Proceedings of the 9th Workshop on the Representation and Processing of Sign Languages, pages 209–216
Language Resources and Evaluation Conference (LREC 2020), Marseille, 11–16 May 2020

Video-to-HamNoSys Automated Annotation System

Victor Skobov, Yves Lepage
Graduate School of Information, Production and Systems
Waseda University
v.skobov@fuji.waseda.jp, yves.lepage@waseda.jp

Abstract

The Hamburg Notation System (HamNoSys) was developed for movement annotation of any sign language (SL) and can be used to produce signing animations for a virtual avatar with the JASigning platform. This provides the potential to use HamNoSys, i.e., strings of characters, as a representation of an SL corpus instead of video material. Processing strings of characters instead of images can significantly contribute to sign language research. However, the complexity of HamNoSys makes it difficult to annotate without a lot of time and effort. Therefore annotation has to be automatized. This work proposes a conceptually new approach to this problem. It includes a new tree representation of the HamNoSys grammar that serves as a basis for the generation of grammatical training data and classification of complex movements using machine learning. Our automatic annotation system relies on HamNoSys grammar structure and can potentially be used on already existing SL corpora. It is retrainable for specific settings such as camera angles, speed, and gestures. Our approach is conceptually different from other SL recognition solutions and offers a developed methodology for future research.

Keywords: sign language, machine learning, HamNoSys, corpus annotation

1. Introduction

The Hamburg Notation System 2.0 (HamNoSys) was presented by Prillwitz et al. (1989). An updated version 4.0 is described in (Hanke, 2004). It is meant to annotate movements, using a set of approximately 200 symbols and a standardized structure. This notation system is capable of transcribing any sign in any sign language (SL), which makes it a significant contribution to the current state of sign language research, not only because of its utility in corpus annotation but also because of its ease of processing. HamNoSys has a Unicode-based implementation[1] and has a fully developed markup language implementation Sign Gestural Markup Language (SiGML).

In this paper, by HamNoSys, we will be referring to its manual markup language representation - SiGML, as described by Elliott et al. (2004). In particular, SiGMLs manual features descriptions, the part that implements HamNoSys symbols as inner elements. SiGML can also include non-manual features of sign: pose, eye gaze, mouth, and facial expressions. The production of virtual avatar animations from SiGML was first presented by Kennaway (2002) and is used in the JASigning[2] platform.

The vast majority of sign language data is presented in a video-based corpus format and includes signs labeled along the timeline. Some data already include HamNoSys annotations (Hanke, 2006). However, not all available SL corpora are annotated with HamNoSys because such annotation is time-consuming. In particular, it requires a good understanding of the HamNoSys grammar. Having SL corpora annotated in HamNoSys would simplify and ease the cross-language research and SL data processing. The Dicta-Sign Project (Matthes et al., 2012) aimed at this by creating a parallel multilingual corpus. Their work provided a set of tools for direct HamNoSys annotation and allowed us to synchronize the generated animations with video data. Efthimiou et al. (2012) showed how to use image processing to find a matching sign from the vocabulary automatically, which improved annotation speed. However, video and image data require heavy processing. For example, Östling et al. (2018) presented comprehensive research on 31 sign languages and 120,000 sign videos. The location and movement information was collected using video processing. The collection alone took two months of computing time on a single GPU. Another problem with video-based corpora is an issue with signers' privacy. Most of the SL corpora that available today for researchers include a signing person. Which often leads to limited accessibility of the corpora.

Dreuw et al. (2010) presented a sign language translation system as a part of The SignSpeak Project, which includes an automatic sign language recognition system that is based on sign feature extraction from tracking the body parts on video frames. Curiel Diaz and Collet (2013) proposed another semi-automatic sign language recognition system, which also relies on head and hands tracking on video frames and uses a Propositional Dynamic Logic.

Still, many SL corpora do not include HamNoSys annotations. Hrúz et al. (2011) used an existing dictionary with annotated video data and custom classes for the categorization of signs in sign language corpora, making substantial progress towards annotating new data. However, such methods rely on an existing vocabulary, which is not always available. Ong and Ranganath (2005) discussed how a large-vocabulary recognition system still has to deal with sign translation and cannot fully grasp the actual meaning behind the signs and their different lexical forms. Additionally, the authors investigate how the annotation of signs allows us only to a certain degree to have an understanding of their meaning. Ebling et al. (2012) mentioned the problem of expressing the vocabulary of sign language using a spoken language. In order to have a raw sign language without interpretation and have full use of character-based natural language processing methods, an automated sign movement annotation system is required.

[1]HamNoSysUnicode: `http://vhg.cmp.uea.ac.uk/tech/hamnosys/HamNoSysFonts.pdf`

[2]JASigning: `http://vh.cmp.uea.ac.uk/index.php/JASigning/`

2. Methodology

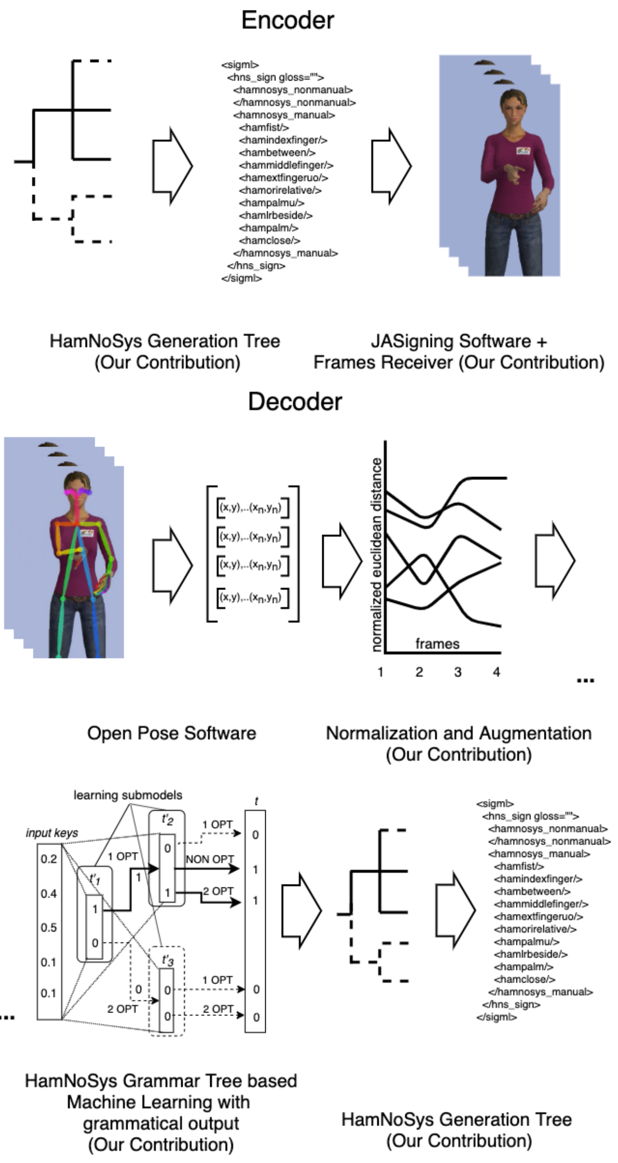
The main goal of our research is to develop a Video-to-HamNoSys decoder, having a HamNoSys-to-Video encoder. Thus, we propose our system in the form of the Encoder-Decoder model demonstrated in Figure 1.
For the training of the automatic annotation system, we require an extensive data set of sign videos, correctly transcribed in HamNoSys. The system should be able to transcribe any sign movement. Consequently, the training dataset has to be diverse and language-agnostic. To this end, we need a generation method of random HamNoSys annotations that incorporates all rules of the annotation system. Each generated sign annotation has to be accepted by the JASigning virtual avatar animation tool to produce signing animations. We use virtual avatar animations of the JASigning platform to synthesize movements from generated annotations. To cover all possible HamNoSys movements for any sign language, we need to generate signs randomly, representing random and chaotical movement. The encoding process will be handled mostly by JASigning software, and It will be detailed in Section 3..
The preprocessing steps, described in Section 4.1., will allow us to generate the necessary training data to train the decoder. The decoding process will be taken care of by our proposed tree-based machine learning model, detailed in Section 4..

3. Encoder

In this section, we first introduce a new generation method with a grammar tree that plays an essential role in the whole system. We describe the building process and the properties of the generation tree in the next Subsection, 3.1.Generation grammar tree. We then introduce the generation method with the generated data set that will be analyzed and compared to existing HamNoSys data. The overview of the encoder is shown in the upper part of Figure 1.

3.1. Generation Grammar Tree

To build the generation grammar tree, we start with the basic HamNoSys structure presented in the upper part[3] of Figure 2, and worked through each of the single rule blocks: Handshape, Hand position, Location, and Action. The notation for two hands was added afterwards. The verification was done by running the JASigning avatar animation (Kennaway, 2002). The process of adapting the rules was as follows: first, add the simplest ones; then add the specific rule cases. The presented work by Hanke (2004) was used as a description and guide of the HamNoSys. For the grammar source, we utilized the SiGML Data Type Definition (DTD) *hamnosysml11.dtd*[4] files with regular expressions. In particular, the *hamnosysml11.dtd* file provides a good overview and a basic understanding of HamNoSys grammar.
The upper part of Figure 2 displays the basic HamNoSys structure for one hand annotation. The lower part of Figure


Encoder
HamNoSys Generation Tree (Our Contribution)
JASigning Software + Frames Receiver (Our Contribution)
Decoder
[(x,y),..(xn,yn)]
normalized euclidean distance
frames
1
2
3
4
...
Open Pose Software
Normalization and Augmentation (Our Contribution)
learning submodels
input keys
0.2
0.4
0.5
0.1
0.1
t'1
t'2
t'3
t
1 OPT
NON OPT
2 OPT
HamNoSys Grammar Tree based Machine Learning with grammatical output (Our Contribution)
HamNoSys Generation Tree (Our Contribution)

```
<sigml>
  <hns_sign gloss="">
    <hamnosys_nonmanual>
    </hamnosys_nonmanual>
    <hamnosys_manual>
      <hamfist/>
      <hamindexfinger/>
      <hambetween/>
      <hammiddlefinger/>
      <hamextfingeruo/>
      <hamorirelative/>
      <hampalmu/>
      <hamlrbeside/>
      <hampalm/>
      <hamclose/>
    </hamnosys_manual>
  </hns_sign>
</sigml>
```

```
<sigml>
  <hns_sign gloss="">
    <hamnosys_nonmanual>
    </hamnosys_nonmanual>
    <hamnosys_manual>
      <hamfist/>
      <hamindexfinger/>
      <hambetween/>
      <hammiddlefinger/>
      <hamextfingeruo/>
      <hamorirelative/>
      <hampalmu/>
      <hamlrbeside/>
      <hampalm/>
      <hamclose/>
    </hamnosys_manual>
  </hns_sign>
</sigml>
```

Figure 1: Representation of a tree-based learning system: the upper part shows the encoder, the middle and the lower parts show the decoder of the system

2 is a representation of the same rule in the form of a tree. The root node is a non-terminal grammar symbol. Leaf nodes are representing the rule members and their order. Between the root and the leaves, there are production nodes, which are named "#_OPT" for optional ("#" stands for the number of the option) or "NON_OPT" for non-optional, i.e., obligatory nodes. The optional and non-optional nodes are necessary for the generation algorithm, which will be described in Subsection 3.2.. The lower part of Figure 2 shows the tree leaves which bear *empty*, representing an empty string or symbol in the grammar.

3.1.1. Regular Expressions

Figure 3 shows the examples of regular expressions implemented in a tree form. To describe a boolean *or* in the grammar, the "NON_OPT" node was used with "#_OPT" children for each *or* case. For *zero or one*, we used two "#_OPT" nodes, one of them having an "empty" node as a

[3] Available at https://robertsmithresearch.files.wordpress.com/2012/10/hamnosys-user-guide-rs-draft-v3-0.pdf

[4] Available at http://www.visicast.cmp.uea.ac.uk/sigml/

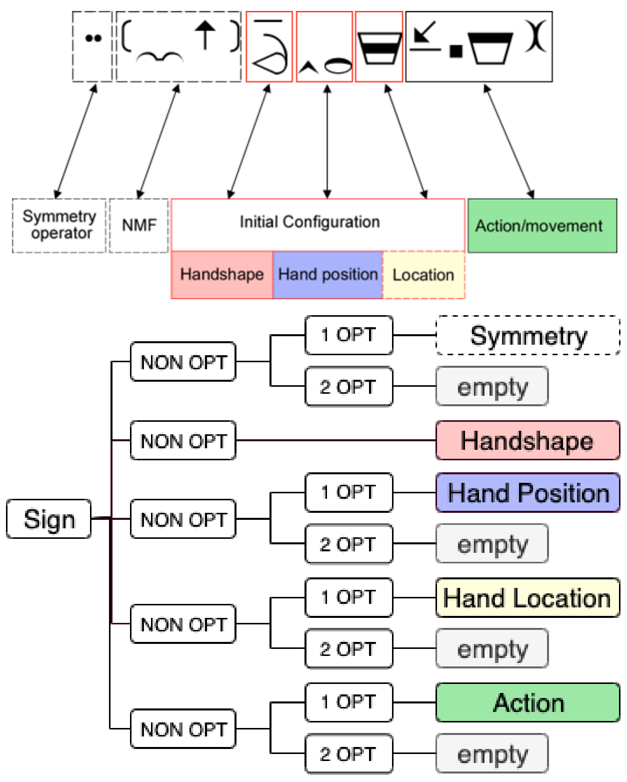


Figure 2: Implementation of the HamNoSys general structure (Up) in tree form (Down).

child. For *zero or more* and *one or more* we used optional "#_OPT" nodes. Any loop or recursive listing is limited only by **two** mentions. For example, the notation of different actions one after another could be infinite, but in order to introduce such rule of repeated notations of actions, and avoid infinite loops, the amount for actions was limited to a maximum of **two** actions.

3.1.2. Symbol Order

The HamNoSys system is order specific. Tree representation of a rule has to respect the order of symbols in the grammar. Keeping the order of leaves in the tree, according to the HamNoSys grammar, is crucial. In the next Subsection 3.2., the generation algorithm returns leaves in a post-order traversal of the leaves, allowing the algorithm to keep the grammatical order of the returned terminals.

3.1.3. Individual HamNoSys Sign Parts

The tree building process continues from root to the terminals. It is possible to build a generation tree for a single HamNoSys non-terminal element for elements like handshape, location, movement, etc. This might be useful if a change in a part of sign notation is needed. The generation of single sign parts was used for validation and modification of a single rule, during the conversion process.

3.1.4. Tree Limitations

During the process of building a sign generation tree, we had to remove a number of HamNoSys symbols, listed in Table 1. Most of them were excluded because they represent sign sentence and text markers: punctuation and location pointers. One symbol *hamupperarm* was removed

hamexclaim	*hamcomma*	*hamspace*
hamfullstop	*hamquery*	*return*
hammetaalt	*hamaltend*	*hamnbs*
hamcorefref	*hamupperarm*	*hametc*
hamaltbegin	*hammime*	*tab*
hamcoreftag	*pagebreak*	*linefeed*
hamnomotion	*hamversion40*	

Table 1: 20 HamNoSys symbols removed from the generation tree

from the hand *location1* rule, because the parser did not accept it. It is also worth mentioning that the *replacement* rule produces a visualization error during the animation, but it was kept regardless. By expanding the HamNoSys grammar, the generation tree can be recreated and updated.

3.1.5. Tree Format

For each of the 83 extracted rules, a tree representation was created with a non-terminal element as its root. These representations were combined into one large complete generation tree, with HamNoSys terminal symbols or *empty* symbols as leaves. The generation tree contains 302,380 leaves and stored in *Newick* format, (.nk or .nwk file). This format can store internal node names, branch distances, and additional information about each node encoded as features. For tree processing, we used the ETE Toolkit[5] package for *Python 3.6*. The resulting tree can produce any valid sign in any sign language, with only the limitations mentioned above. The 83 extracted rules and the generation tree will be released publicly with this paper, and open for future improvement.

3.2. Sign Generation Process

Having a grammar tree structure with all terminal elements as leaves and all non-terminals and internal nodes allows us to retrieve HamNoSys symbols as leaves. This can be achieved by traversing the tree and returning only the required symbols. We first mark the needed symbols. Marking is provided according to the grammar, i.e., the tree structure itself. We developed Algorithm 1 for this process. By using a top-down traversal, from the root to the leaves, the algorithm uses the optional "#_OPT" nodes to select a path to the leaves, which allows us to retrieve and mark the required terminal symbols.
The "#_OPT" optional nodes are used to introduce randomness into the generation process. In Algorithm 1, the goal is to generate a random sign, which represents a random movement. We create a data set of signs distributed over the whole HamNoSys grammar. We can modify the generation process by assigning weights to optional nodes. We can encode multiple signs on the generation tree with probability values between 0 and 1.
The HamNoSys notation allows a precise description of movements: specific annotation for each finger, complex location and position annotation of hand, repetitions, and symmetry of hand movements. The more precise the description, the more complicated the rule to be applied, and

[5]Available at http://etetoolkit.org/

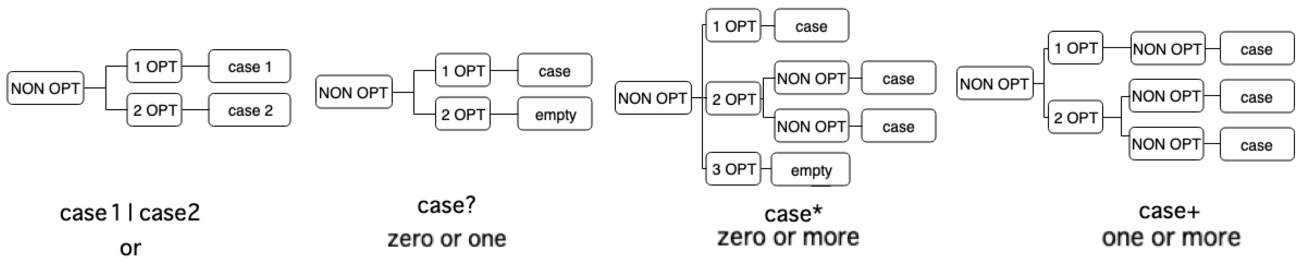


Figure 3: Implementation of regular expressions in tree form

Algorithm 1 Recursive tree walker for weighted generation

```
 1: function SIGNGENERATION($treenode$)
 2:     if tree node is leaf then                              ▷ Check whether the node is a leaf
 3:         return tree node                                   ▷ Terminal retrieved
 4:     else
 5:         Initialize $\mathcal{C} \leftarrow \emptyset$
 6:         for all child $\in$ tree node.children do
 7:             if child.name $=$'OPT' and $\neq$ 'NON OPT' then
 8:                 $\mathcal{C}$,$p_c \leftarrow$ child $\cup\, \mathcal{C}$         ▷ Getting set of optional children nodes and their probabilities
 9:         if $\mathcal{C} \neq \emptyset$ then                   ▷ Pick one child from the set according to their probabilities
10:             Initialize randomChild $\leftarrow WeightedRandomFromSet(\mathcal{C},p_c)$
11:             for all child $\in$ tree node.children do
12:                 if child.name $=$'NON OPT' or child $=$ randomChild then
13:                     $SignGeneration$(child)                ▷ All non optional and one picked optional child
14:         else
15:             for all child $\in$ tree node.children do
16:                 $SignGeneration$(child)                    ▷ If option Set is empty, go further to each child
```

the more complicated the rule, the longer the tree branch. Due to the differences in the topology distances between the root node and leaves, a random selection of optional nodes will produce an unequally distributed set of terminal symbols. As a result, the leaves with longer branches will occur less frequently in the generated sign. The use of such a data set may lead to over- or under-classification problems. Our goal is to provide a source of HamNoSys data, which will be used to train a machine learning system. In the ideal case, all leaves must have equal probabilities of being included in the generated sign, in a balanced data set. Consequently, equalization through the weighted selection process of optional nodes is required.

To accomplish this, in Algorithm 1: before selecting a random child on line 10, we compute the weights of all "#_OPT" optional nodes, according to their probabilities given by Formula 1. For a "#_OPT" child node, we take into account the sum of all leaves under all optional sister nodes and calculate the rate of leaves under each optional node. The probability value stored as a branch distance to the parent node due to its accessibility in the Newick format. With all rates computed, we perform a weighted random selection (WRS), which is provided as function *random.choices(weights)* in the package *random*[6] of *Python3.6* programming language to perform this.

$$p_{opt} = \frac{opt.leavesnumber}{\sum\limits_{opt \in \mathcal{C}} opt.leavesnumber} \tag{1}$$

We discuss the result of leaf probability equalization and its effect on the generated data set in the next section.

3.3. Generated Data

Using the proposed tree structure, we were able to generate data set with 10,000 signs with the weighted random selection (WRS) approach. The generation of 1 sign takes approximately 3.6 sec[7]. Figure 4 shows the distribution of generation tree leaf appearances in the generated sets. Due to the large size of the generation tree, it is hard to show a leaf with zero occurrence in the figure. Table 2 "Percentage of unused leaves" column gives the percentage of leaves with zero occurrences in the set. The desired data set should include all leaves to represent all grammar features of HamNoSys.

3.3.1. Weighted Random Selection (WRS) Data Set

Table 2 indicates that 13 % of the leaves did not appear in any sign of the WRS data set. The average single sign length is 357.98 symbols, and the longest sign consists of 508 symbols, which demonstrates the complexity of HamNoSys and its capability for describing complex movements.

[6]Description at: `https://docs.python.org/3.6/library/random.html#random.choices`

[7]on a single machine with Intel i7 processor 16 GB of RAM using CPU only

		Single Sign Length Stats			Percentage
Source	Unique Signs	Minimal	Average	Maximal	of unused leaves
Open DGS-Corpus	5,475	1	13.02	75	-
Generation with WRS	10,000	113	357.98	508	**12.73%**

Table 2: Comparison of signing data sets and their features

To compare the generated signs with existing natural language signs, we use the Open German Sign Language Corpus (DGS-Corpus)[8] presented by Prillwitz et al. (2008). Figure 4 shows that the average sign length for the DGS set is 13.02, which is significantly lower than in WRS generated set. It means that the DGS rule complexity is ”covered” by the generated data.
The generation program written in *Python3.6* for generating the data will be released publicly with this paper. The WRS generation can be used as an example of encoding the tree weights and generation of signs with specific features.

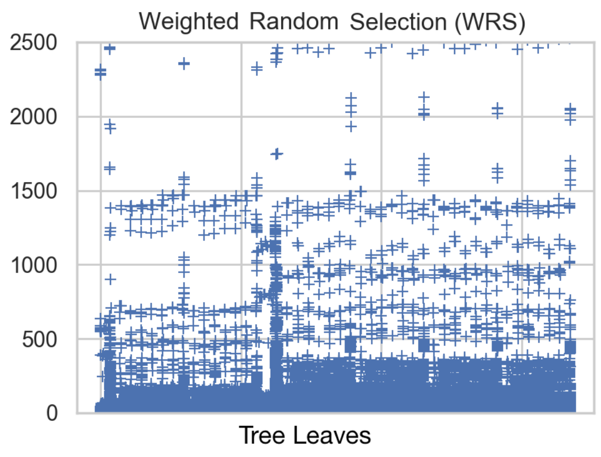


Figure 4: Impact on the distribution of 302,380 tree leaves occurrences in the generated sign set. 10,000 signs using a weighted random selection (WRS) algorithm between optional nodes.

3.4. Animation and Video Frames

The future system must be able to transcribe video materials, so it will receive video frames as an input and HamNoSys annotation as a SiGML file (.sigml) as an output. For that reason, the video of the generated SiGML signs has to be created. With the help of the JASigning tool, it is possible to produce animations virtual avatar from SiGML notation and send video frames of the animation on the network port. An example of the animation frame is demonstrated in Figure 5 (Left).

4. Decoder

In this section, we propose the Video-to-HamNoSys decoder as a part of our automatic video annotation system. The overview of the decoder is shown in the lower part of Figure 1. There are differences in camera settings, angles, and positions relative to the signer across different sign language video corpora. In Section 3.4., we showed how to produce animation frames with the JASigning software and store them as images. During this process, we define the camera angle and position in the virtual space. This step can be adjusted to match the camera setting of existing video corpora.

4.1. Data Preparation

Video frames are needed as training data for our machine learning model. To output a movement transcription as a string of characters from a given video, we do not need to input all the content of video frames, but only that information, which is required for the transcription. For that, we preprocess the video frames and extract the necessary significant movement features. The data preparation process consists of the following steps: animation, keypoints extraction, normalization, augmentation. Each of these steps is detailed below.

4.1.1. Body Keypoints Extraction

After getting the animation frames for each generated sign, we use the OpenPose Software (Cao et al., 2018) to extract the body keypoints. OpenPose can detect 18 points on the body, 25 points on each hand, and 60 points on the face (Simon et al., 2017). Instead of using convolutional layers, we use a pre-trained OpenPose model to extract significant features from the frames. This approach reduces the size of training data and ensures that our decoder model receives only the necessary information about the body movement. The right part of Figure 5 shows the keypoints detected by OpenPose in the frame shown on the left part of Figure 5.

4.1.2. Normalization and Augmentation

OpenPose extracts the coordinates of the body keypoints from frames. The real camera settings in existing SL corpora can be very different from the virtual camera settings in the JASigning software. For instance, different resolution and aspect ratio may be used. Therefore, we added a normalization step to the decoder that should detach the body keypoints from the frame coordinates, and represent them as an array of distances to each other. The unnecessary keypoints, like legs and hips, are removed because they are not involved in the signing process. We calculate the euclidian distances between the remaining keypoints and normalize them against the *base body distance*. The distance that is not significantly changing during the signing process should be set as a *base body distance*. In our case, it is the distance between two shoulder points. Its value is set to 1, and the rest of the calculated distances are put into relation to it. As a result of the normalization step for each frame, all of the sign data is represented by one matrix, which stands

[8]HamNoSys was extracted from i-Lex files Available at `https://www.sign-lang.uni-hamburg.de/dgs-korpus/index.php/welcome.html`

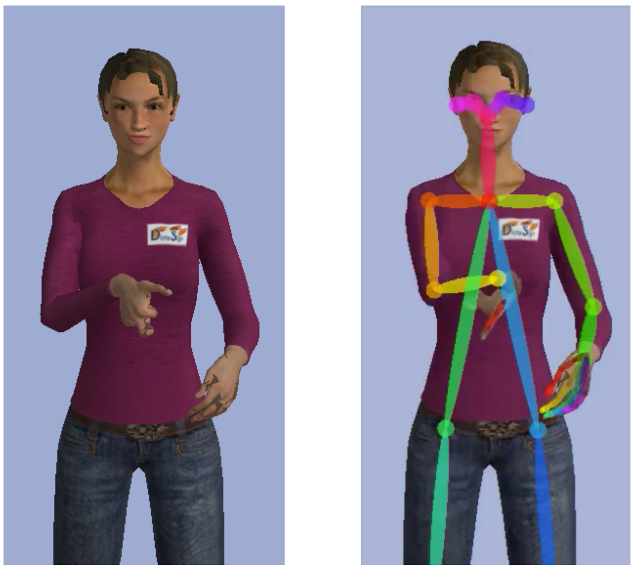


Figure 5: Example of body keypoints (on the right) extracted from a single animation frame (on the left)

for the sign movement. On some of the frames, when the keypoint detection fails, we keep the default value without any change.

As an example, in Figure 6, the movement of one hand getting closer towards the head is represented by decreasing curves, which stands for the distances between the hand and the head keypoints. As the synthetic avatar movements differ from real human movement, and so as to adapt our model to real environment, with the aim to perform well in annotating real human signs, we augment the avatar data with noise.

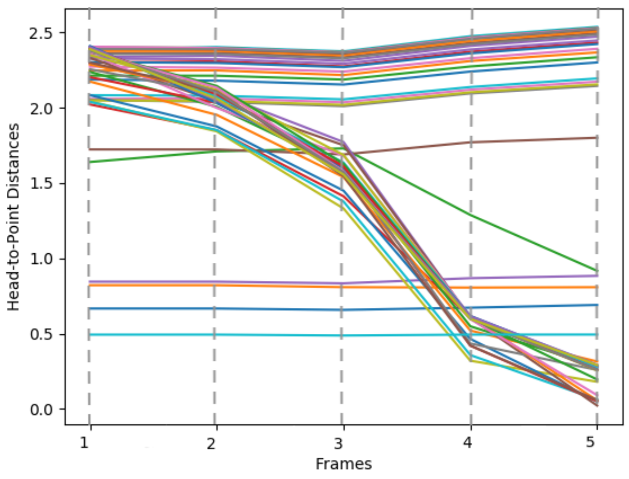


Figure 6: Plotted changes in body keypoint distances to the head keypoint across five frames

4.2. Tree-based Machine Learning System

The resulting decoder has to approximate the whole HamNoSys grammar, to generate grammatically correct output. This can theoretically be achieved by training a deep neural network, on a large number of data. We believe that such an approach is possible today, although it is not transparent. There is a risk that the output will be ungrammatical. This would also require high computational power and resources.

We suggest an alternative way. Since we have all grammar rules at our disposal in one united tree structure (that we used for the generation of the training data), we can exploit this structure to create a learning model based on the large grammar tree, where each output sign will be generated with regard to the extracted body keypoints of the initial animation movement presented in Section 4.2.1. and illustrated by Figure 7. Similar to the generation process, by traversing the generation tree, we will initialize a small learning submodel on the nodes that have optional "#_OPT" nodes as children. (Given in Section 3.2.) Algorithm 1 is used in the training process, but instead of selecting the optional "#_OPT" nodes randomly with WRS, the leaning submodel learns the right selection of optional "#_OPT" nodes. The whole generation path is learned by a chain of submodels based on the sign movement.

After the sign has been generated, it is represented by a one-dimensional vector of the generation tree's leaves. If the leaf is included, it is labeled as 1, 0 if not. This allows us to trace back each rule node in the generation tree, and see the sign recognition task as a rule-classification problem. By visiting the same leaves as during the initial sign generation, the Decoder has to repeat the initial generation process done by the Encoder. Nodes and their associated learning submodels that were not visited during the initial sign generation will not be visited during training, and their submodels will not be trained. In this way, it is guaranteed that the resulting system will always output a grammatically correct HamNoSys annotation.

Figure 7 shows the simplified underlying representation of this idea. Submodels t'_1 and t'_2 are initialized based on the main target vector t. All non-optional nodes are not learned and are set to 1 automatically. As shown in Figure 7, the node is not visited and its submodel t'_3 is not trained, since it is not involved in the generation process of the target sign at hand. This prevents oversampling on negative examples. However, the drawbacks are a reduction of training samples on the deeper levels of the generation tree and, consequently, a degradation of the accuracy, described in Section 5..

4.2.1. Learning Submodels

Each submodel has a dropout layer with a probability of 10 % (*p=0.1*), and a fully-connected layer with a LeakyRelu and Softmax activation functions. As a loss function, The Cross Entropy Loss was used, with the Adam optimizer (Kingma and Ba, 2015). The size of the output is different among all submodels, and it corresponds to the number of optional "#_OPT" children nodes, also viewed as subclasses of the tree-based machine learning system. During the training of a submodel, a subtarget vector is produced, based on the main target. It indicates whether the leaf of the corresponding optional node is included in the target sign. As shown in Figure 7, t'_1 elements are computed from t.

During tree traversal, we preprocess all the training data

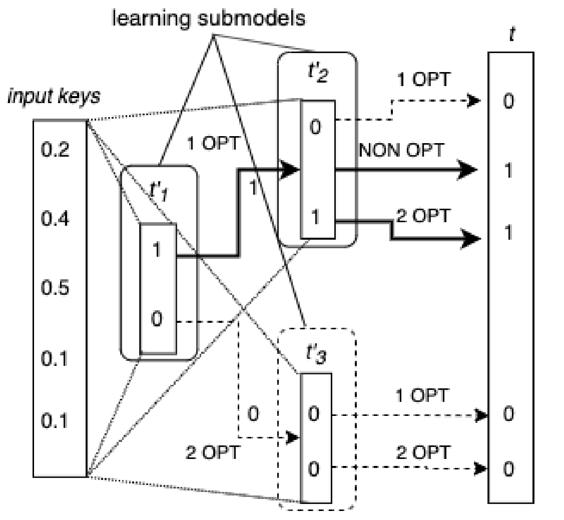


Figure 7: Representation of a tree-based learning system

and extract the subtargets to create training data subsets for each submodel. Table 3 show the number of samples that is possible to retrieve for submodels on different tree levels. We used this approach to analyze the accuracy of submodels on a different tree-levels; this is detailed further in Section 5..

Tree Level	SM	Avg. N per SC	Avg. SC	Avg. Valid. Accuracy
1	1	4561	2	93 %
2	2	1701	3	80 %
3	1	222	12	31 %
4	22	89	2	47 %
5	62	237	4	42 %
6	474	113	2	46 %
7	332	86	4	32 %
8	268	35	2	51 %
9	496	33	4	33 %
All Levels	1658	81	3	**40 %**

Table 3: SM - Number of Submodels; N - Number of training samples; SC - Number of Subclasses; Training efficiency on diferent levels of learning tree

5. Experimental Results

For our experiments, we used a part of the grammar tree, that stands for the description of handshapes. In the upper part of Figure 2, the handshape description is marked with a red color. We generated 60,000 handshape signs for one hand and extracted the body keypoints from them with the same camera position. That took us nearly ten days. The generation of the training data could potentially be simplified if JASigning Software would allow extracting the body keypoints directly from the animation.

Training of all 1,658 submodels with 800 epochs each took around five days. After implementing multiprocessing and utilizing 11 threads, we managed to reduce this time to 30 hours.

Training the tree-based machine learning system resulted in 22 % accuracy on a validation set of the 1,000 sign. In our words, our system correctly predicted 5,728 sized vector that represents a sign. The average accuracy among all submodels in the tree-based machine learning system is 40 % with an average of three classes, and an average of 81 samples per class. It is important to notice that the validation set was generated with a Weighted Random Selection (WRS) Algorithm 1 like the training data.

Our proposal being conceptually new and fundamentally different; it is not directly comparable to other approaches. Table 4 is a comparison attempt, where we give the number of subclasses in our model against the number of signs in other approaches. Our accuracy of 54 % seems to be significantly lower than the results of other models, but the reader should notice that the number of classes is much bigger.

Further investigation of our results led to an interesting finding: the degradation of the accuracy on the deeper levels of the learning tree. Table 3 gives a comparison of the average accuracy of the submodels on the different tree levels, and the amount of training samples with the average number of subclasses. Typically, for machine learning classification algorithms, the accuracy drops for a lower number of training samples and a higher number of classes. Consequently, our results might improve if we increase the number of training samples.

Image Classifier	Unique Classes	Accuracy
(Bheda and Radpour, 2017) Deep CNN	32	82 %
(Bantupalli and Xie, 2018) Deep CNN	100	93 %
Body Keypoints Classifier		
Our Model 1 Frame Average Submodel Accuracy	3	38 %
across all Subclasses	**5,728**	**9 %**
Our Model 5 Frames Average Submodel Accuracy	3	40 %
across all Subclasses	**5,728**	**22 %**

Table 4: Comparison to other arproaches

6. Discussion and Future Work

We presented an automatic annotation system that relies on HamNoSys grammar structure. Proposed approach can potentially be used for annotation of any hand movement.

For future work, we suggest modifying the training data generation process, by extracting the body keypoints from the JASigning virtual avatar directly, skipping saving the frames, and processing them with OpenPose Software. That will allow the generation of the comprehensive training data set, with the whole HamNoSys grammar included, and different camera angles. Potentially generation process could be done during the training ”on the fly.”

Modifications to data normalization and augmentation can be done, to achieve better performance on the real data. The model should learn the differences between the movements of an avatar and multiple human signers. Additionally, the automatic annotation system for facial expressions

and other non-manual features by further use of SiGML attributes will expand the system utilization.
By representing an entire corpus with HamNoSys strings of characters and JASigning generated animations, the size of the corpus could drastically decrease. In this way, processing and sharing corpus data might become more accessible. Processing of SL corpora as strings of characters allows the use of sophisticated NPL techniques for sign language analysis and research. This opens new directions in SL research.

7. Bibliographical References

Bantupalli, K. and Xie, Y. (2018). American sign language recognition using deep learning and computer vision. In Naoki Abe, et al., editors, *IEEE International Conference on Big Data, Big Data 2018, Seattle, WA, USA, December 10-13, 2018*, pages 4896–4899. IEEE.

Bheda, V. and Radpour, D. (2017). Using deep convolutional networks for gesture recognition in american sign language.

Cao, Z., Hidalgo, G., Simon, T., Wei, S., and Sheikh, Y. (2018). Openpose: Realtime multi-person 2d pose estimation using part affinity fields.

Curiel Diaz, A. T. and Collet, C. (2013). Sign language lexical recognition with Propositional Dynamic Logic. In *51st Annual Meeting of the Association for Computational Linguistics (ACL)*, pages pp. 328–333.

Dreuw, P., Ney, H., Martinez, G., Crasborn, O., Piater, J., Moya, J. M., and Wheatley, M. (2010). The signspeak project - bridging the gap between signers and speakers. In Nicoletta Calzolari (Conference Chair), et al., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA).

Ebling, S., Tissi, K., and Volk, M. (2012). Semi-automatic annotation of semantic relations in a swiss german sign language lexicon. In *5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon. Language Resources and Evaluation Conference (LREC 2012)*, pages 31–36.

Efthimiou, E., Fotinea, S.-E., Hanke, T., Glauert, J., Bowden, R., Braffort, A., Collet, C., Maragos, P., and Lefebvre-Albaret, F. (2012). Sign language technologies and resources of the dicta-sign project. In *5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon. Language Resources and Evaluation Conference (LREC 2012)*, pages 37–44.

Elliott, R., Glauert, J. R. W., Jennings, V., and Kennaway, J. R. (2004). An overview of the sigml notation and sigmlsigning software system. In *Sign Language Processing Satellite Workshop of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pages 98–104.

Hanke, T. (2004). Hamnosys-representing sign language data in language resources and language processing contexts. In *Sign Language Processing Satellite Workshop of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, pages 1–6.

Hanke, T. (2006). Towards a corpus-based approach to sign language dictionaries. In *Proceedings of a Workshop on the representation and processing of sign languages: lexicographic matters and didactic scenarios (LREC 2006)*, pages 70–73.

Hrúz, M., Krňoul, Z., Campr, P., and Müller, L. (2011). Towards automatic annotation of sign language dictionary corpora. In Ivan Habernal et al., editors, *Text, Speech and Dialogue*, pages 331–339, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kennaway, R. (2002). Synthetic animation of deaf signing gestures. In Ipke Wachsmuth et al., editors, *Gesture and Sign Language in Human-Computer Interaction*, pages 146–157, Berlin, Heidelberg. Springer Berlin Heidelberg.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Yoshua Bengio et al., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Matthes, S., Hanke, T., Regen, A., Storz, J., Worseck, S., Efthimiou, E., Dimou, A.-L., Braffort, A., Glauert, J., and Safar, E. (2012). Dicta-sign – building a multilingual sign language corpus. In *Proceedings of 5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon (LREC 2012)*, pages 117–122.

Ong, S. C. W. and Ranganath, S. (2005). Automatic sign language analysis: A survey and the future beyond lexical meaning. *IEEE Trans. Pattern Anal. Mach. Intell.*, pages 873–891.

Östling, R., Börstell, C., and Courtaux, S. (2018). Visual iconicity across sign languages: Large-scale automated video analysis of iconic articulators and locations. *Frontiers in psychology*.

Prillwitz, S., Leven, R., Zienert, H., Hanke, T., and Henning, J. (1989). *HamNoSys version 2.0. Hamburg notation system for sign languages—an introductory guide*. Signum-Verlag.

Prillwitz, S., Hanke, T., König, S., Konrad, R., Langer, G., and Schwarz, A. (2008). DGS corpus project-development of a corpus based electronic dictionary German Sign Language / German. In *3rd Workshop on the Representation and Processing of Sign Languages: Construction and Exploitation of Sign Language Corpora (LREC 2008)*, pages 159–164.

Simon, T., Joo, H., Matthews, I. A., and Sheikh, Y. (2017). Hand keypoint detection in single images using multiview bootstrapping. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 4645–4653. IEEE Computer Society.