Proceedings of the 10th Workshop on the Representation and Processing of Sign Languages (sign-lang@LREC 2022), pages 16–23
Language Resources and Evaluation Conference (LREC 2022), Marseille, 20-25 June 2022

Introducing the `signglossR` Package

Carl Börstell
University of Bergen
Sydnesplassen 7, 5007, Bergen, Norway
carl.borstell@uib.no

Abstract

The `signglossR` package is a library written in the programming language R, intended as an easy-to-use resource for those who work with signed language data and are familiar with R. The package contains a variety of functions designed specifically towards signed language research, facilitating a single-pipeline workflow with R when accessing public language resources remotely (online) or a user's own files and data. The package specifically targets processing of image and video files, but also features some interaction with software commonly used by researchers working on signed language and gesture, such as *ELAN* and *OpenPose*. The `signglossR` package combines features and functionality from many other libraries and tools in order to simplify and collect existing resources in one place, as well as adding some new functionality, and adapt everything to the needs of researchers working with visual language data. In this paper, the main features of this package are introduced.

Keywords: sign language, signed language, multimedia, annotation, R, software

1. Introduction

Signed languages are (unless tactile) primarily visual languages. This differentiates them from spoken languages, which although also multimodal at their core (by virtue of being interactive, contextualized and simultaneously vocal and gestural) have established conventions for representing linguistic form in writing, either as (adapted) orthographic writing – e.g. conventionalized spelling or Jefferson transcription (Jefferson, 2004) – or phonetic transcription – e.g. the International Phonetic Alphabet. Although transcription even for spoken languages is only a partial representation of the multidimensional and highly variable properties of actual speech, the situation for signed languages is arguably much worse. Standard practice for representing signs in writing – since the earliest days of signed language research – has been based on so-called *sign glosses* (Miller, 2006; van der Hulst and Channon, 2010; Frishberg et al., 2012; Crasborn, 2015), representing signs with approximate written language translations rendered in small caps – e.g. TOMATO for the American Sign Language (ASL) sign meaning 'tomato', or even by combining the historical parts of the sign as a compound: RED+SLICE. While some notation systems have been developed to render a visually recognizable form of a sign, most notably *SignWriting* (Sutton, 1996) – shown to be useful for linguists and non-linguist signers alike in representing and recreating sign forms through transcription (Pizzuto et al., 2008) – others have sought to represent sign forms through combinations of symbols each representing a form segment of a sign, most notably the phonemic notation system introduced by Stokoe (1960) or later the phonetic machine-readable strings of *HamNoSys* (Prillwitz et al., 1989; Hanke, 2004). Nonetheless, the dominant convention has arguably been the use of sign glosses, which has continued to be used in corpus annotation work for signed languages (Johnston, 2010; Johnston, 2014; Schembri and Crasborn, 2010). However, sign glossing has been criticized for its complete disregard from representing signed languages in their true modality, with scholars arguing for a practice in which signed language examples are always represented in a visual form – preferably video, but possibly still images (Hochgesang, 2022) or a visually motivated transcription (Pizzuto et al., 2008).

Stepping away from the practice of only using written glosses for signed language examples has been lobbied for by Julie Hochgesang, sometimes under the Twitter hashtag #TyrannyOfGlossing. Later, an attempt to phrase this concretely, the Twitter hashtag #GlossGesang was defined as *"Always present sign language data in a visual format (videos/images) without relying solely on glossing"*. As such, even if sign glosses are used, for reasons of keeping unique, machine-readable labels in a database, they should always be accompanied by a visual representation – e.g. Figure 1.[1]

Figure 1: The ASL sign TOMATOix from ASL Signbank (Hochgesang et al., 2022, 1253).

[1] `https://aslsignbank.haskins.yale.edu/dictionary/gloss/1253.html`

The R package `signglossR` was directly inspired by the work of Julie Hochgesang and was originally intended to help researchers work towards visual representation of signed language data by facilitating the access to tools for collecting and modifying image and video data. In the following sections, I will present the main functionalities of the `signglossR` package and how it can be used to work with signed language data.

2. The `signglossR` package

The `signglossR` package is a library written in the programming language R, intended as an easy-to-use resource for those who work with signed language data and are familiar with R. The package contains a variety of functions designed specifically towards signed language research, facilitating a single-pipeline workflow with R when accessing public language resources remotely (online) or modifying and combining a user's own files and data, such as files locally on your own computer. The `signglossR` package combines features and functionality from several other libraries and tools, either implementations in the R language or external software and tools that need to be installed separately. The goal here has been to simplify and collect existing resources in one place, such that a variety of functions and tools useful for signed language research are all available in a single package for easier workflow without the need to switch between programming languages and without the need for command-line programming. The package specifically targets processing of image and video files (§2.2–2.4), but also features some interaction with software commonly used by researchers working on signed language and gesture, such as *ELAN* (§2.5.1) and *OpenPose* (§2.5.2).

2.1. Installation and Dependencies

2.1.1. Installing `signglossR`

Since the `signglossR` package is not hosted on CRAN[2], it needs to be installed directly from the GitHub repository.[3] In order to do this, users will need to have the `devtools` (Wickham et al., 2021) or `remotes` package (Csárdi et al., 2021) installed, then install the `signglossR` package from the remote GitHub repository:

```
install.packages("devtools")
library(devtools)
# or ...
install.packages("remotes")
library(remotes)

install_github(
  "borstell/signglossR")
```

When the package has been (successfully) installed, it can be accessed in the R environment by loading it in the R session: `library(signglossR)`

[2] `https://cran.r-project.org`

[3] `https://github.com/borstell/signglossR`

2.1.2. Installing Dependencies

The `signglossR` package is built on top of – and combining functions from – several other R packages, which are included as dependencies. However, some of the video and image processing functions depend on additional software external to R. In such cases, the `signglossR` functions will run commands on the local system externally in the background, which requires an external (prior) install of these tools: for video processing functions, this concerns `FFmpeg` (FFmpeg Team, 2022); for image processing, this concerns `ImageMagick` (ImageMagick Development Team, 2021), when possible in its R implementation using the `magick` package (Ooms, 2021). Running certain video or image processing functions in `signglossR` would therefore result in errors if these dependencies are not installed on the computer executing the code: follow instructions on their respective websites regarding installation!

These tools/packages are very powerful on their own and the main benefit of running them through `signglossR` is to facilitate the work for users familiar with R, but inexperienced or uncomfortable with working directly in the command line. Furthermore, the `signglossR` package was written to align with the tidy-style workflow using the `magrittr` (Bache and Wickham, 2022) piping function `%>%`, such that the output of one function can be used as input for another, creating a sequence of multiple operations.

2.2. Accessing Online Resources

As of the current version of `signglossR` (v2.2.2), the package has functions to access data from three signed language resources: the Swedish Sign Language dictionary *Svenskt teckenspråkslexikon* (Svenskt teckenspråkslexikon, 2022) (§2.2.1); the ASL dictionary *ASL Signbank* (Hochgesang et al., 2022) (§2.2.2); and the Swedish Sign Language (STS) Corpus *Svensk teckenspråkskorpus* (Öqvist et al., 2020) (§2.2.3). These are resources freely available online and whose maintainers have been informed about the access and processing functions of `signglossR`: users are advised to acknowledge these sources accordingly – try the `cite_source()` function! – and follow their respective terms of use. Using material from other sources should be done according to *their* terms and license.

2.2.1. The Swedish Sign Language Dictionary

Svenskt teckenspråkslexikon[4] (Svenskt teckenspråkslexikon, 2022) is an online dictionary of Swedish Sign Language (*svenskt teckenspråk*, STS). The dictionary contains some 17 000 sign entries, many with form variants, along with example videos of signs used in sentences, still images of the sign, phonemic transcription, a unique ID number and sign glosses used for the corpus and dictionary projects (Mesch et al., 2012).

[4] `https://teckenspraklexikon.su.se`

With `signglossR`, you can convert ID numbers to sign glosses – and vice versa – using the `id2gloss()` and `gloss2id` functions. For example:

```
> id2gloss(123)
[1] "VEM"
> gloss2id("VEM")
[1] "00123"
```

These functions work better from ID to gloss than the reverse: the IDs are always unique, whereas the glosses are simply searched for in the database and can result in multiple (or no) string matches.
The function `get_image()` downloads a video based on the unique ID, saves the file to your local computer (destination path can be specified) and outputs the filename to the console. It is also possible to combine with the previous function, to go from sign gloss to ID and fetch that sign image.

```
> get_image(1241)
>
> get_image(gloss2id("VEM"))
```

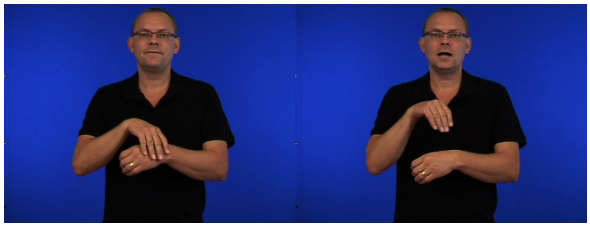

Figure 2: The sign SVERIGE ('Sweden') (Svenskt teckenspråkslexikon, 2022, 1241).

As can be seen from Figure 2, the sign SWEDEN contains multiple still images in the dictionary, and thus they are combined automatically through the `signglossR` function into a side-by-side image as a single file. However, it is also possible to generate an overlay image. Here, the function runs an external `ImageMagick` command that takes the first image at 25% opacity and overlays it onto the second image, creating a "ghost" outline of the earlier part of the sign and a clear image of the later part – see Figure 3.

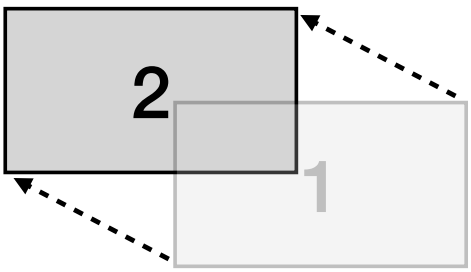

Figure 3: Producing an overlay image from two images with a 25% opacity "ghost" outline for the first image.

This can be achieved directly when downloading a file from the dictionary using the following function call:

```
> get_image(1241,
            overlay = TRUE)
```

This creates the following output, which can be useful as it takes up less horizontal space and illustrates the dynamic movement in a single frame – see Figure 4.

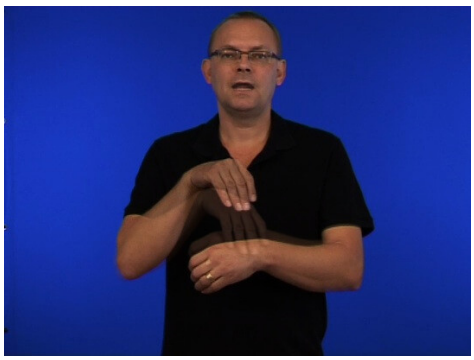

Figure 4: The sign SVERIGE ('Sweden') (Svenskt teckenspråkslexikon, 2022, 1241) with overlay.

Additionally, the functions `get_video()` and `get_gif()` can download a sign as video (`.mp4`) or GIF (`.gif`) directly:

```
> get_video(1241)
>
> get_gif(1241)
```

2.2.2. ASL Signbank

The second dictionary that can be accessed directly through `signglossR` is *ASL Signbank* (Hochgesang et al., 2022). The dictionary has videos and still images of signs, and both of these can be accessed directly. Converting between sign glosses and ID numbers is also possible. Note that the default language selection in `signglossR` is STS, so using general functions to access ASL resources requires specifying the language using `acronym = "ASL"`:

```
> id2gloss(1253,
           acronym = "ASL")
[1] "TOMATOix"
> gloss2id("TOMATOix",
           acronym = "ASL")
[1] "1253"
```

For the ASL Signbank images, we can directly specify that we want the sign gloss to be added to our downloaded image using `glosstext = TRUE`, and modify details about the fontsize and location of the label (`"southwest"` means bottom left) – the resulting image of the following code was seen in Figure 1:

```
> get_image(1253,
      acronym = "ASL",
```

```
            glosstext = TRUE,
            fontsize = 30,
            gravity = "southwest")
```

As with the STS dictionary, ASL Signbank sign videos can be downloaded directly using the `get_video()` function, again specifying `acronym = "ASL"`.

2.2.3. The Swedish Sign Language Corpus

The Swedish Sign Language (STS) Corpus (Mesch et al., 2012; Mesch et al., 2014) and specifically its online interface (Öqvist et al., 2020)[5] can be accessed using the function `search_corpus()`, for which a sign gloss is the input, and running the command opens a browser tab with the search hits for that sign gloss.

```
> search_corpus(id2gloss(1241))
```

2.3. Image Processing

Besides accessing image files directly from language resources online, `signglossR` also contains functions to process such files, either as part of the pipeline of accessing them from those resources, or applied to any image file locally.

2.3.1. Crop and Annotate

Trimming images can be done directly with the `get_image()` function, using the `trim` argument (`trim = .6` means 40% of the total width is cropped, equally on both sides with the image centered at the middle). This can be particularly useful when signs have many still images and you want side-by-side outputs but with efficient use of horizontal space – this is illustrated in Figure 5 in which the sign KÖPENHAMN ('Copenhagen') has four images representing the sign and each image is cropped to 60% of the original width:

```
> get_image(9979, trim = .6)
```

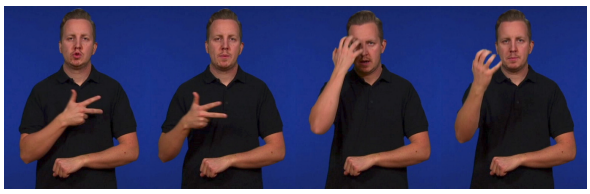

Figure 5: The sign KÖPENHAMN ('Copenhagen') (Svenskt teckenspråkslexikon, 2022, 9979).

Trimming and annotating can also be done on any image file using the `make_image_ex()` function and specifying a region to crop and a text annotation:

```
> make_image_ex("path/to/image",
            crop = TRUE,
            region = "400x300",
            text = "GLOSS")
```

[5] https://teckenspraksleXikon.su.se/

2.3.2. Combine Images

Combining and overlaying images can also be done with local files. For example, if we try to directly create an overlay of the KÖPENHAMN sign from Figure 5, the result is what we find in Figure 6.

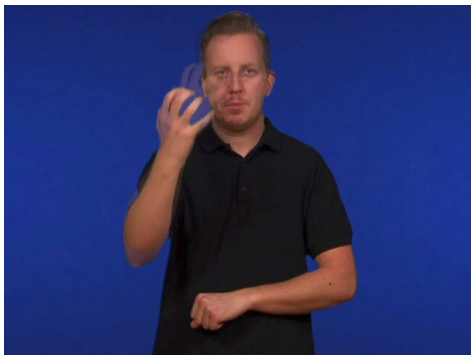

Figure 6: The sign KÖPENHAMN ('Copenhagen') (Svenskt teckenspråkslexikon, 2022, 9979) with overlay.

Because the ghost overlay is performed recursively, earlier frames are too weak to come through in the final output. Instead, we could process each individual image and combine them in stages, as in Figure 7:

```
> combine_images(
    c("image1", "image2"),
    overlay = TRUE,
    trim = .6)
> combine_images(
    c("image3", "image4"),
    overlay = TRUE,
    trim = .6)
> combine_images(
    c("image1-2", "image3-4"))
```

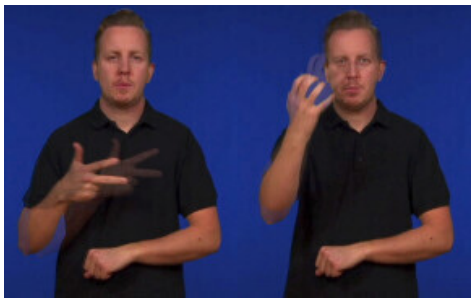

Figure 7: The sign KÖPENHAMN ('Copenhagen') (Svenskt teckenspråkslexikon, 2022, 9979) with segmented overlay.

Alternatively, images can also be combined vertically, allowing for a vertical stack to save horizontal space:

```
> combine_images(
    c("image1", "image2"),
```

```
    trim = .6)
> combine_images(
    c("image3", "image4"),
    trim = .6)
> combine_images(
    c("image1-2", "image3-4"),
    stack = TRUE)
```

Figure 8: The sign KÖPENHAMN ('Copenhagen') (Svenskt teckenspråkslexikon, 2022, 9979) with segments stacked.

2.3.3. Censor Images

Sometimes we need to anonymize our data, for instance by blurring the face of the signer in an image. With `signglossR`, this can be achieved by using the `censor_image()` function, either by manually specifying the region to be censored (by an opaque square or by blurring), or by using an automated method from the `opencv` package (Ooms and Wijffels, 2021). As long as the face is not obstructed (e.g. by the signer's own hands), the automatic method works quite well, as illustrated in Figure 9. This function works with any local file, including one just piped as it was accessed from an online resource:

```
> get_image(1241,
            overlay = TRUE) %>%
    censor_image()
```

2.4. Video Processing

Besides the `get_video()` function described in §2.2, there are several functions to modify video files in different ways, described below.

2.4.1. Playback Speed and Repetition

The `make_video_ex()` function can be used to modify video files. For example, piping our SVERIGE sign

Figure 9: The sign SVERIGE ('Sweden') (Svenskt teckenspråkslexikon, 2022, 1241) with overlay and face censoring.

video, the first of the following commands would result in the video file being slowed down to 40% speed, and the second would result in a video file played once at original speed, then repeated at 30% speed:

```
> get_video(1241) %>%
    make_video_ex(speed = .4)
> get_video(1241) %>%
    make_video_ex(speed = .3,
                  rep = TRUE)
```

2.4.2. Making GIFs

The `make_gif()` function can be used to convert a video file to a GIF.

```
> get_video(1253,
            acronym = "ASL") %>%
    make_gif()
```

2.4.3. From ELAN to Multimedia Examples

Arguably the most advanced functions in the `signglossR` package are `make_elan_image()` and `make_elan_video()`, both of which input an ELAN file in order to generate image and video files, respectively, for use as linguistic examples (e.g. for a paper or presentation). How they work is that they input an ELAN file with specifications of a segmentation tier and a gloss tier, and then goes through these to create image or video outputs. Figure 10 shows a mock example, using data from the STS Corpus (Öqvist et al., 2020) with English glosses for illustration. Here, there are two tiers visible: *segment* and *gloss*.

What the ELAN segmentation function in `signglossR` does is that it groups annotation cells (annotations on the *gloss* tier) according to the segmentations made on the segmentation tier (annotations on the *segment* tier) – see Figure 11.

Using `make_elan_video()` with our mock example, the output would be a shorter video file spanning only the segment duration (if several segments are found, each can output a separate video) and annotating

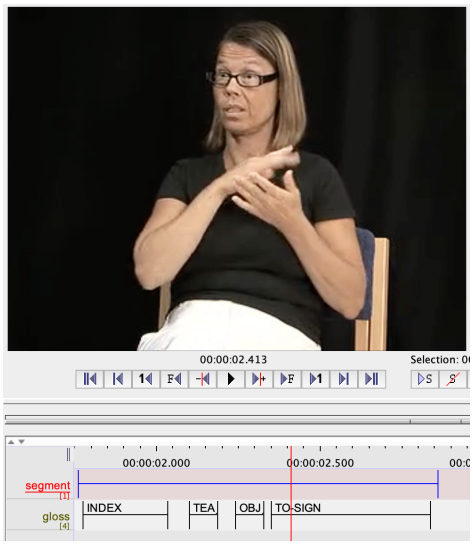

Figure 10: ELAN annotation view using a file from the STS Corpus (Öqvist et al., 2020) with glosses re-annotated in English. Original video sequence: SSLC01_021 00:04:08.

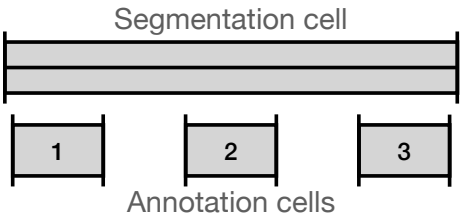

Figure 11: Schematic representation of annotation cells within the same segment.

the video with the sign glosses – thus, it can be used as a basic subtitling tool when illustrating examples.

```
> make_elan_video("file.eaf",
    segmentation_tier = "segment",
    gloss_tier1 = "gloss")
```

The function `make_elan_video()` retrieves the video path from within the ELAN file, but in case the video file is located elsewhere or there are multiple associated video files, users are advised to specify the video path explicitly as an argument to the function.
With `make_elan_image()`, the output will be an image sequence with each sign (what is segmented on the gloss tier) represented by an image. Here, there is an additional option of also creating overlaps for individual images. In this case, it selects the first and last frames of each gloss duration (the 1st and n^{th} frames of a sign of length n; see Figure 12) and creates an overlay for each, after which each sign image is combined into a horizontal sequence, as illustrated in Figure 13. If overlay is not selected (`combine = FALSE`), only the first frame of each sign is selected.

```
> make_elan_image("file.eaf",
    segmentation_tier = "segment",
    gloss_tier = "gloss")
```

The function `make_elan_image()` calls `make_image_ex()` internally and can therefore include a step of, for example, cropping each frame before combining. Furthermore, the output of these – and any image-generating – function can be piped to, e.g., `censor_image()` for an additional processing step censoring the face(s) in the image.

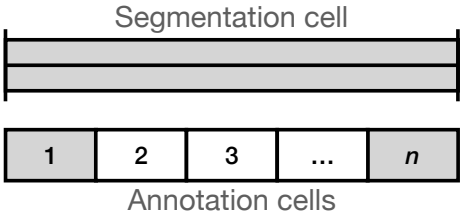

Figure 12: Schematic representation of video frames within the same annotation duration.

2.5. Processing Other File Types

In addition to image and video processing, there are a few functions in the `signglossR` package that aim to facilitate direct processing of file formats commonly found in signed language linguistics, namely ELAN files (§2.5.1) and OpenPose files (§2.5.2).

2.5.1. ELAN

ELAN (Brugman and Russel, 2004; Wittenburg et al., 2006; Crasborn and Sloetjes, 2008; ELAN (Version 6.3) [Computer software], 2022) is an annotation tool used by many researchers working on multimodal data, from signed languages to gesture and behavioral psychology. ELAN annotation files (`.eaf`) are underlyingly XML (`.xml`) files and can be processed as such. The `signglossR` function `read_elan()` inputs a path to a directory, parses all ELAN files in that directory and outputs a data frame (or, tibble) with the data in a typical long format of rows and columns, which lets the researcher get direct access to ELAN files from R without any intermediate export.

```
> path <- "path/to/directory"
> data <- read_elan(path)
```

2.5.2. OpenPose

OpenPose (Cao et al., 2017; Cao et al., 2019) is an open source tool used to input video (or image) files in order to estimate the body pose of any humans depicted, with keypoints for various anchor points on the body (e.g. nose, mouth, shoulders, wrists, etc.). This software has only just started gaining traction within signed language research, where it can be used to estimate signing activity and the location of hands/arms in signing space from videos. For each frame in a video, OpenPose outputs a JSON (`.json`) file linking each individual in the video to a key–value format

Figure 13: The output of `make_elan_image()` on a sequence from the Swedish Sign Language (STS) Corpus (Öqvist et al., 2020) with glosses re-annotated in English in ELAN, each sign represented by first-and-last frames overlaid and annotated with respective glosses and concatenated. Original video sequence: SSLC01_021 00:04:08

of keypoints and their respective estimated locations in the frame. The function `read_openpose()` inputs a path to a directory, parses all JSON files in that directory and outputs a data frame with the data either in a wide format (`wide = TRUE` is default), where each row represents a frame and each keypoint has its own column, or in a long format of rows and columns with each datapoint on a single row.

```
> path <- "path/to/directory"
> wide_data <-
    read_openpose(path)
> long_data <-
    read_openpose(path, wide=F)
```

3. Final remarks

In this paper, I have given a brief overview to the `signglossR` package and its functionality. Originally created with the intention to facilitate the use of visual representation of signed language data, it has grown to encompass other aspects of working with multimodal data and related software – e.g. ELAN – specifically in R, a language popular among linguists but with few packages dedicated to signed language data. It is then hopefully easier to proceed with work on such data with other R packages designated for, e.g., statistical analyses, although there is also more functionality and improvement in the planning stage by the author already. Since all the code is open source – just as most of the packages and software it is built on – other users are free to use, adapt or expand on the functionality and ideas of `signglossR`. Furthermore, if anyone using the package would encounter problems or wishes to relay feedback on their user experience, they are more than welcome to contact the author directly or file an issue in the GitHub repository.

4. Acknowledgments

Thanks to Julie Hochgesang whose presentation on glossing in Berlin on September 23rd 2019 (and advocacy using the Twitter hashtags #TyrannyOfGlossing and #GlossGesang) inspired a first (Python) prototype of an ELAN-to-image example generator. Thanks to three anonymous reviewers for comments on this paper. A special thanks to Jon Keane (`https://github.com/jonkeane`) for assistance and suggestions after the very first `signglossR` package launch in 2020.

5. Bibliographical References

Bache, S. M. and Wickham, H., (2022). *magrittr: A Forward-Pipe Operator for R*. R package v. 2.0.2.

Brugman, H. and Russel, A. (2004). Annotating multimedia/multi-modal resources with ELAN. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC 2004)*, pages 2065–2068.

Cao, Z., Simon, T., Wei, S.-E., and Sheikh, Y. (2017). Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*.

Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S., and Sheikh, Y. A. (2019). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Crasborn, O. and Sloetjes, H. (2008). Enhanced ELAN functionality for sign language corpora. In Onno Crasborn, et al., editors, *Proceedings of the LREC2008 3rd Workshop on the Representation and Processing of Sign Languages: Construction and Exploitation of Sign Language Corpora*, pages 39–43, Marrakech, June. European Language Resources Association (ELRA).

Crasborn, O. (2015). Transcription and Notation Methods. In Eleni Orfanidou, et al., editors, *Research Methods in Sign Language Studies*, pages 74–88. John Wiley & Sons, Ltd, Chichester.

Csárdi, G., Hester, J., Wickham, H., Chang, W., Morgan, M., and Tenenbaum, D., (2021). *remotes: R Package Installation from Remote Repositories, Including 'GitHub'*. R package v. 2.4.2.

ELAN (Version 6.3) [Computer software]. (2022). Max Planck Institute for Psycholinguistics, The Language Archive, Nijmegen.

FFmpeg Team. (2022). FFmpeg, v. 5.0.1.

Frishberg, N., Hoiting, N., and Slobin, D. I. (2012). Transcription. In Roland Pfau, et al., editors, *Sign language: An international handbook*, pages 1045–1075. De Gruyter Mouton, Berlin/Boston, MA.

Hanke, T. (2004). HamNoSys – Representing Sign Language Data in Language Resources and Language Processing Contexts. In Oliver Streiter et al., editors, *Proceedings of the LREC2004 Workshop on the Representation and Processing of Sign Languages: From SignWriting to Image Processing. Information techniques and their implications for teaching, documentation and communication*, pages 1–6, Lisbon, May. European Language Resources Association (ELRA).

Hochgesang, J. (2022). Managing Sign Language Acquisition Video Data: A Personal Journey in the Organization and Representation of Signed Data. In Andrea L. Berez-Kroeker, et al., editors, *The Open Handbook of Linguistic Data Management*, pages 367–383. The MIT Press.

ImageMagick Development Team. (2021). ImageMagick, v. 7.0.10.

Jefferson, G. (2004). Glossary of transcript symbols with an introduction. In Gene H. Lerner, editor, *Pragmatics & Beyond New Series*, volume 125, pages 13–31. John Benjamins Publishing Company, Amsterdam.

Johnston, T. (2010). From archive to corpus: Transcription and annotation in the creation of signed language corpora. *International Journal of Corpus Linguistics*, 15(1):106–131.

Johnston, T. (2014). The reluctant oracle: Adding value to, and extracting of value from, a signed language corpus through strategic annotations. *Corpora*, 9(2):155–189.

Mesch, J., Wallin, L., and Björkstrand, T. (2012). Sign language resources in Sweden: Dictionary and corpus. In Onno Crasborn, et al., editors, *Proceedings of the LREC2012 5th Workshop on the Representation and Processing of Sign Languages: Interactions between Corpus and Lexicon*, pages 127–130, Istanbul, May. European Language Resources Association (ELRA).

Miller, C. (2006). Sign language: Transcription, notation, and writing. In Keith Brown, editor, *Encyclopedia of Language & Linguistics*, number 1988, pages 353–354. Elsevier, Oxford.

Ooms, J. and Wijffels, J., (2021). *opencv: Bindings to 'OpenCV' Computer Vision Library*. R package v. 0.2.1.

Ooms, J., (2021). *magick: Advanced Graphics and Image-Processing in R*. R package v. 2.7.3.

Öqvist, Z., Riemer Kankkonen, N., and Mesch, J. (2020). STS-korpus: A sign language web corpus tool for teaching and public use. In Eleni Efthimiou, et al., editors, *Proceedings of the LREC2020 9th Workshop on the Representation and Processing of Sign Languages: Sign Language Resources in the Service of the Language Community, Technological Challenges and Application Perspectives*, pages 177–180, Marseille, May. European Language Resources Association (ELRA).

Pizzuto, E. A., Chiari, I., and Rossini, P. (2008). The representation issue and its multifaceted aspects in constructing sign language corpora: Questions, answers, further problems. In Onno Crasborn, et al., editors, *Proceedings of the LREC2008 3rd Workshop on the Representation and Processing of Sign Languages: Construction and Exploitation of Sign Language Corpora*, pages 150–158, Marrakech, June. European Language Resources Association (ELRA).

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, Hamburg.

Schembri, A. and Crasborn, O. (2010). Issues in creating annotation standards for sign language description. In Philippe Dreuw, et al., editors, *Proceedings of the LREC2010 4th Workshop on the Representation and Processing of Sign Languages: Corpora and Sign Language Technologies*, pages 212–216, Valletta, May. European Language Resources Association (ELRA).

Stokoe, W. C. (1960). Sign language structure: An outline of the visual communication system of the American Deaf. In *Studies in linguistics: Occasional papers (No. 8)*, Buffalo, NY. Dept. of Anthropology and Linguistics, University of Buffalo.

Sutton, V. (1996). SignWriting. `https://www.signwriting.org`.

van der Hulst, H. and Channon, R. (2010). Notation systems. In Diane Brentari, editor, *Sign Languages*, pages 151–172. Cambridge University Press.

Wickham, H., Hester, J., Chang, W., and Bryan, J., (2021). *devtools: Tools to Make Developing R Packages Easier*. R package v. 2.4.3.

Wittenburg, P., Brugman, H., Russel, A., Klassmann, A., and Sloetjes, H. (2006). ELAN: A professional framework for multimodality research. In *Proceedings of the 5th International Conference on Language Resources and Evaluation (LREC 2006)*, pages 1556–1559.

6. Language Resource References

Hochgesang, J. A., Crasborn, O., and Lillo-Martin, D. (2022). ASL Signbank. Haskins Lab, Yale University, `https://aslsignbank.haskins.yale.edu/`.

Mesch, J., Wallin, L., Nilsson, A.-L., and Bergman, B. (2012). *Dataset. Swedish Sign Language Corpus project 2009–2011 (version 1)*. Department of Linguistics, Stockholm University.

Mesch, J., Rohdell, M., and Wallin, L. (2014). *Annotated files for the Swedish Sign Language Corpus. Version 2*. Department of Linguistics, Stockholm University.

Svenskt teckenspråkslexikon. (2022). Svenskt teckenspråkslexikon. Department of Linguistics, Stockholm University, `https://teckenspraksLexikon.ling.su.se/`.