Final report for Avatar Ensignes

Hadrien Brochet Osmar Cedron Yoann Coudert–Osmont Antonin Dudermel
Ugo Giocanti Pierre Marcus Chloé Paris Gabrielle Pauvert Amadeus Reinald
Alban Reynaud Victoire Siguret Xuan Thang Tran Étienne Vareille

May 1, 2019

The Avatar-Ensignes project is about creating a sign processing software based on a formal representation of the French Sign Language. This project has three axes: sign recognition with video, sign processing, and sign visualization with a signing avatar.

Software Description The software we wish to create could be assimilated to a word processing software but for sign languages. This obviously means that we have to find equivalents to letters and words; the final software will look nothing like an actual *word* processor.

What we want to keep is the concept:

- the usual word processing software allows an input,
- then it enables you to edit your work without having to type it entirely again,
  - you can change each letter separately
  - and if you want you can get auto-completion for words that appear frequently;
- finally, you get a final product that is potentially anonymous.

Now, let's transpose that to sign languages:

- the input should be a video or a text file;
- editing without having to redo the entire video should be possible:
  - changes should be allowed on each parameter of sign language separately (hand shape, facial expression, placement and so on);
  - the user should have access to a "dictionary" of signs (for which there is no parameter assembling needed because the whole thing is already implemented): each user should be able to add as many signs as they want in the dictionary, we might want to create a module that would be able to gather all this data because it could be useful at some point.
- the final sign language speech should be executed by an avatar so that it is anonymous; an added feature could be the possibility to change the aspect of the avatar, just like one can change the font of a written text.

A database containing video of signed words will allow people to provide data in order to improve the recognition (see Figure 1).

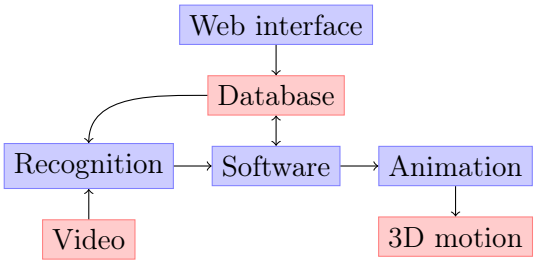

Figure 1: Software Overview

1 WP0: Communication

Team members: Victoire (leader), Antonin, Chloé, Alban, Amadeus

1.1 Website

The website is online at `http://ensignes.org` and some pages are under construction, but we have to wait for some other teams. We wrote articles, both in French and in English, a description of the work done by each team... We need to have a privileged way to contact French Deaf people in order to gather data from them; but we also want to be understood globally. This is why we chose to have the two languages available. As for contacts, we sent a lot of emails -especially to associations and researchers- but none of them answered. So we planned in the last meeting to call associations in person and to go to their headquarters.

1.2 Database

One important step in this project is to make a collaborative database with the help of signing people. We want them to be able to upload hand-made or software-made annotated videos. However, before we even think about the implementation, the first step is to solve legal issues: in order to store and use videos of people, we need them to give us a “Cessation de droit à l’image”. We also need to be complient with the new General Data Protection Regulation. We will contact the "Commission Nationale de l’Informatique et des Libertés" for more information.

2 WP1: Artificial Intelligence

Team members: Etienne (leader), Osmar, Hadrien, Thang

Team’s goal The team aims at correctly extracting the meaning of the sign made by a user in a video format.

Introduction To better understand the use of artificial intelligence in the project, we first need to introduce the concept of machine perception, which could be defined as the capability of a computer system to interpret data in a similar way as humans do through their senses.

Such a wide area of knowledge includes machine vision and more specifically image/video processing, which through a series of techniques can extract useful information from any visual format.

In our particular case, complex series of gestures which belong to any particular sign language will be transformed into decodable word sentences in a matter of seconds without the help of a sign language expert and therefore it is more adaptable to the final user, which will upload his message to our webiste in video format.

Division of work Based on the existing literature, we chose different tasks to be performed according to their importance and relevance to the final goal. Currently, our work is divided between the following tasks.

- Skin segmentation, to detect which part of an image is skin and which is not;
- Hand tracking, to follow the movement of the hand;
- Dimensionality reduction, to find the right tradeoff between quality and information loss so any image can be reduced to an optimal value to be used as a training input for our neural network;
- Image pre-selection, to select by light analysis a smaller number of images (less than 30 fps) on which to perform the whole analysis process.

Each task has been assigned to a team member, along with collaboration including in particular feedback, suggestions and corrections.

Current status The team has done good progress overall. Skin recognition, a critical part of our project, is implemented and shows appropriate results on a small dataset of chosen examples. In a pretty similar way, hand-tracking has been correctly documented and is hopefully ready to be implemented. Furthermore, dimension reduction has been tackled through pixelization and finger segmentation. Last but not least, image pre-selection stumbled upon some problems while dealing with too many frames. In order to deal with this issue, different approaches were used ranging from naive methods to smarter ones.

Re-evaluation of the team's goals The complexity of recognition of a Sign Language, and of image processing as a whole, made us review our goals. The biggest difficulty lied in the robustness of the analysis. The lack of a widespread database for Sign Languages led to focus most of our work on restricted and often self-made datasets, but which almost always have specific acquisition conditions. As a consequence, once we complete the implementation and interaction between each module of the gesture recognition, we will certainly have to review and tweak our algorithms to ensure enough robustness to be production-ready.

2.1 Skin Segmentation

Many different approaches to sign recognition try to break down the image into smaller chunks of meaningful information, before applying more complex techniques such as Hidden Markov Models (HMM) or more recently Neural Networks. Following previous working approaches, we chose to extract the parts of the image containing skin to make it the main components of our feature extraction algorithm.

Based on their simplicity and time efficiency, threshold techniques were chosen to fill this goal. They basically come down to creating a mask that specifies a range within which we recognize a pixel as skin. The first implemented skin segmentation algorithm, which used HSV color space, performed poorly in scenarios where the background image was complex.

At that point we found an approach similar to the one we already used. It consisted in using two color-spaces instead of just one, to increase the performance and efficiency of the skin segmentation. After evaluating different alternatives, an algorithm combining the YUV and RGB color-spaces was implemented.

This technique does not only outperform many other techniques found in the literature, but also doesn't add much complexity to our previous base code. In the following paragraphs, the definition of this technique, the implementation details and the results of the algorithm [1] will be specified.

2.1.1 Definition

First and foremost, to better understand how it works, we should talk about the YUV colorspace. The acronym stands for luminance (Y) and the chrominance components (U and V). Separating luminance from the chrominance helps reducing the effects of lighting variations [1]. However, it doesn't allow to represent as many colors as its RGB counterpart. Thus, we naturally found the existing approach of mixing both color-spaces plausible and aligned with our needs.

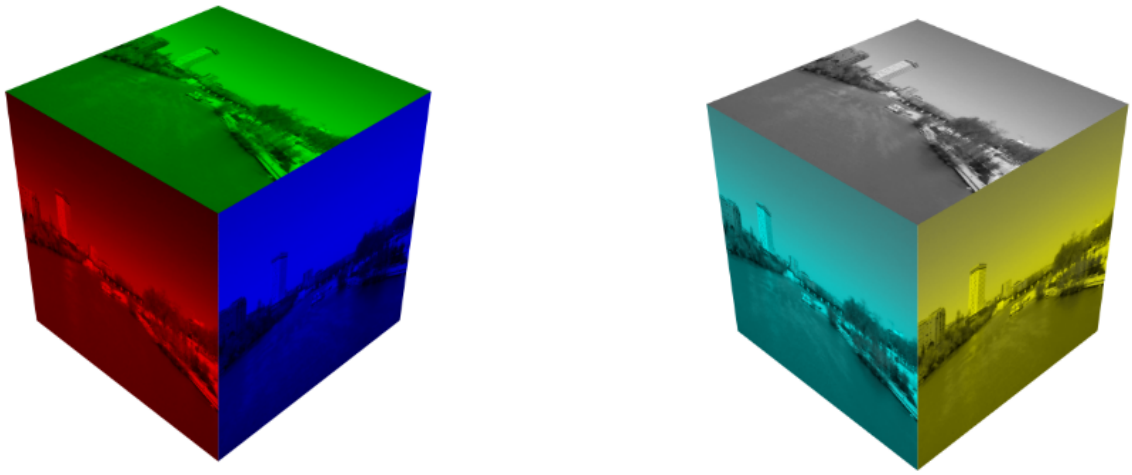

Figure 2: RGB (Left) and YUV (Right) colorspaces [2]

As a side note, the YUV color-space (which was originally used for TV transmission) is widely used as an output format of many web-cams.

2.1.2 Implementation

A threshold algorithm works by separating a pixel in an image by assigning it to one of two different classes just by following some condition. In image processing, there exists direct comparisons with a fixed value (although there exists some adaptive alternatives). Since this algorithm was similar in type to the former one, the main difference consisted in extending its use to a second color-space and choosing the right threshold values to detect skin color more efficiently.

[1] establishes such values as:

$$\begin{cases} 80 < U < 130 \\ 136 < V < 200 \\ V > U \\ R > 80, G > 30, B > 15 \\ abs(R - G) > 15 \end{cases} \tag{1}$$

The pseudocode of the algorithm could be specified as follows 1:

The implementation of the algorithm can be found in this repository `https://gitlab.aliens-lyon.fr/avatar-ensignes/gesture-recognition`

2.1.3 Coding practices

To ensure the quality and the adequate formatting of the code, two different code quality tools have been used so far: Black (an opinionated style formatter) and flake8 (which follows the guidelines of PEP8). Both are executed automatically at the moment of doing a commit through a pre-commit hook.

Algorithm 1 Skin Segmentation

```
function SkinSegmentation(img, img_weight, img_height)
    Preprocess(img, width < 400px)
    Normalize(img)
    skinmap ← NewImage(img_weight, img_height, yellow)

    for all pixel in img do
        R, G, B ← pixel
        ConvertToYUV(R, G, B)
        if IsSkin(R, G, B, Y, U, V) then
            skinmap[pixel_x, pixel_y] ← purple
        else
            skinmap[pixel_x, pixel_y] ← yellow
        end if
    end for
    return skinmap
end function
```

2.1.4 Results

For the implementation of this algorithm, we used a docker instance with OpenCV-Python installed on it. The implementation takes advantage of the numpy matrix operations to quickly apply the masks and constraints. Each example corresponds to a different sample:

1. RWTH Phoenix (easy instances 3) [3], that gathers a large vocabulary corpus of German Sign Language gathered from national German television;
2. HGR (medium instances 4) [4] [5] [6] that contains the gesture from Polish Sign Language;
3. Self-collected images (hard instances 5), that represent very random inputs where the background tends to make the segmentation difficult.

Knowing that the final data samples on which we will work on are not yet available, these tests are given as indicative results.

2.1.5 Conclusion

Despite some good results for the skin segmentation in our examples, robustness needs to be tested on an actual database, in fact, it is quite probable that applying more tweaks to the constraints will be needed. Also, the integration of other analysis techniques and their testing remain to be done.

2.2 Movement Tracking

This function is still unimplemented. The high difficulty of this task made mandatory an in depth review of the state of the art. From this, we imitated some approaches and designed a candidate algorithm on paper.

2.2.1 Overview of the research

The tracking algorithm we will discuss later on uses the common assumption that the considered object (tracked hands) is moving most of the time. Some results and methods using hand recognition and tracking have been done :

Figure 3: RWTHPhoenix Dataset

Figure 4: HGR1 Dataset

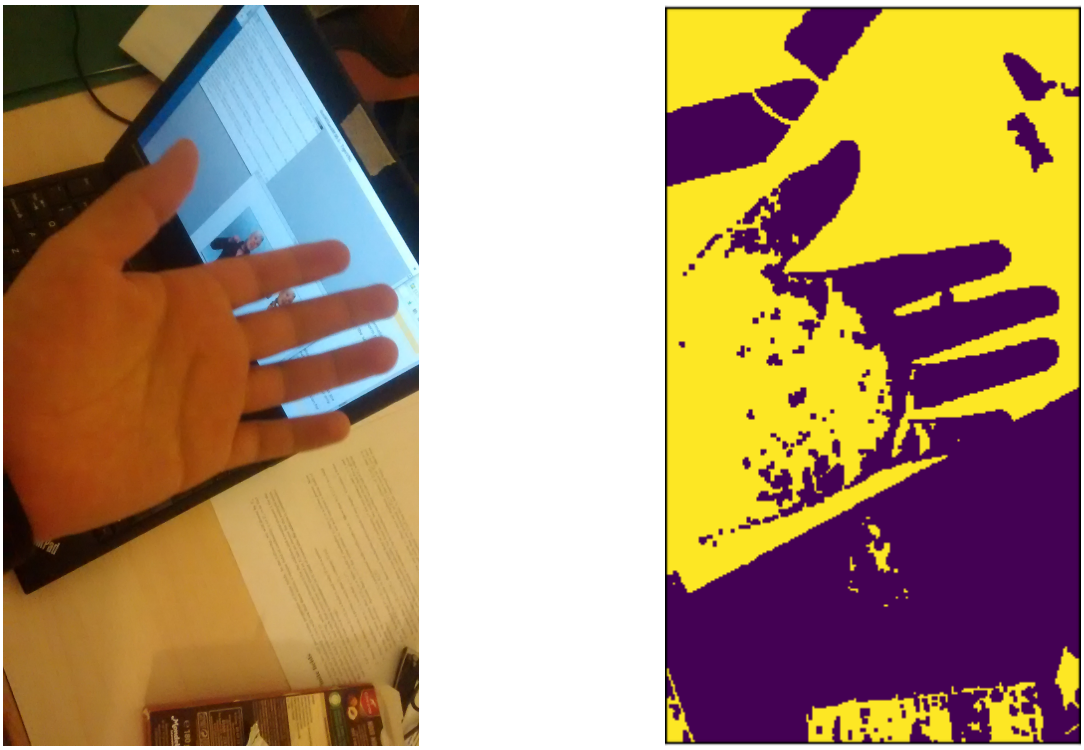

Figure 5: Self Collected images

- Allen et al [2006] extended the camshift algorithm. Camshift was used by the authors because, according to them, it consumes the lowest number of CPU cycles assuming that there is only a single hue in the color space model. The result showed that camshift-based trackers fail unless weighted histograms are used. If they are used, then some noise such as snow and sleigh colors could be eliminated.

- Patwardhan and Roy [2007] designed the novel tracker capable of hand tracking based on position, shape, size and appearance. Their tracker could successfully track the hand gestures in spite of change in shape and position during tracking.

- Elmezain et al. [2009] proposed mean-shift analysis and Kalman filter method for tracking. The algorithm they proposed can track shapes in complex background. In their research, using 3-D map as depth information can solve the overlapping problem. Hence, face and hands can be easily distinguished.

- Rautaray et al. [2012] surveyed most of the previous papers of interest and summarized the overview of hand gesture recognition. All of the basic techniques included in these phases are discussed in detail. Then they described the application domains of human gesture recognition. The paper itself was a survey paper for other researchers.

The method we choose finds its inspiration in the above works.

2.2.2 Mean-shift Algorithm

Mean-shift is a procedure for locating the maxima of a density function given discrete data sampled from that function. It is useful for detecting the modes of this density. Given a set of datapoints, the mean-shift algorithm iteratively assigns each datapoint towards the closest cluster centroid. The direction to the closest cluster centroid is determined by the locations of the majority of nearby points. So at each iteration, each data point will move closer to where the majority of the points are, which is or will lead to the cluster center. When the algorithm stops, each point is assigned to a cluster.

The mean-shift is different from *K-means* algorithm because the mean-shift does not require the number of clusters beforehand. This number is determined by the algorithm with respect to the data.

We hope to use the mean-shift analysis to measure the similarity between a hand target and a hand candidate that is localized in the next frame. We hope that the use of the mean shift algorithm will at least enable to distinguish each hand, the background and the body. The distance between points might be corrected using skin color segmentation.

2.2.3 Kalman filter

Kalman filter is the optimal filter (in the least mean squared error sense) for track prediction. It is a combination of a predictor and a filter:

- The predictor estimates the location of the object at time k given $k-1$ previous observations.

- Reduction of noise is introduced by inaccurate detections.

- When observation k arrives, the estimate is improved using an optimal filter to estimate the target position at time $k+1$: the filtered estimate is the best estimate of the true location of the target given k observations at time k.

2.2.4 Tracking algorithm

The main research is, given an initial labeling of a hand and of a face, to find the position of the hand and face in the following images. We may suppose that the tool for skin segmentation is available at this point (so basically, we have both the original images and a black and white mask that separate skin and other components). We use the Kalman prediction after each mean-shift optimization, which gives a measured location of the hand target. The uncertainty of the estimate can also be computed and then followed by the Kalman iteration, which drives the predicated position of the hand target.

Initialize Tracks We need a function to create an array of tracks, where each track is a structure representing a movement in the video. The purpose of this structure is to maintain the state of a tracked object. This state consists of the information used for detection of tracking assignment, tracking termination, and display. The structure must contain the following fields : the ID of the track, the current bounding of the object, a Kalman filter object used for motion-based tracking and some information like the number of frames since the track was first detected, the total number of frames in which the track was detected and the number of consecutive frames for which the track was not detected.

Detect object This was a difficult part. The goal of this part is to separate the image into clusters, which should represent objects. The choice of the clustering algorithm is essential, as we want some continuity along time. Still, using the skin color mask, we have good hopes of being able to separate parts of the image that correspond to the hands.

Predict New Locations of Existing Tracks In this part we use the Kalman filter to predict the centroid of each track in the current frame, and update its bounding box accordingly. This is the most critical part of the algorithm, as it is the one which does the tracking.

Assign Detections to Tracks This function uses a version of the Hungarian algorithm to compute an assignment which minimizes the total cost. It returns an M x 2 matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of tracks and detections that remained unassigned.

We note that some tracks and detection might not be assigned. We prefer not to assign a track rather than to blindly use an uncertain value in the future. Since the track that interests us is the hand track, we can do this without much problem for all the other tracks. For hand track, we might have to force the correspondence, or if the problem doesn't happen too much, to require the user of the software to specify themselves the position of the hand.

An unassigned track might be deleted if too much time passes without assignment to it. Similarly, we can create track for unassigned detections.

2.2.5 To conclude

A tracking algorithm exists, and the implementation seems feasible. We will have to implement it in the future.

2.3 Dimension Reduction

We identified machine learning as the surest way to classify hand gestures. Most research papers we reviewed used either Hidden Markov Models or Neural Networks. The advantage of machine learning against specific hand-designed classification is that when there is enough data, a machine-learning algorithm is more robust than a man-made classification program. This is especially true in our context. It is highly

difficult to design classification rules by hand, as the object we want to classify has a high number of shapes and dispositions for a same gesture.

However, there are demerits to this method. The first is that we need a large dataset, with enough variety to encompass most situations. The lack of a pre-existing database for the LSF (French Sign Language), and the difficulty to produce a big dataset ourselves, hinder our use of machine learning. The second demerit is that the training time on large datasets is usually huge, and might require more computing power than we have.

All the above-mentioned reasons make dimension reduction necessary.

2.3.1 Definition

The goal of this field is to, given the image of a hand, reduce it to a small number of descriptors.

The image of a hand usually consists in more than ten thousands pixels. Thus, a machine-learning algorithm needs to have enough data to "understand" pixel variations and correlations. Usually, image classification bypass this problem because the objects classified are very different. But here, we have to differentiate objects of the same color, and with some gestures very close to each other.

We thus tried some methods to reduce an image to a smaller number of parameters, on which we can apply machine learning. We note that the usual models for dimension reduction try to reduce the whole dataset, making a smaller dataset of smaller dimension. Here, we only work on individual images. What we want is a representation of the image of a hand, that can be applied to any image.

2.3.2 Pixelization

The first idea to represent an image with a lower number of descriptors is to use the same image in lower quality. This approach is promising. As we can see in the image, the shape is preserved, and the separation between fingers can still be seen after reduction. This remains to be tested for a wider range of images, which was made difficult by the lack of larger data and since the previous steps of the processing weren't operational yet.

We observe that the reduction takes an image of 50000 pixels and creates an exploitable image of 500 pixels from it. It is a reasonable number of descriptors, if we were to have a reasonably large dataset. Still, there are two drawbacks: the loss of precision might not be robust in the general case, and we did not extract meaningful information. This is why we tried to use another method conjointly.

2.3.3 Finger Segmentation

Our goal in this section is to take the image of a hand and extract the position of the fingertips without use of ML. This is still a work in progress, as the simplest approach failed.

The first problem is that finger extraction is heavily dependent on the orientation of the hand. We have to find the orientation of the hand for most application of finger segmentation. A good idea would

be to locate first the palm or wrist, and hopefully the fingers will be at the opposite side of the image. This approach has two problems. The first is that for certain configurations of the hand, the opposite direction is not the right direction. Still, if the direction found is the same for all images of a given hand configuration, this might not be too serious of a problem. The second problem is that this approach is dependent on the bounding box around the hand. Without accurate bounding, this cannot be applied. This task is differed to a later date for now. Those problems are the main reason why what is implemented does not work at the moment.

To compute the palm position, one can compute the widest convex zone of the image, and take its center. This needs too much computation power, so what was done is take the widest horizontal rectangle fitting in the hand shape. To avoid nonsensical results, the ratio between the highest and lowest diameter is bounded. We then just need to take the center of the zone to have an approximate center of the palm. In a few examples, this indeed detected an appropriate zone.

We can also find the position of the wrist. We basically just need to find the biggest intersection of the hand and of the border of the frame. It does not allow us to extract the position of the palm center, but is less subject to noise and variations of the image.

Once this is done, we can detect fingers. We are currently trying to implement some techniques from [7] and [8].

2.4 Images pre-selection

In order to analyze the video, we decompress it into a sequence of images. However the number of images generated from a video is way too big and has a lot of redundancy. Our task is to extract meaningful and sufficient images with light computations, to lessen the amount of computation on the whole.

So far, we opted for a naive approach. To reduce the number of images, we select only one image every 5 images (the parameter can of course be changed). If the sequence of image happens to be too big for the next steps of our work, we might consider developing a smarter algorithm, although this seems to be a challenging problem. Indeed, if we want to keep only the images that differ considerably one from another, we could miss some small changes on the pictures like a hand gesture. This is especially hard since this step happens before any processing step on the pictures.

A further refinement could be achieved through the use of hand tracking. First, we give the hand tracking method on one over four/five frames, naively. Then we could identify the moments of low velocity, which are likely to be static gestures. We might then lessen the number of frames needed to analyze the sequence.

3 WP2: 3D Animation

Team members: Pierre (leader), Yoann, Ugo, Hadrien, Amadeus, Thang, Etienne

Team's goal and organization of work This team is in charge of the 3D animation, i.e. the rendering of Sign Language with an avatar.

For this, we have two goals and we split our team into two sub-teams according to them:

(i) model an avatar and design the basic poses and animation;

(ii) work on switching positions to produce Sign Language.

Method To start with, we only focused on one hand animation. It allows us to test our tools on a smaller scale. Animation is new to us, so we needed to break it down to smaller parts on which to succeed first in the project. Here is the method we followed:

1. produce the French Sign Language alphabet with only one hand;
2. add the second hand and coordinate it with the first one;
3. focus, in this order of priority, on the face, the chest, the neck and finaly the elbows.

3.1 Shapes and Animations in Blender

Yoann and Pierre were in charge of this part.
To achieve the first goal, we decided to use Blender as a modelling software. It has an active community and therefore many game engines can use models made in Blender. Plus it is multi-platform. An advantage of Blender is that it is under the license GPL. Thus, any data produced with Blender can be used freely. The goal of the sub-team Blender is to create different parts of the human body in 3D and to associate animations to them. However, as stated earlier, we decided to work on only one hand first.

A simple way to do that without having to read the whole and vast Blender documentation, is to find made-up (or partially made-up) skeletons on the web. They must be sufficiently articulated for us to be able to add a huge list of animations. We tried to find such models but it appared to be quite difficult to find. This is why Yoann created and rigged a model for the hand in Blender (i.e. added bones in the hand). To use it in Godot, we needed to export it to a format named Collada. The Godot website provides a specific Collada exporter that properly converts a Blender file into one adapted to Godot.

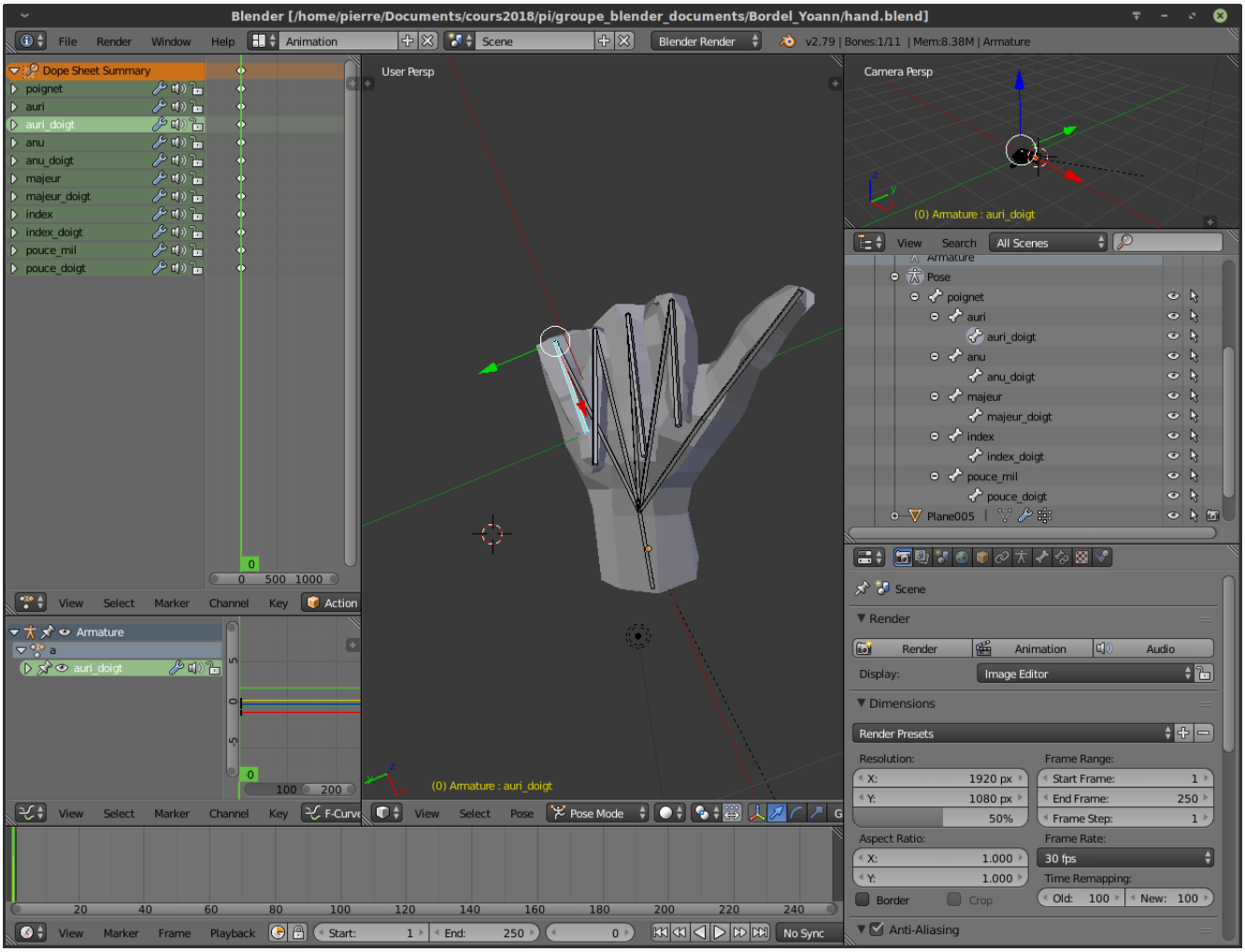

Figure 6: Yoann's hand in blender

3.2 Animation, Movements and Coordination in Godot

3.2.1 What is Godot?

Ugo and Pierre were in charge of this part. Ugo focused more on understanding the functionalities of Godot and teaching it to the other members of the group. Pierre worked on running the animations that were conceived in Blender. The first goal was to understand the global operation of a game engine and to learn how to use it, especially the 3D part. Our choice was to do this in Godot, a game engine that is free, opensource and very well documented (see `http://docs.godotengine.org/en/3.0/`). The first part of the work consisted in building a bibliography about the software. Some useful notes and links were put in the dossier Notes.

Godot is designed to be used with Blender. It allows the user to code their scripts with many different object-oriented languages such as C++, C# or GDScript. GDScript is very close to Python, but designed for scripts in Godot. We chose this one for the code.

Similarly to Unity or other game engines, Godot works with a system of Scenes and Nodes, and we had to organize the project taking all this into account. Indeed, we had to choose to work with a not very realistic humanoid, as Godot is primarily designed for video games. The best for us was to work separately on the different parts of the body, divided into different scenes (like different players in a multi-player game). The main drawback of this choice is that there is a risk that the final result will be less

realistic. Indeed, it might look like a cartoon avatar, with floating hands and a large head. We will have to be sure that the signing of our avatar is understandable. This choice took us a long time, because we first had to be familiar with the concepts of Godot. The main advantage of dividing the work this way is that we will be able to gradually add different limbs to the avatar. Moreover, it allows the two sub-teams to work in parallel: we can begin to work on a specific scene in Godot, without having to wait for the entire body to be created in Blender. Thus, this choice gives us a huge scheduling advantage. Another great advantage of Godot is that it enabled the format team to code the user interface. For a more precise idea of our work, take a look at our git deposit, that you can clone with the following link: `git@gitlab.aliens-lyon.fr:avatar-ensignes/groupe_blender_documents.git` The TODO.md shows how we divided the tasks. The README.md explains the organization of the project.

3.2.2 3D Animation in Godot

Our first goal in 3D animation was to become familiar with the 3-dimensional animation tools provided by Godot. We first learned to move one hand in space with simple movements (translations, rotations) (pictures 7). Though this part was not the most difficult, it was an essential step. Indeed as Godot was designed for games, it has a distributed philosphy and it is thus quite easy to work with separate scripts, where one script is used for one scene (here the scenes are the parts of the body). In the same time we succeded in playing the animations that the Blender Team had encoded, by using the Blender exporter BetterCollada. For instance the following screen-shots depict our model in a idle pose and signing the letter A.
The second step in our work was to control two hands in the space. We could deal with the problem in the same way a game deals with two players where each one moves one hand in the space to obtain feasible and satisfying results (see screenshot 8). Difficulties arised when we took the decision to centralize and coordinate each movement within a single main script. This is why the harder part of the project is the coordination of the limbs. We also fear that the avatar will do "infeasible" moves, when we will use the 3D functions of Godot. To deal with these questions, we need to work with the software team (see translation part).

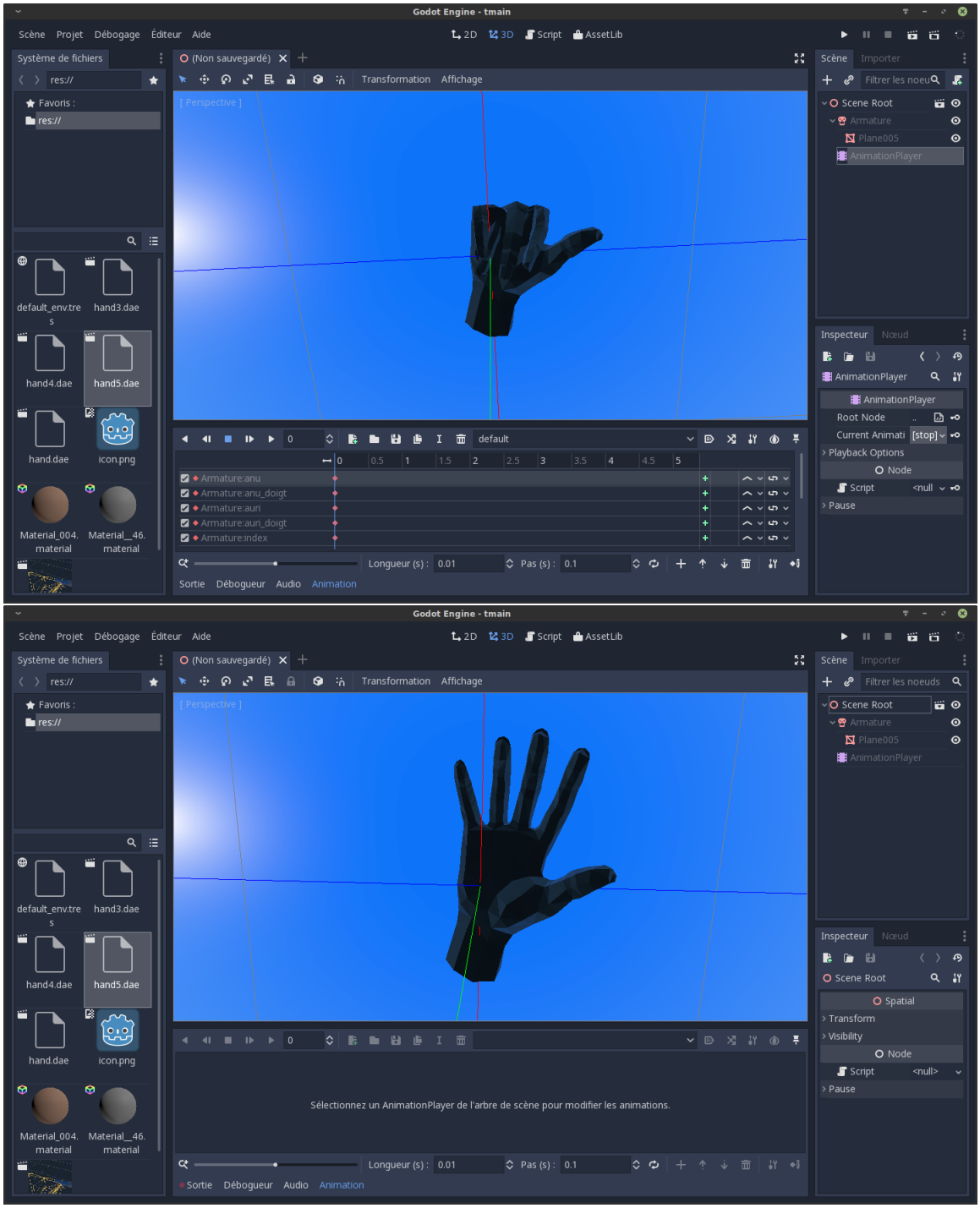

Figure 7: Godot interface

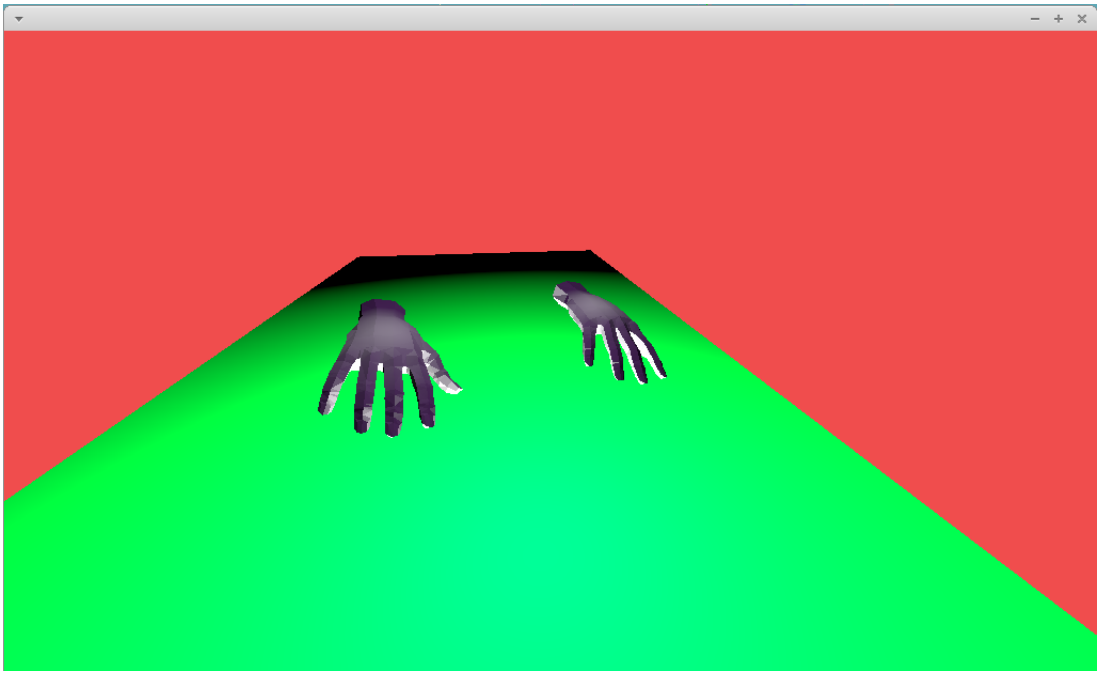

Figure 8: Moving Hands

3.2.3 Translation step: from yaml to Godot

The most important part of our work was to succesfully translate informations given by the format team into 3-dimensional animations that will be observed by the user. We thus needed a very detailed description of signs and movements. That is what the format team succesfully did with their yaml encoding of signs. They took a lot of decisions taking care of our difficulties and consulted us a lot of time. Reciprocally they helped us each time we needed from them little modifications of their work so we would like to thank them for all of their considerations. We had to work from their general yaml format files, which was feasible as yaml can be used easily in Godot scripts (see `https://godotengine.org/asset-library/asset/254`). We had to parse all the information of a given file and to play them with Godot tools. Here we encountered a lot of problems and a part of them are still present. Here is what we succeded in doing:

- Play the animations of the scenes that were encoded in Blender (initial and final).
- Interpolate these animations.
- Get the spatial coordinates of each part of the body that are indicated in yaml files and work with them.
- Make translations in the space.

The principal goal that we have to reach now are the following:

- We did not succeed yet in placing each point precisely. The problem(s) might be the following: 1/ We need to refine our hands movements by changing the points that we want to take as referential for each different movement accordingly. 2/ There might be an error somewhere in our code...
- We have the same problems with the rotations of the hands.
- We did not take in consideration the speed of the movements (however this part can be easily solved, we just need to do this after solving the two previous points).

Here (9) is a picture of what we obtained as a rendering of the Avatar (who is playing a sign). We can remark that we may have missed a design team...

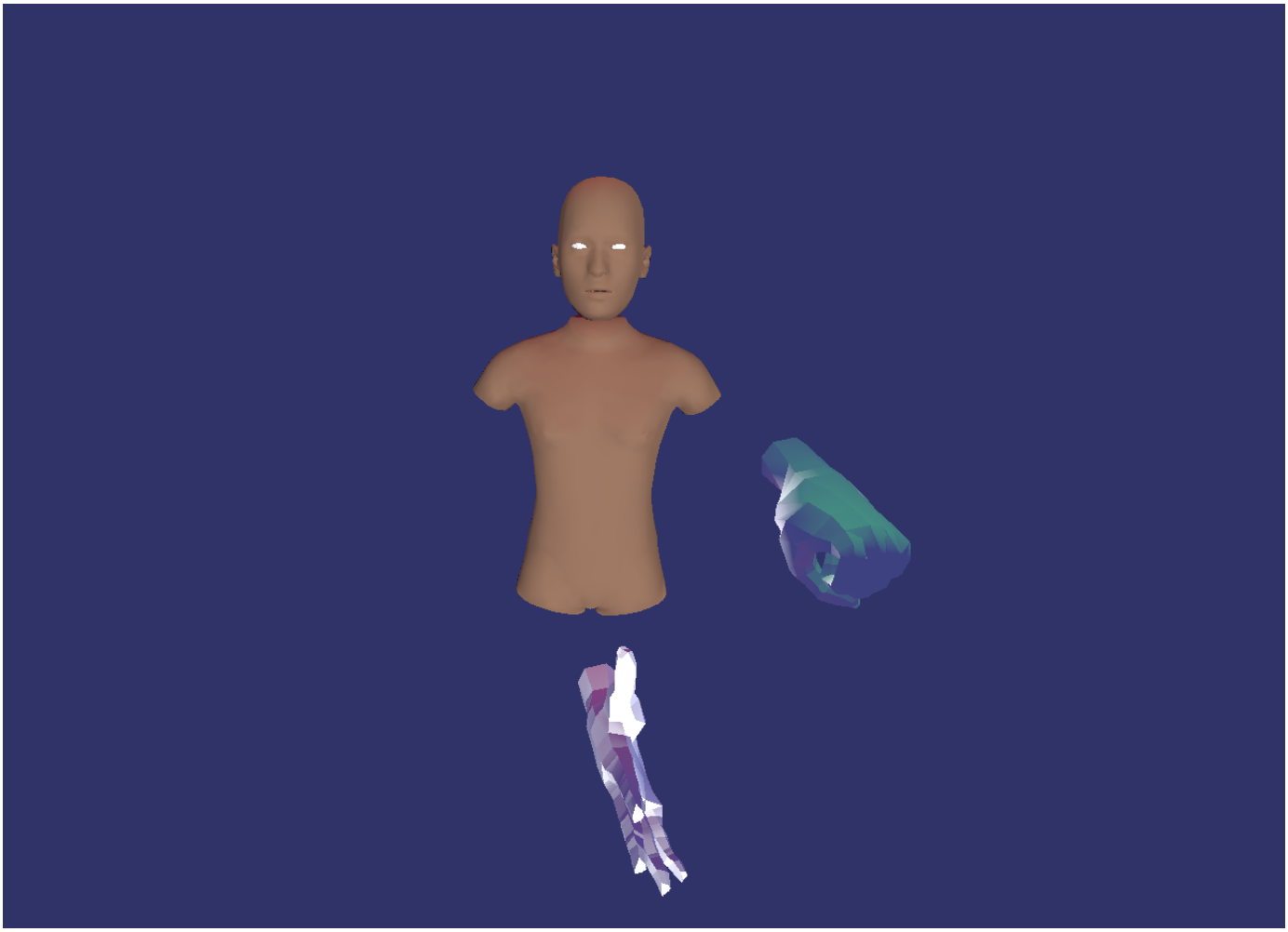

Figure 9: Prototype of the avatar.

4 WP3: Core

Team members: Chloé (leader), Antonin, Alban, Gabrielle, Victoire

Team's goal The aim of this team is to unite all work packages around a user-friendly software that allows Deaf (and potentially illiterate) people to easily input or modify signs with mouse and keyboard.

Division of work This Workpackage is divided into three main tasks:

- Invent a digital way of representing French Sign Language.
- Build the software and its interface.
- Integrate others workpackages into the software.

Team Progress

4.1 Yet Another Format to Describe French Sign Language

Disclaimer: We give here an outline of the format used to represent French Sign Language in the software. The study of Sign Languages and their representation is a rich topic in Linguistic, far beyond our understanding. Therefore we don't guarantee the accuracy of everything here from a linguistical point of view, though we took their work carefully into account in order to create our own format.

4.1.1 Challenge

The aim of this format is to find a non-ambiguous way to discriminate two signs with different meanings. But sign languages are not easy to transcript into a written form. While spoken languages are mainly a succession of phonemes, to represent sign languages, one has to deal with simultaneity and spatiality (particularly describing planes in 3D). Some formats have already been created to represent french sign languages [9, 10].

However, this format must answer many demands to cope with all parts of the software. It must at the same time be

- compact, to allow the interface to be user-friendly;
- intuitive, two similar signs must be as close as possible, to allow auto-completion;
- precise, this format is not for humans but for the animation engine of the avatar.

Finding such a perfect format is impossible (we can not have the cake and eat it) so we had to make tradeoffs between all these desired aspects.

4.1.2 Overview of the format

general structure A sign is defined as a succession of states linked by a movement that describe the hands during the sign. Besides, we add a description of the shoulders and the head.

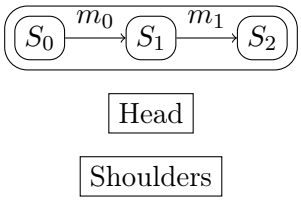

Figure 10: Structure of the format

States The states are quite easy to describe. A hand state is the position, the shape and the orientation of the hand. We only had to collect an exhaustive list of all hand shapes in French Sign Language and to discretize the possible orientations and positions. Face and shoulders are described the same way, using an exhaustive list of possible configurations.

Movement Describing a movement is more problematic. We originally intended to have an exhaustive list of all possible movement shapes, but this list became quickly very big.

However, every move can be described as a succession of straight or curved lines, for instance the leftmost curve, that can be found in `[respect]` is a downward curve, followed by a diagonal line. Atop of that, some signs are clearly a combination of many "subsigns", *e.g.* in `[chantilly]`. This lead to the succession of states in the format.

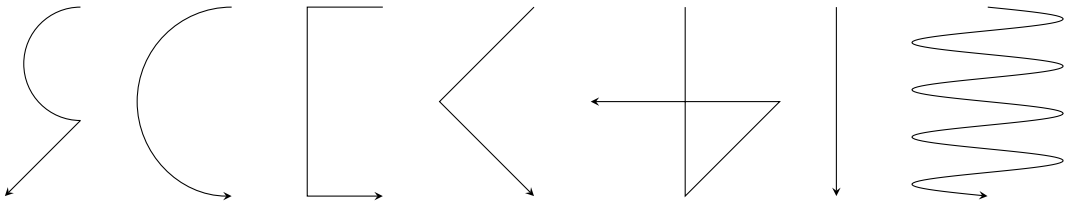

Figure 11: Movements shapes (non-exhaustive)

We also make a distinction between the *inner* and the *outer* move. Indeed, we cannot hope to have sinusoidal moves or spirals (as in `momie`) with just straight lines. Modifications also occurs sometimes not on the whole movement, but only on the hands (e.g. `[difficile]`, only the movement of the fingers is repeated). The *outer* move is the global movement that goes from one state to another. It can be straight or bowed, and can be repeated. The *inner* move concerns the little movements occuring during this big movement: the handshape change and the ondulation of the global move.

4.1.3 possible extensions

One missing thing in this format is the symmetry. This is very frequent in French Sign Language, and having an efficient way to represent symmetry would spare half of the work in many signs. However, we did not manage yet to find a way to simply express symmetry.

Another missing point is the use of the signing space. To refer to older elements in a signed speech (as done in French with pronouns), signing people place them in their signing space and use their position to refer to them (for instance by looking at them). This important point is not implemented in this version of the format, and we hope to find a way to do so.

4.1.4 Comparison with an existing format: HamNoSys

Although this system has been inspired by HamNoSys[10], it differs in many points from its digital implementation (SigML). The aim of HamNoSys is indeed to give a way to represent every Sign Language and also other body representations. The aim of this format is not only to encode the signs as simply and widely as possible, but also to animate them easily. An application[11] that animates 3D avatars with SigML input already exists, it is however copyrighted. A traduction between this system and the above representation could be an interesting experiment to compare the expressivity of both systems.

4.2 User Interface

We also worked on the user interface. Once the parameters and the way they are embedded were established, as well as the aspect and functionalities the user interface should have, we had to actually create it.

The interface was implemented using Godot. Initially, we were thinking of using a more classical language to do so. But the animation team brought to our attention that Godot would allow us to make a better work on this, even though the learning curve is actually not great in the beginning. On top of that, choosing Godot is really beneficial in terms of putting together the whole application: we are able to insert the work from other teams very easily by just using their scenes as children of ours.

At first, we created an interface that had all the desired functions but no care was put into the ergonomy or the easy use of these functions. We needed a first usable draft in order to better understand what needed to be improved. Once we obtained results from the animation team, we tried to use them in the interface, which opened new horizons. To this point, the interface was more about trying to find optimal ways to edit signs and signed sentences in an interface, we thus tried different ways to present it

to the user. Now that we have a more precise idea of the efficient ways to let the user interact with the editing, we can start focusing on the ergonomy and actual design of the interface, which is our next step.

To start with, the "sign-editing" interface was created (see figure 12):

On the top, you can enter a name for this sign, explicit the type if it has a special type (only one hand is used, there is some sort of symmetry between the hands, this is a way to simplify the sign editing that otherwise has many parameters) and some functions to open a sign or save the current sign. In the middle, two buttons send the user to another part of the interface where the parameters can be edited, we will see this part of the interface in the next figure, there is also the possibility to define the movement and potential repetition.

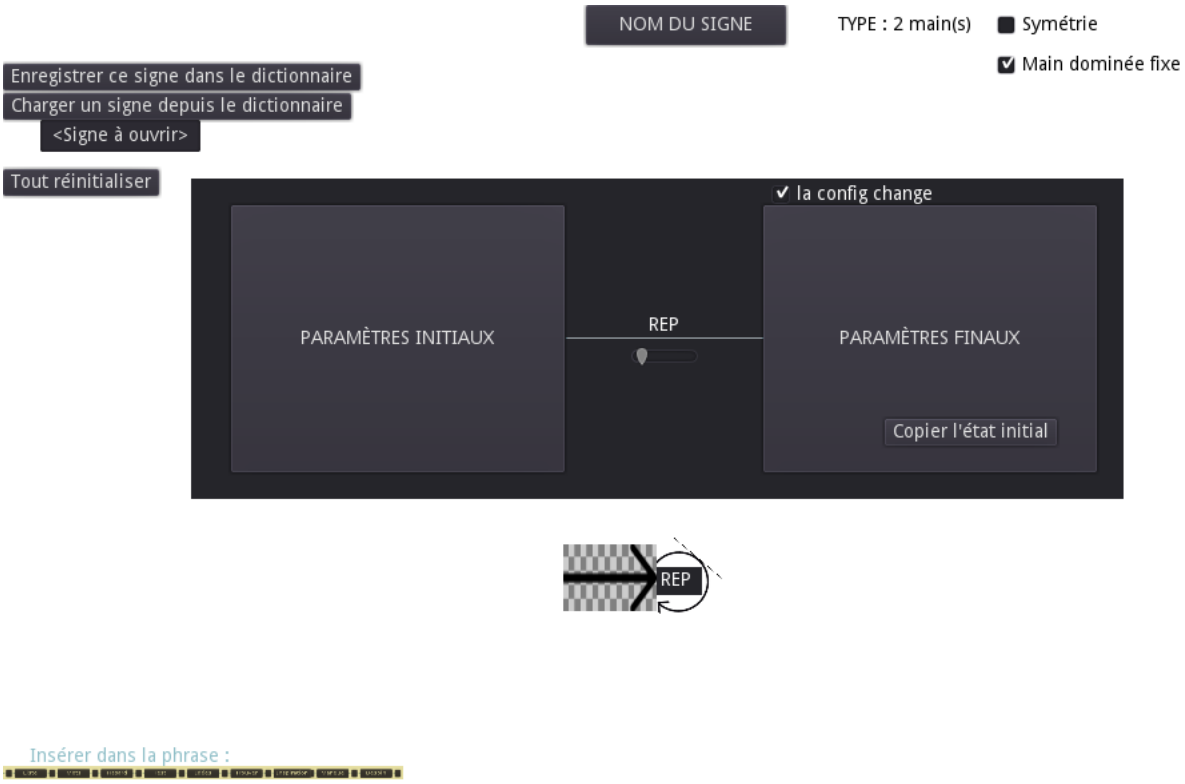

Figure 12: The main menu of the sign-editing interface

Then came the parameter-editing menu (figure 13): it is divided in two parts (dominating and dominated hand, one of them disappearing when the sign is defined as symmetrical or one-handed).

There is a pull-down menu to enter the hand shape (the list is initially very long but some clever and easy-to-remember (for sign language users) shortcuts can be typed in order to reduce the list). The other possibility to define the hand shape is the 3D hand (this was added when we obtained the 3D-hand from the animation team, proving that their work can easily be inserted into our interface), this hand can be rotated in three directions with the keyboard and we created a table of "letters", i.e. what you type on the keyboard to change the hand shape, we were actually able to get all hand shapes with typing only one distinct letter or an accent and a letter (keeping one key pressed while pressing another) for each of them. The only drawback of this method is that the user has to remember all combinations to use it efficiently and even though it is quite logical (and half of them are actually immediate for sign language users), it takes some getting used to. For this reason, we would like to keep the pull-down menu in some way even when we'll make the 3D hand the main element of this.

You can also see on the botton the old way to define the orientation of the hand. It was really not practical (even though it worked fine) and is now made possible by simply rotating the 3D hand.

Finally, there are functions to locate the hands. We needed a reference point on the hand, a placement point and a distance information. The two first points are defined by clicking either on the "pt de réf" button or on the global place where you want your hand and both take you to a more detailed picture of the body part where you can click on the precise location you want (see figure 14). The distance is a simple cursor that let's the user choose freely a position and then computes the closest point in the scale we chose to use.

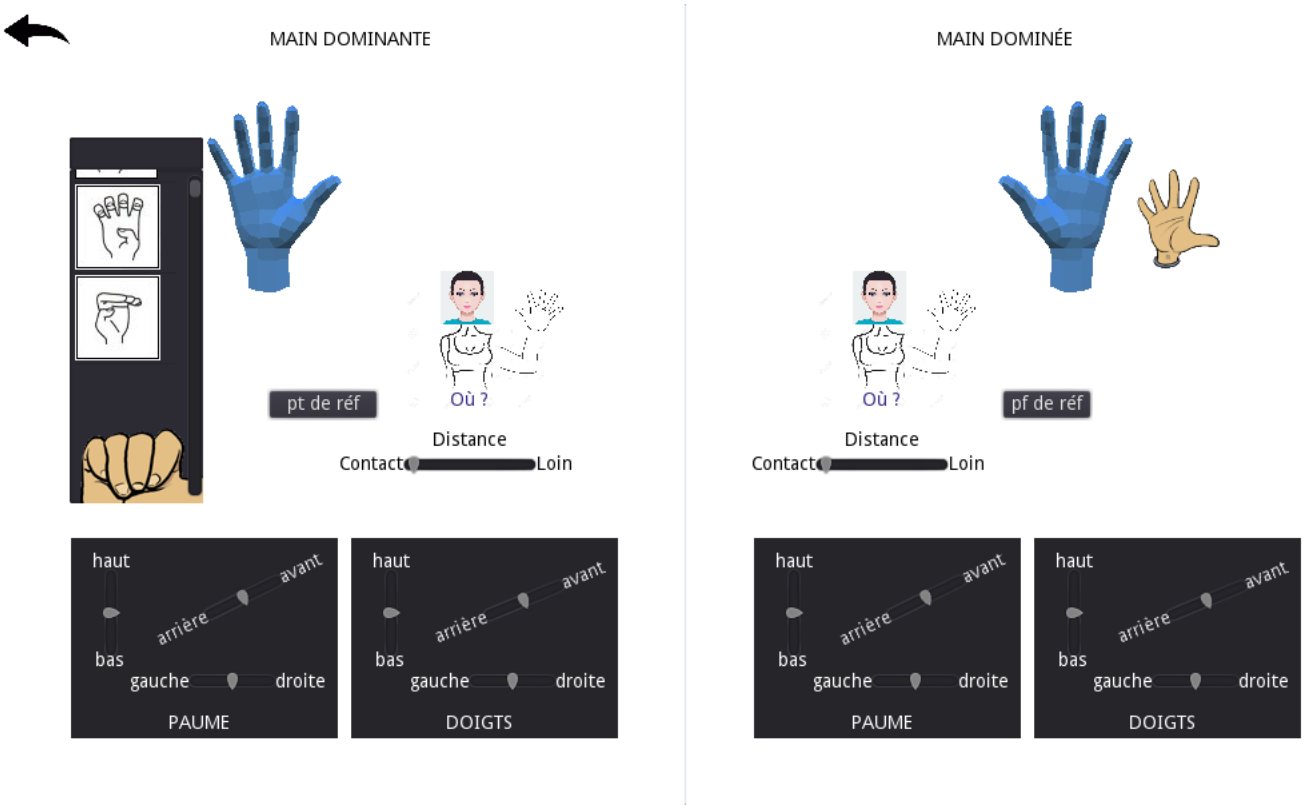

Figure 13: The parameter-editing menu of the interface

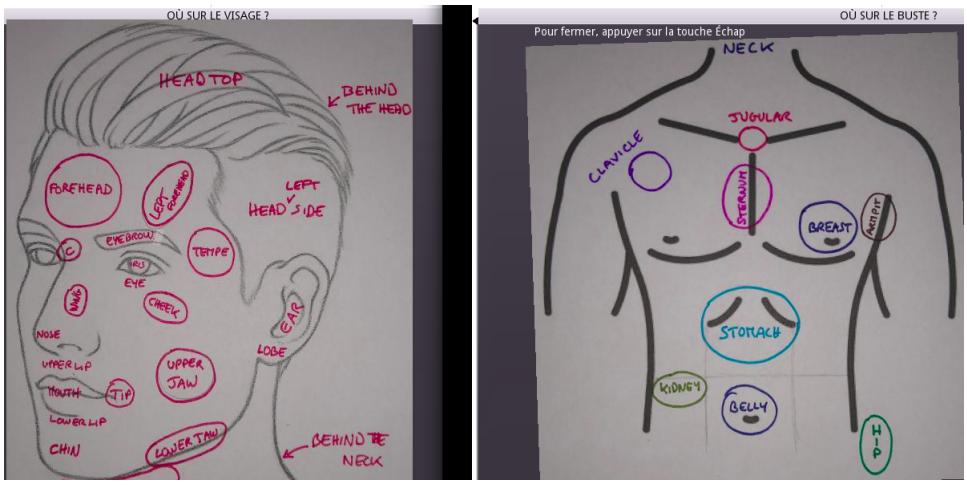

Figure 14: Window dialogs to choose a point on the face or the chest.

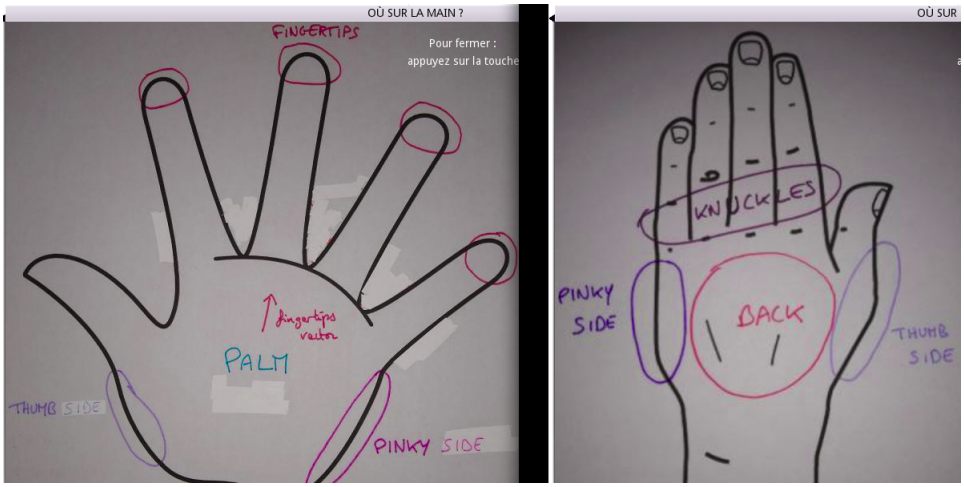

Figure 15: Window dialogs to choose a point on the hand (switch between them with Tab).

This interface uses `godot-yaml` to allow saving each sign's information into a `.yaml` file that follows the format we defined and in reverse, be able to load and display any sign written in our format in a `.yaml` file. We used this to start creating the "dictionnary": we entered fifteen signs by hand and saved them in the dictionnary. This was actually quite easy and quick to do and once everything is finished, we will save as many signs as possible in the dictionnary in order to make the software easier to use.

We then moved on to the sentence editor (figure 16): You can click on any word to be taken to the sign-editing interface, the name of the sign in the sentence will be updated. You can click on any grey rectangle between two words and it will add a new word there, that you can then edit like any other word. This part of the interface also has open and save functions.

Figure 16: Sentence Editor

With all the functions implemented, it is mostly a matter of ergonomy to make them individually more user-friendy. Aside from that, we decided that we wanted to put everything in one page, we drew the new interface and are now implementing it. The new face of the interface will have a huge panel in the top left corner for the sentence, a smaller panel in the top right corner for the avatar and a large banner at the bottom for the sign editing (with a reorganizing of the functions that lets the user zoom on the one their are working on). This way, it will be easier to know what is going on in the interface.

More details can be found on the git repository of the software team:

- a description of the format created in collaboration with all team members (`parameters.pdf`);
- an example of a list for the *hand shape* parameter, whose pictures will have to be replaced as soon as we produce our own;
- some pictures of our hand written work in /WIP_pictures;
- the code of the interface in Godot, that can be run once opened in Godot.

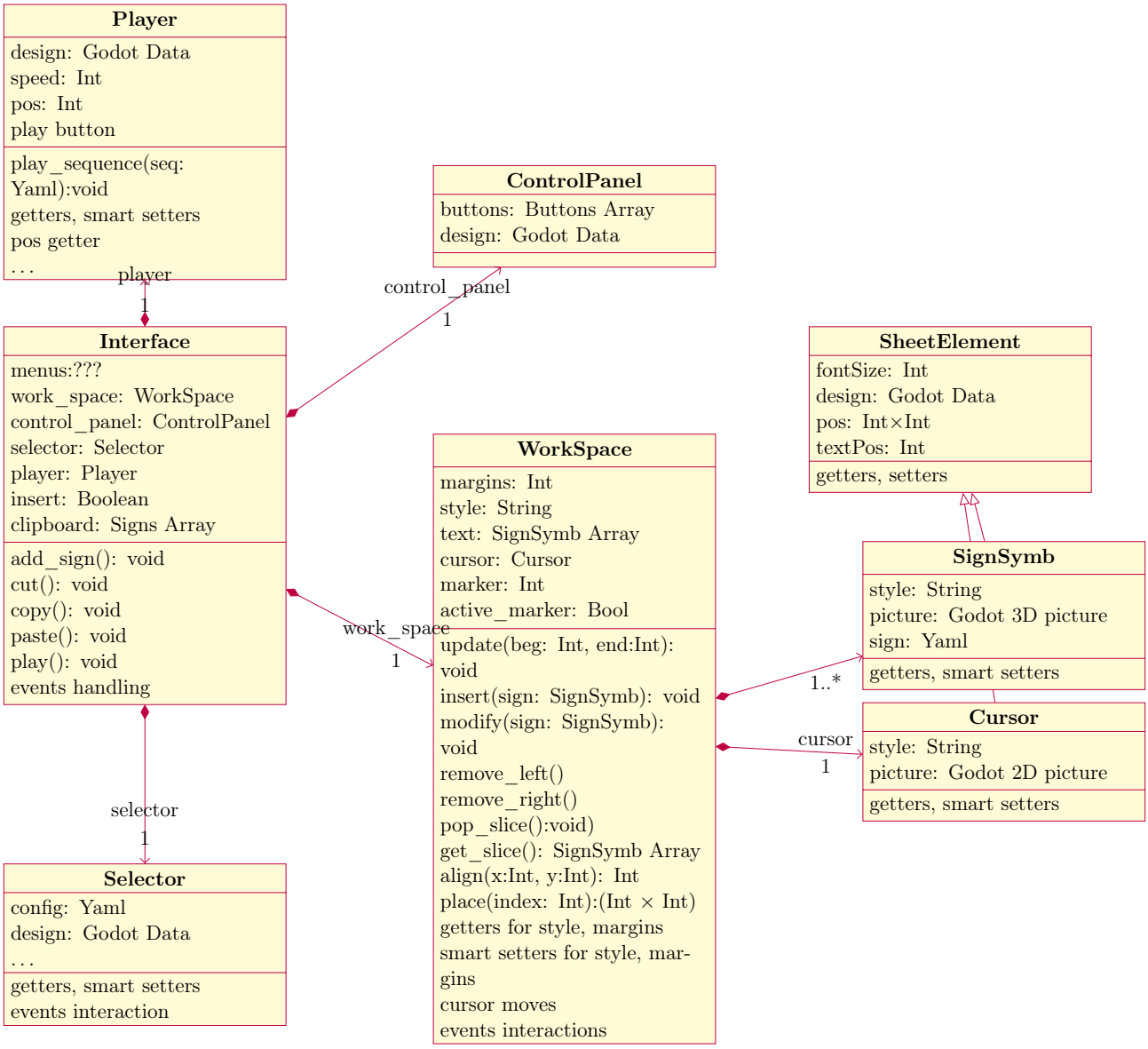

Figure 17: UML diagram for the new interface

Conclusion

Seven month after the beginning of Avatar ENSignes, we are now with a promising work-in-progress software about french sign language processing. Through a digital interface we are now able to input signs with keyboard and mouse. We also have an avatar that can play signs, and we developped algorithms to recognize signs from pictures. To sum up, we are in the middle of the third line of the PERT diagram (figure 18). The reorganization of the workpackages was quite efficient, since the animation team managed to get a first avatar.

There is however still a lot to do:

- add a more pedagogical presentation of the format. We are mainly lacking in pictures
- work on the ergonomic version of the interface (see UML diagram 17)
- motion tracking algorithms

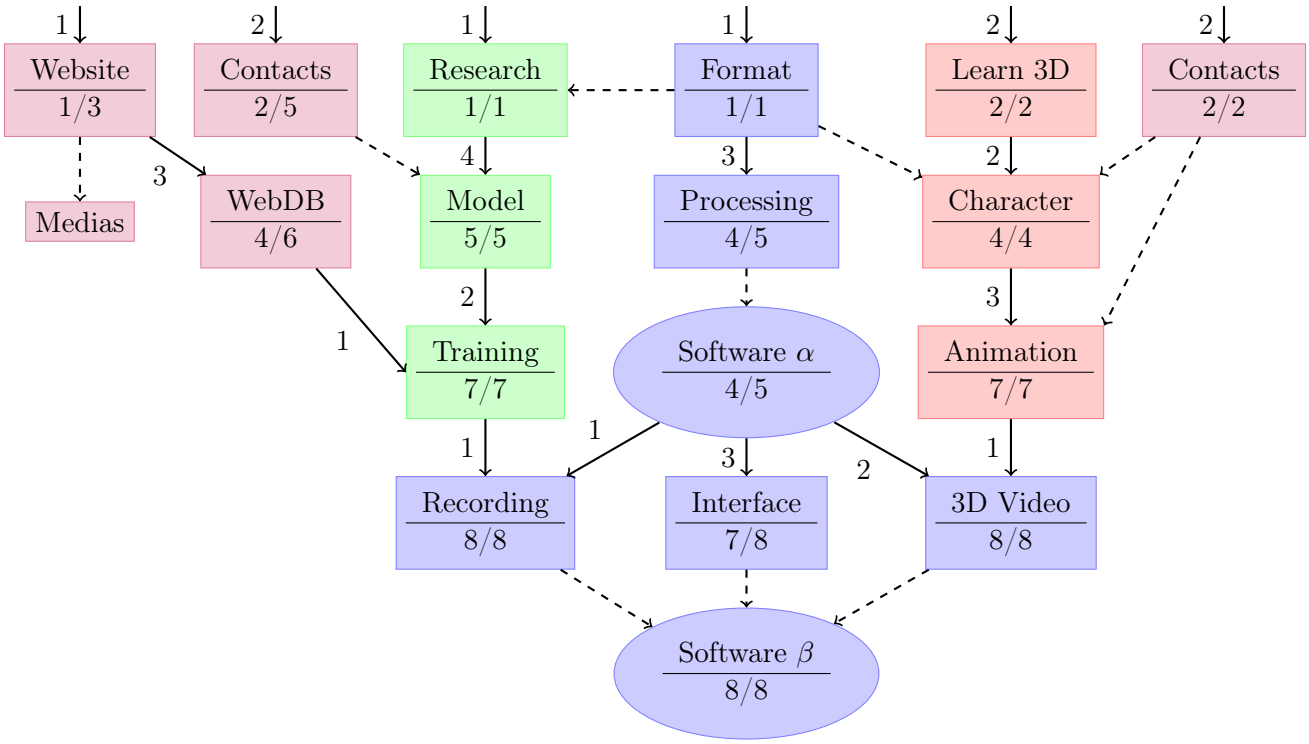

Figure 18: initial PERT diagram

- continue the avatar to make it fully animated

We also need to find people motivated to rejoin the project and thus work hard on communication.

References

[1] Zaher Hamid Al-Tairi, Rahmita Wirza OK Rahmat, M Iqbal Saripan, and Puteri Suhaiza Sulaiman. Skin segmentation using yuv and rgb color spaces. *JIPS*, 10(2):283–299, 2014.

[2] Reiner Creutzburg, Carla Leticia Ricalde Correa, Mauricio Moreno Sánchez Briseño, Juan Carlos Flores Oyervides, and César Chapela Carranco. Multimedia y compresión de datos. page 29, 2016.

[3] Jens Forster, Christoph Schmidt, Thomas Hoyoux, Oscar Koller, Uwe Zelle, Justus H Piater, and Hermann Ney. Rwth-phoenix-weather: A large vocabulary sign language recognition and translation corpus. In *LREC*, pages 3785–3789, 2012.

[4] Michal Kawulok, Jolanta Kawulok, Jakub Nalepa, and Bogdan Smolka. Self-adaptive algorithm for segmenting skin regions. *EURASIP Journal on Advances in Signal Processing*, 2014(170):1–22, 2014.

[5] Jakub Nalepa and Michal Kawulok. Fast and accurate hand shape classification. In Stanislaw Kozielski, Dariusz Mrozek, Pawel Kasprowski, Bozena Malysiak-Mrozek, and Daniel Kostrzewa, editors, *Beyond Databases, Architectures, and Structures*, volume 424 of *Communications in Computer and Information Science*, pages 364–373. Springer, 2014.

[6] Tomasz Grzejszczak, Michal Kawulok, and Adam Galuszka. Hand landmarks detection and localization in color images. *Multimedia Tools and Applications*, 75(23):16363–16387, 2016.

[7] Shekhar Raheja Ankit Chaudhary, Jagdish L. Raheja. A vision based geometrical method to find fingers positions in real time hand gesture recognition. *Journal of Software*, 7, 2012.

[8] Ching-Liang Su. Finger extraction, finger image automatic registration, and finger identification by image phase matching. *Applied Mathematics and Computation*, 188(1):912 – 923, 2007.

[9] Sylvie Gibet and Alexis Héloir. Formalisme de description des gestes de la langue des signes française pour la génération du mouvement de signeurs virtuels. *Rev. Trait. Autom. Lang.*, 48(3):115–149, December 2007.

[10] Thomas Hanke. HamNoSys-representing sign language data in language resources and language processing contexts. In *LREC*, volume 4, pages 1–6, 2004.

[11] UEA. JASigning. UEA, 2019.