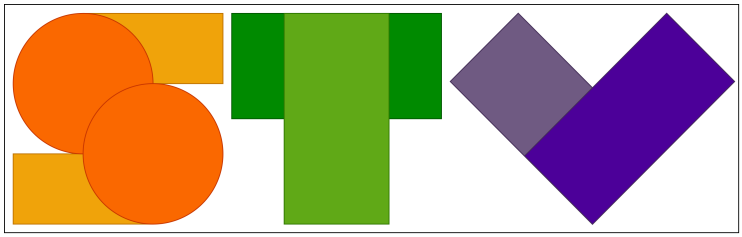
Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG



StudyToolkitVid

A toolkit for creating studies to evaluate quality of videos

Digital and Data Literacy in Teaching Lab

Created by:

Christian Schuler
Dominik Hauser
Anran Wang

Mentor: Prof. Dr. Timo Baumann, OTH Regensburg

12.10.2023

Contents

1 User Guide

1.1 Pre-Requirements

In order to install StudyToolKitVid, Python (3.10+) is needed.
It is advised to install git to easily clone the repository.

1.2 Installation

1.2.1 Linux/Ubuntu (tested on 22.04)

Listing 1.1: Cloning the repository,

```
1 git clone https://github.com/christianschuler8989/StudyToolkitVid
```

Listing 1.2: Virtual Python Environment,

```
1 python -m venv venvStudyToolkitVid
2 source venvStudyToolkitVid/bin/activate
```

Listing 1.3: Install Python packages,

```
1 pip install -r requirements.txt
```

1.2.2 Windows (tested on Windows 10)

Navigate to the directory where you want to install StudyToolKitVid and clone the repository via the command line and according git command:

Listing 1.4: Cloning the repository,

```
1 git clone https://github.com/christianschuler8989/StudyToolkitVid
```

Or: Manually download the repository via the GitHub page.

Navigate to the StudyToolKitVid directory with a command line and install the required packages:

Listing 1.5: Install Python packages,

```
pip install -r requirements.txt
```

1.3 Setup

Listing 1.6: Setup the Toolkit's workspace,

```
python main.py -s
```

Listing 1.7: Test the Toolkit,

```
python main.py -t -n 1 2 3
```

1.4 A Toolkit in Three Parts

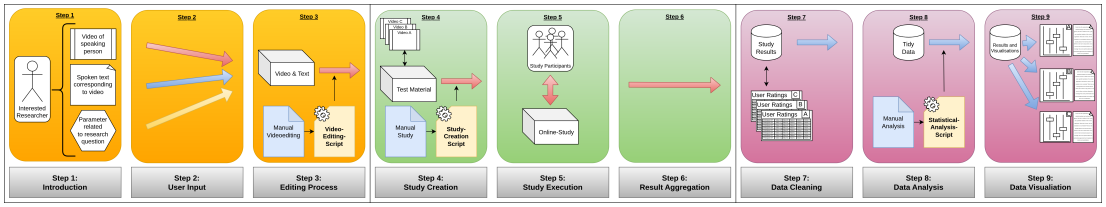


Figure 1.1: Modular concept of the researchers' toolkit StudyToolkitVid. In 3 parts with 3 steps each.

Listing 1.8: Run the Toolkit,

```
python main.py -r
```

1.5 Part 1 - Media Editing

1.5.1 Step 1 - Introduction

Media Editing is a Python script primarily designed for video editing, with a specific focus on its application in lip synchrony studies. While its primary function

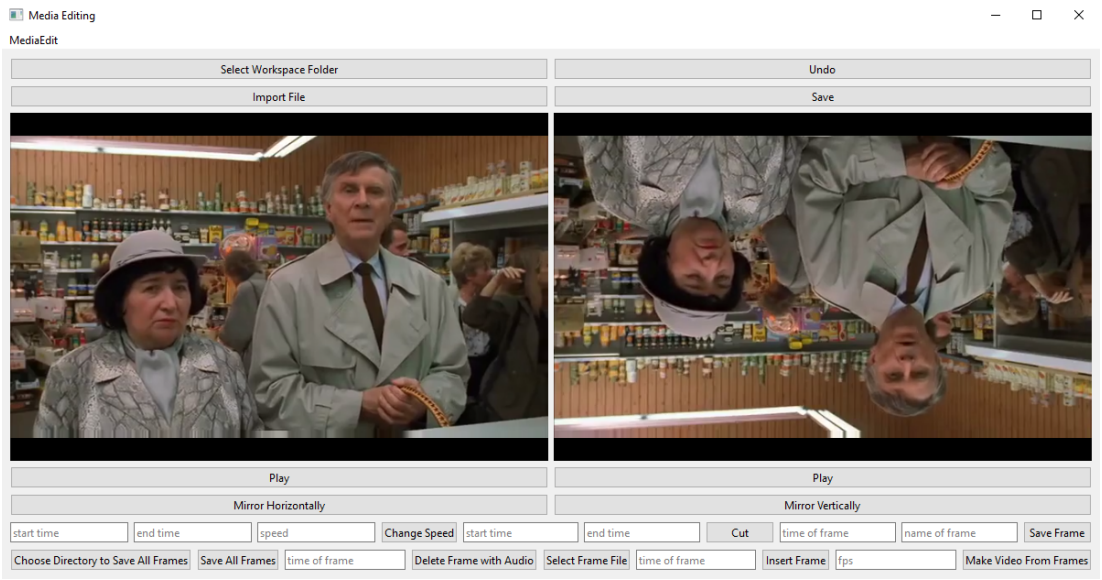


Figure 1.2: User interface for the Media Editing script. Left: original video & Right: edited video.

is video editing, it also includes audio editing capabilities. This script allows for precise manipulation of both video and audio content.
Key features include the creation of test material for lip synchrony studies: Media Editing offers specialized tools tailored to the precise requirements of lip synchrony studies. Additionally, the script provides audio editing functionalities, allowing users to align audio content with their project needs.
With a strong emphasis on precision, Media Editing ensures that editing tasks are carried out with the utmost accuracy. This precision is especially valuable for scientists and researchers who rely on the software to edit video snippets for testing and research purposes.

1.5.2 Step 2 - User Input

The User Interface (UI) for the Media Editing script can be seen in Figure 1.2.
To use the UI, start by selecting a workspace folder. This folder serves as the location for storing temporary files created during the editing process. Next, choose the file you want to edit. You can then play or pause the video or audio file using the play/pause button. In the top-right corner of the UI, you'll find two buttons for undoing the last (up to 5) steps and a button to save the current edited media file.
The editing functions are located at the bottom of the interface:

- Mirror Horizontally / Vertically: Mirrors the video either horizontally (x-

axis) or vertically (y-axis).

- Change Speed: Adjust the speed of a subsection of the media by specifying a starting and ending time, as well as the desired speed (multiplicative). It's recommended not to go below 0.5 speed to avoid introducing significant artifacts. For short sequences it is recommended to choose a high speed factor since the audio will be disappearingly small and the processing cannot continue.
- Cut: Create a sub-clip of the media file by specifying start and end times.
- Save Frame: Save a single frame at a specific timestamp with a given name to a designated folder.
- Save All Frames: Initiate 'Choose Directory to Save All Frames' to select a directory, then save all frames to the chosen location with numerical filenames for easy sorting.
- Delete Frame with Audio: Remove a frame and its corresponding audio snippet (based on frames per second).
- Insert Frame: Use 'Select Frame File' to choose a frame that will be inserted into the video at a specified time.
- Make Video From Frames: Given a folder containing images (frames) and a desired FPS (Frames Per Second), create a video.

The default file formats are mp4 for videos and mp3 for audio data, but you have the option to change them to less compressed formats if needed. This is still work in progress for the UI but is implemented in the Media Editing script.

1.5.3 Step 3 - Editing Process

Some of the editing as well as the user input has been described in Step 2. Now we will look at the script instead of the UI. The script is based on moviepy and ffmpeg. Furthermore we require numpy, imageio, and textgrid to be installed (see Installation 1.2).
The script contains one class "editing" which can be initialized with a path to a file and a default path for saving. For this documentation the editing class will be called edit. This is an example for an initialization:

```
edit = editing('.../StudyToolkitVid/examples/mediaEditing/lohse.mp4',
               '.../StudyToolkitVid/examples/out/')
```

The functions are described below.

getClip(): returns the currently clip - moviepy-video/audioclip.
Usage: edit.getClip() -> returns moviepy-clip.

getAudioClip(): returns the audio clip of the current clip.
Usage: edit.getAudioClip() -> returns moviepy-audioclip.

setAudioClip(path_to_clip): exchanges the current audio clip with the audio file located at paht_to_clip.
Usage: edit.setAudioClip('.../StudyToolkitVid/examples/audio.mp3')

undo(): undo-function for the last editing steps. Can be called up to 5 times. Stores the edited versions of the clips in the memory.
Usage: edit.cut(0.5, 1.5) -> edit.undo() -> clip is now identical to the clip prior the cut-operation.

cut(star, end): cuts the clip to a subclip beginning at start and up to end (float input). Negative input is either 0 (for start < 0) or clip.duration (for end < 0 or end $>$ clip.duration).
Usage: edit.cut(0.36, 12.94) -> the current clip is now a snippet from the original clip beginning at 0.36 and ending at 12.94 of the original clip.

getFrame(frame_time): returns an image which is the frame at the given frame_time (float).
Usage: edit.getFrame(1.33) -> returns an numpy array (image) of the frame that is closest to the given frame time.

saveFrame(frame_time, frame_name, extension = 'png'): saves a frame at a given frame_time with the given frame_name as a png file to the default output directory. The extension parameter can be adjusted for other file types.
Usage: edit.saveFrame(0.32, 'OpenMouth') -> saves the frame as 'OpenMouth.png' to the default save directory.
edit.saveFrame(1.43, 'trail5', extension = 'jpg') -> saves the frame as 'trail5.jpg' to the default save directory.

saveAllFrames(path_optional = '', extension = 'png'): saves all frames

of the video to the default save directory. The save directory can be specified as well as the extension of the image files.
Usage: edit.saveAllFrames() -> saves all frames as png in the default save directory.
edit.saveAllFrames(path_optional = '.../StudyToolkitVid/examples/frames/, extension = 'gif') -> saves all frames to the given directory with a .gif extension.

removeFrameTime(frame_time): removes a frame at a specific time while also removing the audio that plays during this frame.
Usage: edit.removeFrameTime(3.33) -> removes the frame that is closest to the timestamp 3.33 (including audio).

removeFrameIndex(frame_index): removes a frame with a given index (integer) including the audio that plays during the frame.
Usage: edit.removeFrameIndex(3) -> removes the third frame and the audio that plays during this frame.

insertFrame(frame_path, frame_time): inserts a frame at the frame_time without changing the audio (makes it asynchronous).
Usage: edit.insertFrame('.../StudyToolkitVid/examples/frame123.png', 12.33) -> inserts the frame 'frame123.png' at the 12.33 second.

saveClip(name_of_clip, path_optional =", extension ="): saves the current clip with a given name. The path can be optionally be defined, else it will be saved to the default save directory. The extension will be set to mp4 for video and mp3 for audio if not specified. If specified, it has to fit the type of the clip, i.e., for a video it could be mp4.
Usage: edit.saveClip('trail15-open') -> saves the current clip as 'trail15-open.mp4' or 'trail15-open.mp3' depending if it is a sound or a video clip.
edit.saveClip('trail15-open', extension = 'mkv') -> saves the current clip as 'trail15-open.mkv' in the default save folder !BUT! requires the clip to be a video clip.

makeVideoFromFrames(frames_path, frames_per_second): creates a video from a folder that contains images (frames). **!Requires!** (as of now) the images to be png, jpg or jpeg. The frames per second is required to determine the time each frame will be shown and thus the duration of the clip.
Usage: edit.makeVideoFromFrames('.../StudyToolkitVid/examples/manyFrames/',

30) -> creates a video from all frames that can be found in the folder manyFrames and plays them at 30 frames per second.

mirrorAtX(): mirrors a video at the X-Axis (horizontally).
Usage: edit.mirrorAtX() -> mirrors the video horizontally.

mirrorAtY(): mirrors a video at the Y-Axis (vertically).
Usage: edit.mirrorAtY() -> mirrors the video vertically.

changeSpeed(speed, start, end): changes the speed of the current clip for a specific subsection. The speed is technically not limited but too large values for short clips will result in the clip forced to a duration of 0, i.e., non existent anymore. For speeds under 0.5 many artifacts will be introduced.
Usage: edit.changeSpeed(1.231, 0.4, 2.1) -> changes the speed (multiplicative by 1.231) of the clip from 0.4 seconds to 2.1 seconds; the rest will be unchanged.

deleteFrameSynchronous(frame_time): deletes a frame at the given frame time and makes the video synchronous again by fusing the previous and next frame to a new frame. The audio will be unchanged.
Usage: edit.deleteFrameSynchronous(2.1) -> deletes the frame and inserts a new -fused- frame in the same place while leaving the audio untouched.

changeSpeedSegmentsAudio(intervals, speed_factor): changes the speed of multiple segments in the video. If two segments are not positioned directly after each other, the unchanged video/audio will play. Segments are given as [[start_1, end_1], [start_2, end_2], ...]
Usage: edit.changeSpeedSegmentsAudio([[0.5, 0.54], [0.62, 0.66], [1.2, 1.32]], 1.5) -> changes the speed of the three given segments by speeding them up by 50% (1.5 times).

saveAudioFromTextGrid(row, path_textgrid, path_to_save_audios, start_index, end_index): saves audio snippets according to a textgrid for the clip (video or audio). The textgrid needs to be a Praat-style textgrid where the row defines the coarseness of the text (words, phonemes, etc.) The path to the textgrid has to be given and a path to save all the audio snippets.
Usage: edit.saveAudioFromTextGrid(0, '.../StudyToolkitVid/examples/mediaEditing/lohse.TextGrid', .../StudyToolkitVid/examples/mediaEditing/audios/') -> ex-

tracts the words (row 0) from the current clip (here: example lohse.mp4) from the according textgrid and saves them to the folder audios.

createAudioFromAudios(name_of_file, path_to_audios, path_to_save = '', extension = ''): creates one audio clip from many audios and saves it. **!Does not!** exchange the current clip with the newly created audio. Requires the name of the file that will be saves, the path where the audios are located, and the path where to save.
Usage: edit.createAudioFromAudios('composedAudio', '.../StudyToolkitVid/examples/mediaEditing/audios/', '.../StudyToolkitVid/examples/mediaEditing/') -> saves a composed audio with the name 'composedAudio.mp3' in the mediaEditing folder. The extension could be changed to another audio format

getTextGridInformation(text_of_interest, path_textgrid, path_save): extracts the 'text_of_interest' in the current clip according to a textgrid (Praat-style) in a txt. The text of interest will be listed with timestamps of the occasions. The
Usage: edit.getTextGridInformation('Name', '.../StudyToolkitVid/examples/mediaEditing/lohse.TextGrid', '.../StudyToolkitVid/examples/mediaEditing/') -> extracts the text 'Name' in the current clip according to the lohse.TextGrid file (in the style of a txt, for example: Name 0.59 - 0.71 (next line) Name 5.41 - 5.94). in the medieEditing directory.

extractTextOccasionsFromGrid(text_of_interest, path_textgrid, path_save, extension = ''): extracts occasions of the text of interest in the current clip to a given directory as audio/video files. If the input is a video, the output will be video snippets with the text occasions (default mp4) and if the input is audio, the extracted occasions will be audio (default mp3). The extension can be changed but needs to fit the file format.
Usage: edit.extractTextOccasionsFromGrid('Name', '.../StudyToolkitVid/examples/mediaEditing/lohse.TextGrid', '.../StudyToolkitVid/examples/mediaEditing/', extension = 'mkv') -> extracts all 'Name' occurences in the current clip according to the given textgrid and saves them as mkv files.

getTextGrid(path_to_media, path_to_txt, path_to_save, name_of_file, language_code): downloads and saves a TextGrid of the media file on the computer (=> !Requires! internet connection and the server of University Mu-

nich must be online). An API-Call is made to use the "WebMausBasic" function of the University Munich: `https://clarin.phonetik.uni-muenchen.de/BASWebServices/interface/WebMAUSBasic`. The media file is not the currently edited file but a given file with a txt-file that contains exactly the spoken content (nothing more, nothing less). The WebMaus will then align the spoken text with the exact time of when the text has been spoken/when the phonemes have been spoken. The parameters path_to_save and name_of_file are for the downloaded TextGrid (do not add the .TextGrid extension to the parameter for the filename). A language_code is based on the spoken language. The supported codes are: [aus-AU, afr-ZA, sqi-AL, eus-ES, eus-FR, cat-ES, nld-BE, nld-NL, eng-AU, eng-US, eng-GB, eng-SC, eng-NZ, ekk-EE, fin-FI, fra-FR, kat-GE, deu-AT, deu-CH, deu-DE, gsw-CH, gsw-CH-BE, gsw-CH-BS, gsw-CH-GR, gsw-CH-SG, gsw-CH-ZH, hun-HU, isl-IS, ita-IT, jpn-JP, gup-AU, ltz-LU, mlt-MT, nor-NO, fas-IR, pol-PL, ron-RO, rus-RU, spa-ES, swe-SE, tha-TH, guf-AU]
Usage: edit.getTextGrid('.../StudyToolkitVid/examples/mediaEditing/lohse.mp4', '.../StudyToolkitVid/examples/mediaEditing/lohse.txt', './myTextGrids/', 'lohse', 'deu-DE') -> Calls the API for Maus-Basic, creates a TextGrid for the given video 'lohse.mp4', downloads it, and stores it in the directory myTextGrids with the name 'lohse.TextGrid'

1.6 Part 2 - Study Setup

1.6.1 Step 4 - Study Creation

1.6.2 Step 5 - Study Execution

1.6.3 Step 6 - Result Aggregation

1.7 Part 3 - Statistical Analysis

1.7.1 Step 7 - Data Cleaning

1.7.2 Step 8 - Data Analysis

1.7.3 Step 9 - Data Visualization