Author(s): Karppa, Matti; Viitaniemi, Ville; Luzardo, Marcos; Laaksonen, Jorma; Jantunen, Tommi

Title: SLMotion – An extensible sign language oriented video analysis tool

Year: 2014

Version:

Please cite the original version:

Karppa, M., Viitaniemi, V., Luzardo, M., Laaksonen, J., & Jantunen, T. (2014). SLMotion – An extensible sign language oriented video analysis tool. In N. Calzolari, K. Choukri, T. Declerck, H. Loftsson, B. Maegaard, J. Mariani, A. Moreno, J. Odijk, & S. Piperidis (Eds.), Proceedings of the 9th international conference on Language Resources and Evaluation (LREC 2014) (pp. 1886-1891). European Language Resources Association (LREC). LREC proceedings. http://www.lrec-conf.org/proceedings/lrec2014/pdf/209_Paper.pdf

SLMotion – An extensible sign language oriented video analysis tool

Matti Karppa*, Ville Viitaniemi*, Marcos Luzardo*, Jorma Laaksonen*, Tommi Jantunen†

*Department of Information and Computer Science,
Aalto University School of Science, Espoo, Finland,
firstname.lastname@aalto.fi

†Sign Language Centre, Department of Languages,
University of Jyväskylä, Finland,
tommi.j.jantunen@jyu.fi

Abstract

We present a software toolkit called *SLMotion* which provides a framework for automatic and semiautomatic analysis, feature extraction and annotation of individual sign language videos, and which can easily be adapted to batch processing of entire sign language corpora. The program follows a modular design, and exposes a Numpy-compatible Python application programming interface that makes it easy and convenient to extend its functionality through scripting. The program includes support for exporting the annotations in ELAN format. The program is released as free software, and is available for GNU/Linux and MacOS platforms.

Keywords: automatic annotation, video analysis, sign language

1. Introduction

SLMotion is a general purpose video analysis software toolkit that we have developed for analysing sign language corpora. The software allows researchers to perform different kinds of analyses on pre-existing video footage via non-invasive computer vision methods. This could be used, e.g., to verify theoretical statements concerning sign language phonetics, or be used as an assisting tool for semiautomatic or automatic annotation of pre-existing video corpora.

Our software consists of two major parts: a generic control framework that enables users to conduct experiments and analysis projects using Python scripts, and a library of independent analysis components that can be used as parts of such projects. While our use of the software has been limited to the analysis of sign language corpora (Karppa et al., 2012; Karppa et al., 2011; Viitaniemi et al., 2013; Luzardo et al., 2013), the framework itself does not contain inherent limitations that would prevent its application to other domains of video analysis, such as research of gestures or non-verbal interaction between speakers of spoken languages.

The usefulness of the software comes from the framework which allows the users to share their work with colleagues. The modular design of the component framework allows creation of self-contained components that can be applied to other projects in a plug & play manner. Scripting with the Python Application Programming Interface (API) enables users to share the scripts they have made and allows recycling of old projects. Users without a background in programming may choose to use premade scripts that accomplish their desired tasks. Such scripts could, for example, define processes that annotate sequences of visual silence or lack of motion, measure velocities of the articulators, typically the hands, or create annotations corresponding to eye blinks.

The source code of the software, and Ubuntu Linux and MacOS X binaries are publicly available for download free of charge at `http://research.ics.aalto.fi/cbir/software/` along with more extensive technical documentation. The software is licensed under the very permissive FreeBSD licence, so it can be used for commercial work in addition to academic settings.

The rest of the paper is organised as follows: Section 2. discusses some related work. Section 3. describes the system and its capabilities. Section 4. lists some of the work where we have implemented our methods within the framework described here. Section 5. concludes the paper.

2. Related work

A lot of work on computer-vision based approaches towards sign language analysis has surfaced in the recent years. This also includes large corpora, the appropriate analysis of which without automatic tools would be an insurmountable task. For example, in (Forster et al., 2012), the authors describe a sign language recognition and translation corpus based on weather forecasts signed in German Sign Language. They also describe experiments with hand and head tracking on the videos in the corpus. In (Charles et al., 2013), an automatic system is described for estimating human pose in long sequences of sign language video.

However, to the authors' best knowledge, no free software alternatives exist to SLMotion. This is because SLMotion works at a higher level of abstraction; it provides a framework for implementation and experimentation. Most related work that has been released to the public in the form of software has focused on narrower, domain-specific problems, and in many cases the proposed methods could be considered to be implemented as *components* within our system. Very little work can be found that operates at the same level of abstraction.

The AVATecH project (Auer et al., 2010) has had similar aims at creating interchangeable components with a standardised interface for the purpose of automatic or semi-automatic annotation of audio/video corpora. Similarly,

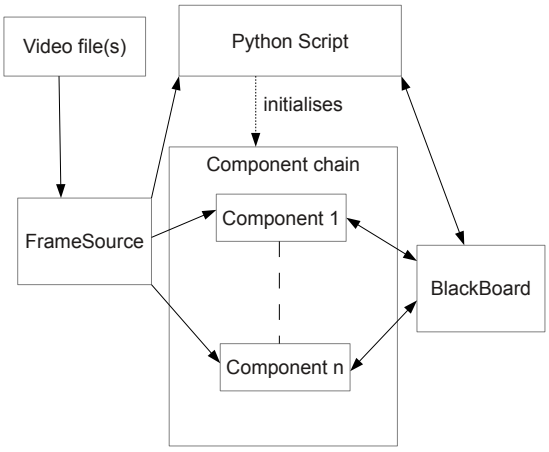

Figure 1: SLMotion execution model. The components only interact with one another through the black board. The Python script controls the execution at the top level, but it cannot directly influence the behaviour of the components once they have been initialised.

in (Lenkiewicz et al., 2012), the tracking of the hands and the head has been explored within the AVATecH project. However, to our knowledge, the described recognisers are not free in the free software sense, which may be considered a limiting factor. Also, based on an assessment of the description in (Lenkiewicz et al., 2012), our scripting system could give the more technically-oriented user a more fine-grained level of control without sacrificing the simplicity of use, should the associated components be designed properly.

3. System description

3.1. Framework architecture

Overall, the most important design principle behind SLMotion has been extensibility. This has yielded a modular structure that makes it possible to add new functionality in the form of self-contained components.

The execution model follows a black board model and is shown in Fig. 1. At the core is a component chain, i.e. a succession of self-contained processing tasks that are run for the input video one after another. Ideally, each component performs only one task at a time and is unaware of any other components in the chain. The only way for the components to interact with one another is through the black board which acts as an abstract data repository. The components will store their own results there and likewise load the results of previous components if they should depend on them. The components provide a range of metadata regarding their requirements and the results they provide for other components so that the system can often verify that all necessary data will be available in time and tell the user what component might provide the missing data.

On a higher level, the initialisation of the component chain is performed by a Python script that runs in an embedded Python interpreter. The program exposes an API that allows the user to initialise the components to their desired settings, set up the desired visualisations for video output, select which features to extract, manipulate the resulting data on the black board, or feed in data from an external source to the black board. The interface is also Numpy[1]-compatible which should make a wide range of numerical methods accessible. The state of the analysis can also be stored into a file as a "blackboard dump" and loaded back again. The dumps can be used for processing large videos in pieces, or performing multiple analyses on the same video without having to recompute results common to each task.

Listing 1 shows a simple example of a possible Python script that could be run with SLMotion. The second textual line shows that the SLMotion API is accessed simply as a Python module. This is followed by the initialisation of the component chain with the `setComponents` function. Components are created by passing a list of `Component` objects to the function. The `Component` objects act as component specifications. They include the name of the component and the parameters, passed as a dictionary with parameter names as keys and arguments as values. This behaviour is seen, for example, with the `StaticElmSkinDetector` component where the parameter `model` is set. The component chain in Listing 1 will first perform Viola-Jones face detection, skin detection using a pre-trained Extreme Learning Machine (ELM) model, connected blob extraction to find contiguous pixel areas in the skin mask, and finally locating the hand using the preceeding results. The actual processing is triggered by a call to the `process` function. Finally, the visualisation options are set so that the results computed by the script are used to draw the location of the hand as a rectangle superimposed on the input frame. The result can be seen in Fig. 2. The final lines demonstrate how the result data can be accessed from Python side. In this case, the result would be a simple list of four integers, describing the rectangle. The output, corresponding to Fig. 2 where a single frame was used as input, would have been `[267 194 63 122]`.

While the architecture gives the developer a lot of flexibility with design, it places certain limitations on the data that can be processed. Most importantly, since the video is scanned from the beginning to the end multiple times, and the different components are run independently of one another, all results must be preserved in the main memory over the entire duration of the process. This can be a challenge if some of the components produce large matrices for each frame in the video, as the memory requirements can grow very large. This may limit the length of videos that are processed with certain components. In order to alleviate this effect, SLMotion can transparently apply Zlib compression to matrices. This is particularly effective in the case of binary matrices, such as skin masks. It is not possible to give numerical estimates about maximum lengths of videos that can be processed since the limit is affected by the amount of RAM in the system, the resolution of the video, and most importantly the results that the components require. Components that only provide data in the order of kilobytes per frame can practically handle arbitrarily long videos. On the other

[1] `http://www.numpy.org/`

Listing 1: A Python script example.

```
#!/usr/bin/python

from slmotion import *
import cv2

# create the component chain
setComponents([Component('FaceDetector'),
               Component('StaticElmSkinDetector',
                         {'model': 'elm_skin_balanced.txt'}),
               Component('BlobExtractor', {'minblobsize': 1}),
               Component('HandLocator', {'padding': 5})])

process()

setVisualisation('showFrame; showRect handlocation')
outputVideo()

# this is a simple example of how to access the results from Python end
for i in range(len(framesource)):
    print blackboard.get('handlocation', i)
```

Figure 2: The hand location as detected with the script in Listing 1.

hand, with components that create dense and poorly compressible matrices of size comparable to the input frame size, the practical limit may, in the case of high definition video input, be on the order of few minutes. Fortunately, even in the worst case, it is usually possible to process any video in pieces.

Also, since components expect random access to video frames, running independent components in parallel within one process is currently not supported, but luckily this problem can be overcome in corpus processing by the fact that multiple separate processes can be run on parallel. Experiments conducted with parallelisation within a single process showed that the I/O overhead presented a major bottleneck, choking performance. This is made worse by the fact that in a lot of cases it is not possible to reliably fetch a given, arbitrary frame from the video stream. Experience has shown that the common video libraries tend to become confused with exact frame indexing if seek to a random frame is requested, so the only way to ensure that frame numbering is consistent is by scanning the video from the very beginning one frame at a time until the desired frame has been reached. However, when processing video corpora where several independent video files are to be processed in a similar manner, the limitations with parallel processing can be overcome by simply performing the computations in a distributed manner. The scripts can be applied as such on several independent processes, each processing one video at a time, possibly on different computers on a computational cluster.

3.2. User interface

SLMotion comes with two user interfaces: a Command-Line Interface (CLI) that is used by default and a graphical frontend for the command-line interface. The Graphical User Interface (GUI) offers a fast and convenient way to use the program for analysing individual videos. If the user has a Python script at hand – either his or her own, or possibly from the community – all the user needs to do is to select the script, the video file to process and any required output files. The GUI has been implemented with the Qt library and can be seen in Fig. 3.

Advanced users may find the command line interface convenient for tasks such as automatic corpus processing. The CLI allows for external scripting, e.g., to run the same processing task on multiple files in parallel on a computation cluster. We have experimented with this setting with our in-lab Condor cluster, consisting of some 100 workstations or 400 CPU cores. The process described in (Karppa et al., 2011) was run in less than 24 hours for the entire Suvi corpus of the Finnish Sign Language dictionary, consisting of some 5500 videos of dictionary signs and example sentences. For example, in order to run the script presented in Listing 1, supposing that the script is stored as `hand_location_test.py`, the user might create the image seen in Fig. 2 to a file

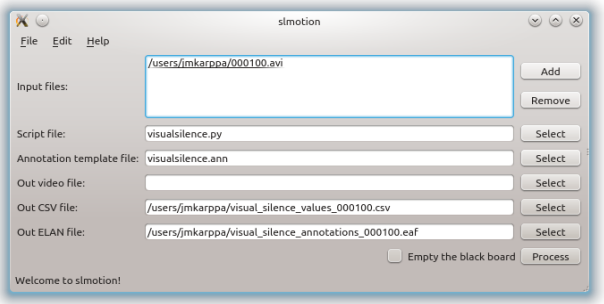

Figure 3: The graphical user interface (GUI).

called `hand_location.png` by analysing a file called `input.png` by issuing the following command:

```
$ slmotion input.png \
    --script hand_location_test.py \
    -o hand_location.png
```

Interface like this should be familiar to UNIX users, making scripting straightforward.

3.3. Interfacing with external systems

The default input format consists of either video files or image sequences. SLMotion also supports Kinect recordings via the OpenNI library, and the depth information is accessible to the analysis components in the same way as the ordinary RGB video data.

As to output, the standard way SLMotion stores features it has extracted from the data is via comma-separated value (CSV) files. If it should be necessary, no restrictions are in place to prevent the user from making more complicated or specialised file formats accessible through Python scripts. The program can also perform various visualisations of the input data at different stages of analysis, and the visualisations can be stored in a video file. These visualisations may include, e.g., drawing binary maps of some low-level processing stage, such as skin detection, or it may be the drawing of circles around detected facial landmark points. Visualisations like this may provide a more intuitive access to computational processes and their results. They may help the researcher make sense of numerical results, or they can be used to assist human inspection, perhaps by highlighting important details of the video, such as blinks or other detected events. Most importantly, they allow the human observer to perform a sanity check on the key parts in the video analysis used to produce the results: the computational methods are never perfect, but a human can often tell immediately if misdetections or other erroneous results have been produced, which may distort the numerical end results.

The program also works in conjunction with ELAN[2] (Wittenburg et al., 2006) which is the standard annotation tool in sign language research. The CSV files created by SLMotion can be included in ELAN projects as such. In addition, the program can create ELAN-compatible EAF annotation files according to user-specified rules, typically depending on the values of the features extracted. An example of automatically generated annotations with our visual silence detector in ELAN can be seen in Fig. 4.

With minor modifications, it could be possible to adapt the program to work as an ELAN video recogniser. However, the recogniser API of ELAN places some limitations: the EAF file format does not offer a support for directly linking time series to the annotation files, making the numerical data less useful than it could be. Also, while the recogniser API can be used to emulate our GUI, transferring data between the programs is somewhat tedious, and the number of annotation tiers that will be added must be fixed on ELAN end while SLMotion could handle an arbitrary number of tiers.

[2] `http://tla.mpi.nl/tools/tla-tools/elan/`

3.4. Component library

We provide a range of native C++ components with SLMotion. Most of these components are directly related to the applications listed in Section 4. We try to make all the code we have used available to the public for scrutiny. Most importantly, this includes our original work, such as the hand-head occlusion detector described in (Viitaniemi et al., 2013) and the head pose estimator described in (Luzardo et al., 2013). The former attempts to detect the locations where one of the hands occludes the face, and the latter attempts to recover the head pose in terms of yaw, pitch, and roll angles. These are available as components called `FaceOcclusionDetector` and `HeadPoseEstimator`, respectively. For the latter component, pre-trained Support Vector Machine (SVM) models are provided along with the program package.

We have also implemented ourselves some applications of methods described in the literature. For example, as parts of the method described in (Karppa et al., 2011), we have implemented a variant of Kanade-Lucas-Tomasi (KLT) tracker (Shi and Tomasi, 1994), building heavily on the parts already implemented in the OpenCV library. The component is available in the library as `KLTTracker`. Within the same work, we also implemented a component for tracking Active Shape Models (ASM) (Cootes et al., 1995), trained to track the silhouettes of skin-coloured blobs in the image corresponding to the hands and the head. This component is available as `AsmTracker`. The library also includes auxiliary components, often needed as a prerequisite by one of the more advanced components. These include `BlobExtractor`, which extracts the contiguous pixel regions, the shape of which is tracked by the previously described `AsmTracker` component. Other helper components include skin detectors based on Extreme Learning Machines (ELM) (Huang et al., 2006), among others.

We also provide wrappers for some pre-existing third party libraries. These include components for performing face detection using the OpenCV implementation of the Viola-Jones cascade classifier (Viola and Jones, 2001). We also have a component for facial landmark detection using the flandmark library by Uřičář et al. (Uřičář et al., 2012). We also have recently added support for generating synthetic hand images with the LibHand (Šarić, 2011) library. This is related to upcoming work on hand configuration estimation.

The Python API also provides a lightweight interface for users to create components of their own without having to touch the source code of the program itself. Examples are provided for getting started. This is a particularly appealing approach in the case of the simpler components, such as the `VisualSilenceDetector`, which detects time sequences where no motion occurs by the means of difference images. An example of the application of this component can be seen in Fig. 4 where the video has been automatically annotated for visual silence.

4. Applications

We have successfully applied the developmental versions of SLMotion in several publications. The initial application was with the method we presented in (Karppa et al., 2011) and focused on kinematic analysis of hand and head movements. The method was built on tracking interest points with the KLT method (Shi and Tomasi, 1994), and simultaneously tracking the silhouette of skin-coloured regions using ASMs (Cootes et al., 1995). The components used in this work, including Viola-Jones face detection (Viola and Jones, 2001), skin detection, and KLT and ASM trackers, are included in the component library (cf. Section 3.4.). Although the method may not be very robust, it can be used as an example of how the modular structure of our framework can be applied to such multistage processes.

The head movement data produced with SLMotion for (Karppa et al., 2011) was used in (Puupponen, 2012) to analyse the Finnish Sign Language head movements from phonetic and linguistic perspectives. (Karppa et al., 2011) was followed by an application of the same method to footage that was simultaneously recorded with motion capture equipment for reference. We performed correlation and regression analysis on the data obtained by our method, and it turned out to perform surprisingly well, despite expected deficits with recovery of depth information. These results were presented in (Karppa et al., 2012).

More recently, we have implemented methods for hand-head occlusion detection within our framework in (Viitaniemi et al., 2013). There we describe a method based on a fusion of local video properties and global hand tracking. The method was tested for locating hand-head occlusions with a corpus of videos from the Suvi, the on-line dictionary of Finnish Sign Language (Finnish Association of the Deaf, 2003), and produced promising results. The method has also been used as part of our baseline solution to the sign spotting benchmark setting, based on the aforementioned dictionary video material, described in (Viitaniemi et al., 2014). The baseline solution builds on Dynamic Time Warping (DTW) (Rabiner and Juang, 1993) matching of spatial non-face skin distribution histograms. All experiments were conducted with SLMotion.

We have also experimented with head pose estimation. In (Luzardo et al., 2013) we described a method based on SVMs for estimating the head pose in terms of yaw, pitch and roll angles. The method was tested with the Pointing04 data set (Gourier et al., 2004). These methods are also available in the component library.

5. Conclusion

With a public release, we hope to create a community of users who contribute to the development, create new components of their own, and share their scripts with us and other users. We will actively continue to develop SLMotion, adding new features as we go along with new work within the context of computer analysis of sign language.

Acknowledgment

This work has been funded by the following grants of the Academy of Finland: 140245, Content-based video analysis and annotation of Finnish Sign Language (CoBaSiL), 251170, Finnish Centre of Excellence in Computational Inference Research (COIN), 34433, Signs, Syllables, and Sentences (3BatS), and 269089, The aspects of the Grammar and Prosody of FinSL (ProGram).

6. References

Auer, Eric, Wittenburg, Peter, Sloetjes, Han, Schreer, Oliver, Masneri, Stefano, Schneider, Daniel, and Tschöpel, Sebastian. (2010). Automatic annotation of media field recordings. In *Proceedings of the ECAI 2010 Workshop on Language Technology for Cultural Heritage, Social Sciences, and Humanities (LaTeCH 2010)*, pages 31–34.

Charles, James, Pfister, Tomas, Everingham, Mark, and Zisserman, Andrew. (2013). Automatic and efficient human pose estimation for sign language videos. *International Journal of Computer Vision*.

Cootes, Timothy F., Cooper, David H., Taylor, Christopher J., and Graham, Jim. (1995). Active Shape Models - Their training and application. *Computer Vision and Image Understanding*, 61(1):38–59, January.

Finnish Association of the Deaf. (2003). `http://suvi.viittomat.net`. First published in 2003, published on a new platform in 2013.

Forster, Jens, Schmidt, Christoph, Hoyoux, Thomas, Koller, Oscar, Zelle, Uwe, Piater, Justus, and Ney, Hermann. (2012). Rwth-phoenix-weather: A large vocabulary sign language recognition and translation corpus. In *Proceedings of 8th Language Resources and Evaluation Conference (LREC 2012)*, pages 3785–3789, Istanbul, Turkey.

Gourier, N., Hall, D., and Crowley, J.L. (2004). Estimating face orientation from robust detection of salient facial st ructures. In *FG Net Workshop on Visual Observation of Deictic Gestures*, pages 1–9.

Huang, G.B., Zhu, Q.Y., and Siew, C.K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, 70(1-3):489–501.

Karppa, Matti, Jantunen, Tommi, Koskela, Markus, Laaksonen, Jorma, and Viitaniemi, Ville. (2011). Method for visualisation and analysis of hand and head movements in sign language video. In Kirchhof, C., Malisz, Z., and Wagner, P., editors, *Proceedings of the 2nd Gesture and Speech in Interaction conference (GESPIN 2011)*, Bielefeld, Germany. Available online as `http://coral2.spectrum.uni-bielefeld.de/gespin2011/final/Jantunen.pdf`.

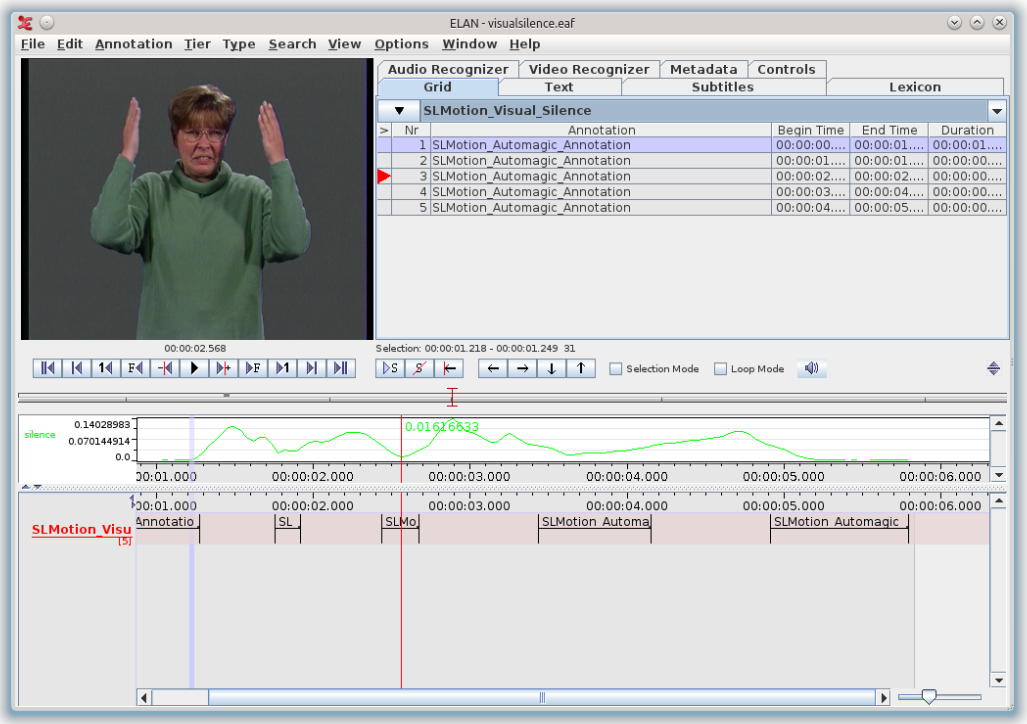

Figure 4: Automatically annotated visual silence in a very short video clip in ELAN. The window shows a typical view where there is one automatically generated tier by placing a threshold on the numerical silence values. The values can be seen plotted in the time series view above.

Karppa, Matti, Jantunen, Tommi, Viitaniemi, Ville, Laaksonen, Jorma, Burger, Birgitta, and De Weerdt, Danny. (2012). Comparing computer vision analysis of signed language video with motion capture recordings. In *Proceedings of 8th Language Resources and Evaluation Conference (LREC 2012)*, pages 2421–2425, Istanbul, Turkey, May. Available online at `http://www.lrec-conf.org/proceedings/lrec2012/pdf/321_Paper.pdf`.

Lenkiewicz, Przemyslaw, Gebre, Binyam Gebrekidan, Schreer, Oliver, Masneri, Stefano, Schneider, Daniel, and Tschöpel, Sebastian. (2012). Avatech – automated annotation through audio and video analysis. In *Proceedings of 8th Language Resources and Evaluation Conference (LREC 2012)*, pages 209–214, Istanbul, Turkey.

Luzardo, Marcos, Karppa, Matti, Laaksonen, Jorma, and Jantunen, Tommi. (2013). Head pose estimation for sign language video. In *Proceedings of the 18th Scandinavian Conference on Image Analysis*, volume 7944 of *LNCS*, Espoo, Finland, June. Springer Verlag.

Puupponen, Anna. (2012). Horisontaaliset ja vertikaaliset päänliikkeet suomalaisessa viittomakielessä [Horizontal and vertical head movements in Finnish Sign Language]. Master's thesis, University of Jyväskylä, Jyväskylä, Finland. In Finnish.

Rabiner, L. R. and Juang, B., (1993). *Fundamentals of speech recognition*, chapter 4. Prentice-Hall, Inc.

Shi, Jianbo and Tomasi, Carlo. (1994). Good features to track. In *Proceedings of IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR '94)*, pages 593–600, June.

Uřičář, Michal, Franc, Vojtěch, and Hlaváč, Václav. (2012). Detector of facial landmarks learned by the structured output SVM. In Csurka, Gabriela and Braz, José, editors, *VISAPP '12: Proceedings of the 7th International Conference on Computer Vision Theory and Applications*, volume 1, pages 547–556, Portugal, February. SciTePress — Science and Technology Publications.

Viitaniemi, Ville, Karppa, Matti, Laaksonen, Jorma, and Jantunen, Tommi. (2013). Detecting hand-head occlusions in sign language video. In *Proceedings of the 18th Scandinavian Conference on Image Analysis*, volume 7944 of *LNCS*, Espoo, Finland, June. Springer Verlag.

Viitaniemi, Ville, Jantunen, Tommi, Savolainen, Leena, Karppa, Matti, and Laaksonen, Jorma. (2014). S-pot – a benchmark in spotting signs within continuous signing. In *Proceedings of 9th Language Resources and Evaluation Conference (LREC 2014)*.

Viola, Paul and Jones, Michael. (2001). Rapid object detection using a boosted cascade of simple features. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01)*, pages I:511–518.

Šarić, Marin. (2011). Libhand: A library for hand articulation. Version 0.9.

Wittenburg, Peter, Brugman, Hennie, Russel, Albert, Klassmann, Alex, and Sloetjes, Han. (2006). Elan: a professional framework for multimodality research. In *Proceedings of LREC 2006, Fifth International Conference on Language Resources and Evaluation*.