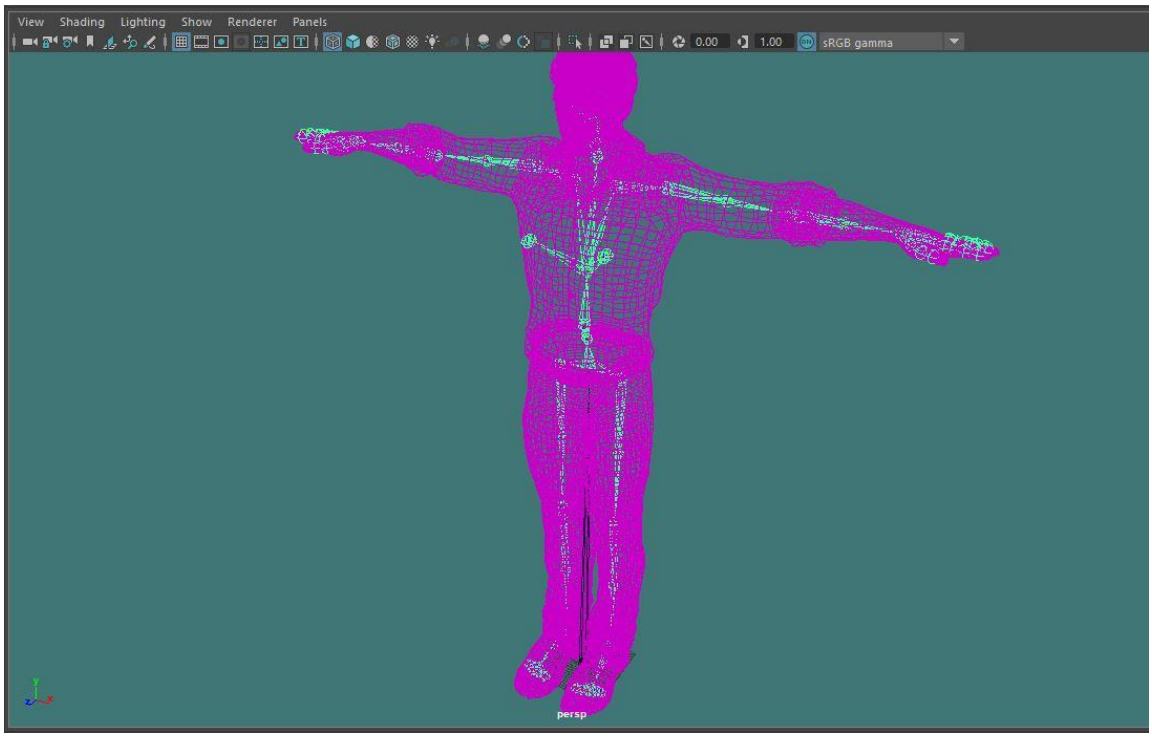


Figure 31: New Rigging.

Regarding face additional steps and a different approach of classic rigging was followed. Starting from the neck, 70 new joints were added with the corresponding facial keypoints as described in Figure 8. The other end is actually connected with avatars texture in order to move parts of the face. Figure 31 shows how the additional rig was developed and how it changes the face expressions.

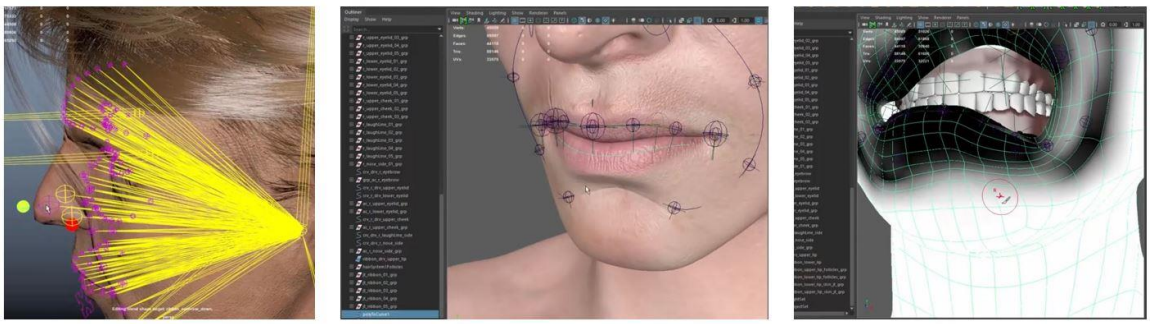


Figure 32: Face Rigging.

Following the updates of the recording module the reproduction of the recording data was separated and tested in three different parts: head, body-arms and hands with fingers. A JSON File was used for the head as the exact point of the face could be represented for each keypoint. Regarding body data an exact movement was achieved using and testing different type of files, like bvh, and taking advantage of Unity's animation system, referred as Mechanical. Figure 32 shows a frame from a sign.

Figure 33: Body and arm.

7. USER INTERFACE

This chapter presents modifications in the UI of the capturing module. These changes reflect to the feedback we got from users in the 1st Intermediate Tests of EasyTV.

7.1. Simpler UI for End-Users

We made certain re-adjustments to both screens of the capturing module in order to satisfy user requests and preferences. More specifically, for the language selection panel that initializes the application, we excluded the 'OK' button on the bottom of the screen and now the transition to the main screen is done directly by clicking on any of the listed languages. This button seemed redundant to users in the Intermediate Tests and for that reason has now been discarded. The difference between the old and the new language panel can be seen in Figure 33.

Figure 34: Differences between language selection panels.

Concerning the main screen of the application, we replaced the old UI with a much simpler one. More specifically, the previous UI included 2 textboxes and 4 buttons. The first textbox allowed the definition of a user-specified storage folder for the exporting files. Moreover, the second textbox allowed the user to type in annotations (i.e. subtitles) for the current sign recorded. Annotations were stored to standard text files in the user-specified folder by using a small button next to the second textbox. While both textboxes remained as such, in the new UI text storage functionality has changed and annotations are now automatically stored when a recording session ends.

Figure 35: Old UI of the application's main screen.

The core modification in the UI is that the functionality of the 3 main buttons has now been replaced by a single 'all-in-one' button. In the previous UI, a button served to start and stop recording, while the other two buttons supported the processing and display of the captured motion data. In the new UI, this procedure has become even more automated. Using just a single button, the user can now start and stop the recording session, while the data processing phase and the display of 3D data take place when the user clicks on the button for the second time to stop the recording session. This modification alleviates the need to provide any explanations to the end-users about how they should be using the application. The functionality is more obvious to them now with a single 'Start Recording / Stop Recording' button.

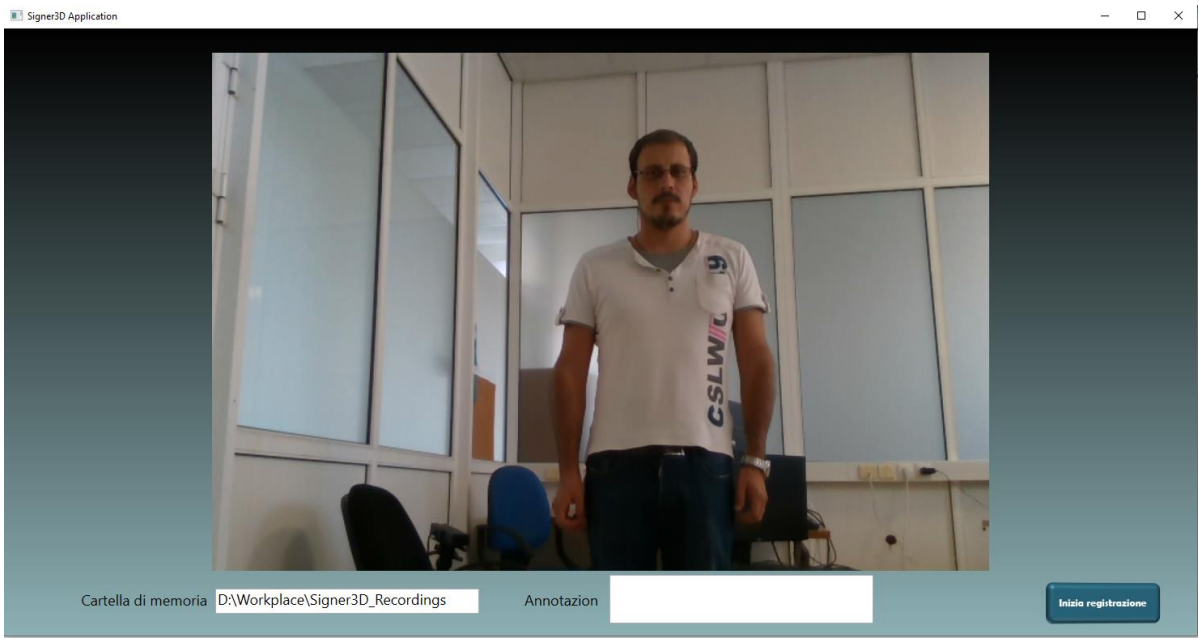


Figure 36: New UI of the application's main screen.

Another notable change in the UI is that the new layout has been made responsive. The UI's elements (sensor viewing area, buttons, textboxes and labels) can now fit any screen size and resolution and thus, allow the user to set the preferred window size for using the app. This was a basic request in the Intermediate Test since it can be regarded as a personalization feature of the application. Finally, the 'Signer3D' logo has been removed in the new UI, as well as the depth viewing window, in order to allow for a larger viewing area and shred the focus on the signer.

7.2. Multilingual Support for 5 Languages

We have implemented support for 5 languages, as mentioned in the EasyTV DoA. When starting the Sign Language capturing application, a pop-up menu is presented to the user to choose his/her preferred language (Figure 36). The languages supported are the following: Catalan, Spanish, English, Italian and Greek.

Figure 37: Support of different languages for the capturing module.

After the selection of language, the main screen of the application opens. In the main screen, labels on menus and buttons are shown in the user-specified language, although the user has to personally change the language (i.e. system language) when writing in the textboxes provided. Annotations for the recorded signs are then stored in standard .txt files, in the user-specified language.

8. CONCLUSIONS

This document presented final additions in the implementation of the EasyTV capturing technology. The structure of the EasyTV project, as outlined in the DoA, lead us to define a multi-phase architecture for the development of the capturing module; an architecture that also fits the markerless setup and aims to provide accurate and robust motion capture results for Sign Language. At the heart of our multi-phase architecture lies the keypoint detection phase, where motion analysis algorithms process the acquired frames and find specific points of interest for the hands, face, and body of the signer. It is crucial to consider the fluctuating effects that the quality of 3D data causes on realistic avatar performance. For that reason, we augmented our architecture by adding an extra post-processing step: the motion refinement phase. We believe that by developing robust refinement techniques that resolve issues with Sign Language recordings, we can reach a satisfying quality of 3D data, capable of producing natural motion in the avatar playback process.

9. REFERENCES

[1]. S. Krig, “Interest point detector and feature descriptor survey,” in Computer Vision Metrics. Springer, 2014, pp. 217–282.

[2]. S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh. Convolutional pose machines. In CVPR, 2016.

[3]. T. Simon, H. Joo, I. Matthews, and Y. Sheikh, “Hand keypoint detection in single images using multiview bootstrapping,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2017.

[4]. https://github.com/CMU-Perceptual-Computing-Lab/openpose