OpenHSV Documentation

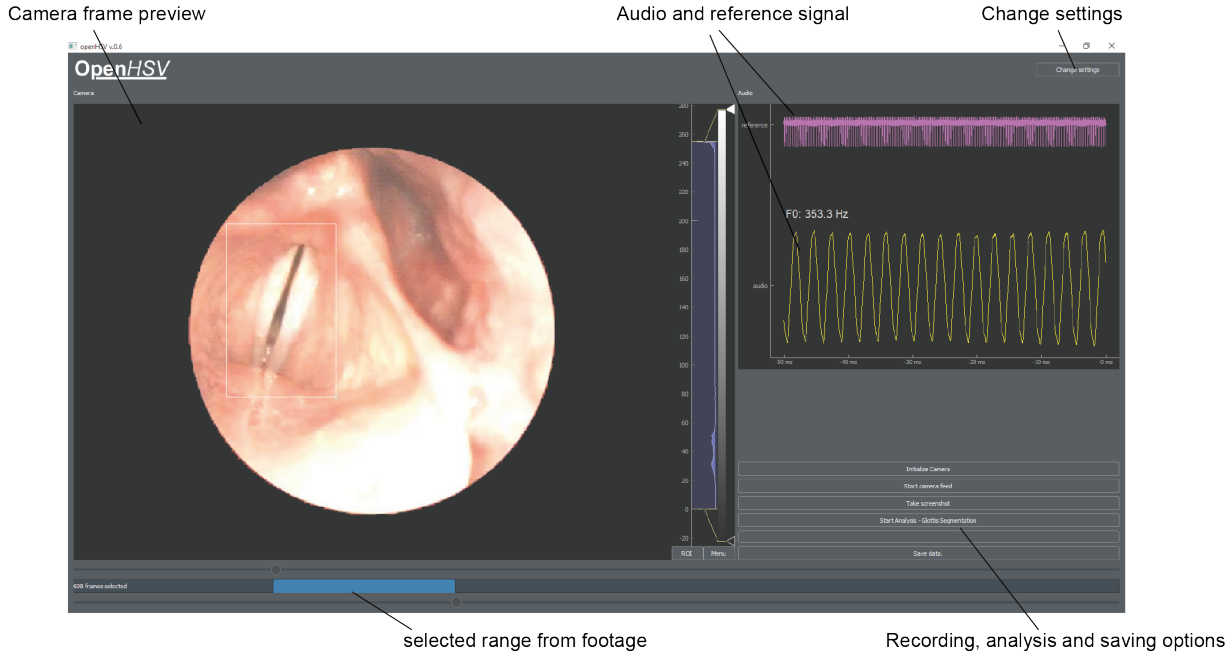
Release 0.5

Andreas M Kist

Dec 10, 2021

User Guide

OpenHSV is an open platform for laryngeal high-speed videoendoscopy. This documentation will help you to setup OpenHSV and to do data analysis.



Note: OpenHSV is only a **research** tool and is **not FDA approved**. You may consult your local ethics committee before using OpenHSV in your environment/clinic.

CHAPTER 1

Introduction

OpenHSV is an open source system for high-speed videoendoscopy (HSV). In contrast to videostroboscopy, with HSV we are able to see each vocal fold oscillation cycle with high temporal resolution. To not only visualize but also quantify the vocal fold oscillation behavior, OpenHSV contains also a data analysis part that segments the glottal area in each frame and computes quantitative parameters thereof. We further acquire simultaneously audio signals that are synchronized to the video signal. Quantitative parameters derived from the audio signal are also computed.

In this user guide, we provide the steps to built the system yourself.

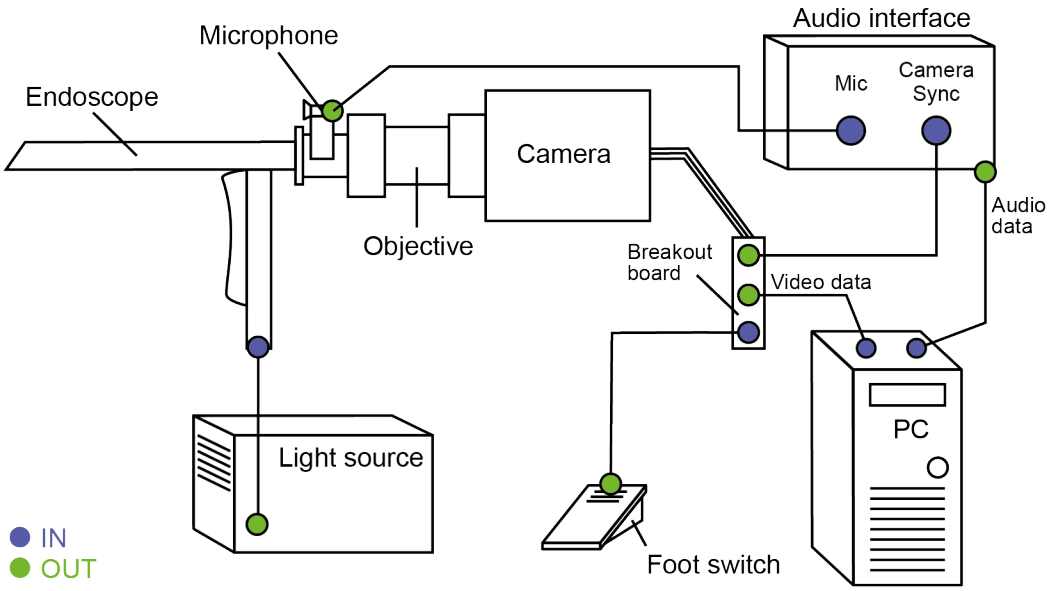
Steps:

1. Hardware setup
2. Software setup
3. First acquisition
4. Data analysis

CHAPTER 2

Hardware setup

The OpenHSV system consists of an **audio**, **video**, **illumination** and **acquisition** module.



2.1 Audio

The audio signal is acquired using a high-quality lavalier microphone, e.g. DPA 4060. This is connected to a conventional USB-powered two-channel audio interface. Two channels, because we need one channel for the microphone and one for the camera synch. The microphone is mounted on a custom 3D-printed clamp that is mounted on the endoscope.

2.2 Video

Follow these steps:

1. Mount objective/video coupler to camera
2. Connect video coupler to endoscope
3. Connect light guide to endoscope
4. Mount microphone mount on endoscope
5. Connect camera to breakout board
6. Connect microphone to audio interface
7. Connect camera synch out to audio interface (you may need a BNC -> TSSR adapter)
8. Connect camera to computer (patch cable)

You can start and stop the acquisition using the OpenHSV software; however, stopping the acquisition using a foot switch is better suited in an examination setting. For that, connect a foot switch to the TRIG IN port. Ensure that the foot switch has a BNC connector.

Note: Foot switch may come with loose ends. This may need some additional tinkering, such as soldering a BNC connector to the cable. As replacement, you may also use a screw-type BNC connector.

2.3 Illumination

For high-speed videoendoscopy, a powerful illumination unit, such as STORZ TL 300, is unevitable. Connect the power cable and the light guide accordingly. You may need 100% of light output, depending on your video coupler and recording sampling rate.

2.4 Acquisition

We use a conventional personal computer (PC) installed with Windows 10. Depending on your PC, you may add another Gigabit Ethernet adapter to have both, network access and camera data transfer. With IDT cameras, the Ethernet adapter needs to have a similar IP address. Check with the camera's IP address and change it accordingly for your case. E.g. the camera has 100.0.0.100, then you may use 100.0.0.99 on your PC.

CHAPTER 3

Software setup

We assume that Windows 10 is running on your system, and you have basic knowledge in Python.

3.1 Requirements

Ensure that the following software is installed on your computer:

- Python 3.x, tested with Python 3.6 and Anaconda package
- IDT SDK (to access drivers, get from distributor)

3.2 Supported cameras

OpenHSV currently provides only support for the CCM series of IDT high-speed cameras, but can be extended to your individual case.

3.3 Install and Operate OpenHSV

Clone our Github repository and install the `openhsv` package with `pip`:

```
pip install setup.py
```

Run OpenHSV by executing the `main.py` script.

```
python main.py
```

An easy way to create a shortcut to the OpenHSV software is to create a `bat` file with the following content:

```
# Activate anaconda3 environment
call "path/to/activate.bat" "path/to/anaconda3"
# Go to openhsv directory
pushd "path/to/openhsv"
# Execute main.py without showing console
python.exe -u "main.py"
```

You may then place a shortcut to that file on the Desktop to allow easy access for examiners.

A common problem is that the camera drivers are not found. Ensure that the driver files are either available in the system PATH or directly in the OpenHSV directory. You may need all DLLs from the IDT SDK.

3.4 Testing environment

If no camera is available or other parts of OpenHSV should be tested, we supply a dummy camera that loops through the example video shipped with OpenHSV. You only need to change in the `__init__.py` file the following line:

```
from openhsv.hardware.camera import IdtCamera as Camera
```

to

```
from openhsv.hardware.camera import DummyCamera as Camera
```

CHAPTER 4

Your fist acquisition

Follow these steps:

1. Connect everything according to *Hardware setup*.
2. Ensure all software is installed according to *Software setup*.
3. Turn on computer, illumination, and camera.
4. Open the OpenHSV software.
5. Click on “Initialize camera”. This opens a connection to the camera using the default settings.
6. Click on “Start Camera Feed”. Now, the camera should stream images as live preview and the audio signals should show the reference signal and an acoustic trace.
7. Click on “Stop Camera Feed” or use the optional foot switch to stop the recording.
8. Use one of the sliders below to browse through the acquired images.
9. Select a region of interest using the two range sliders. The bright blue area indicates the selected files.
10. You may analyze your data by clicking on “Start Analysis - Glottis Segmentation” after selecting the glottis using the rectangle in the preview window. See also *Data Analysis*.
11. Click on “Save data” to save the audio, video and meta data to the local file system. The frames are downloaded from the camera and stored in conventional formats.

4.1 Saving data

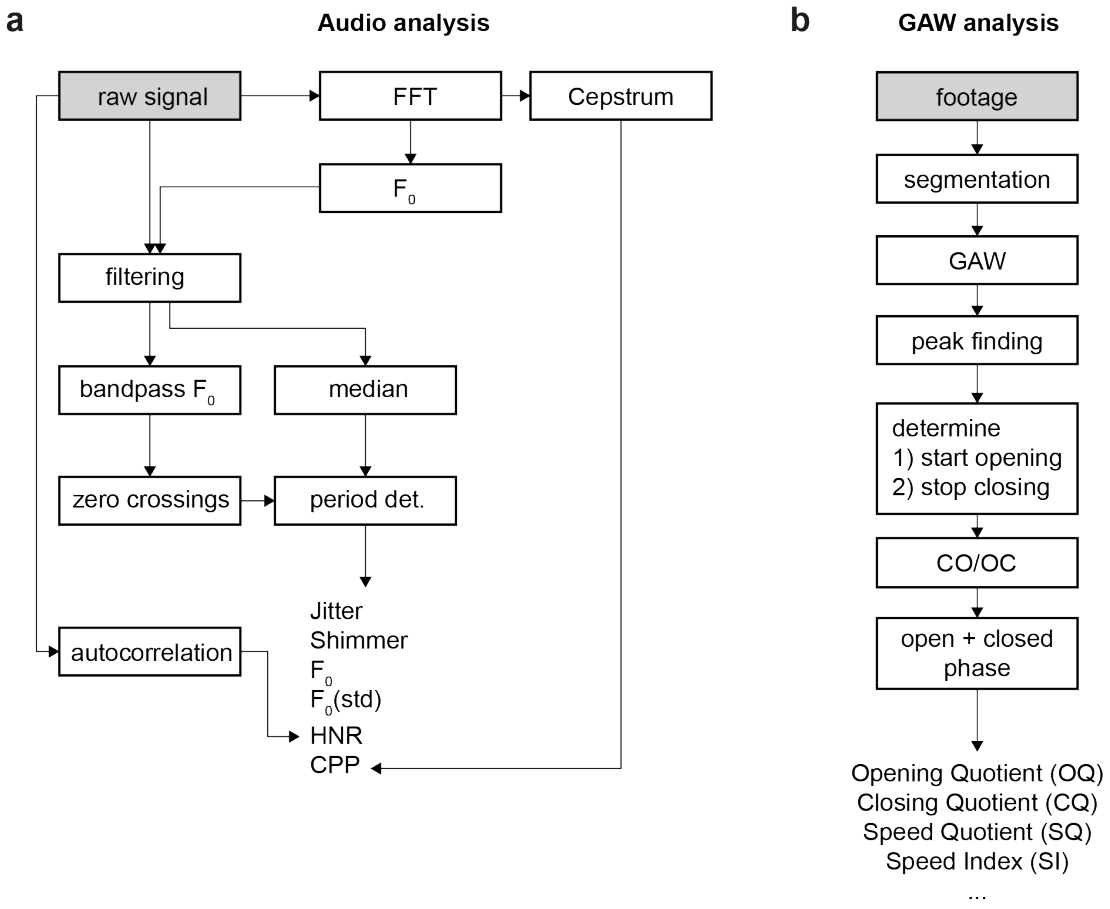
Data is stored in several ways:

Data source	File type
Video data	mp4 (compressed)
	mp4 (lossless, recommended for further analysis)
Audio data	wav (uncompressed)
Meta data	json (human-readable, uncompressed)
Analysis	hdf5 (if available)
Parameters	csv (if available)

CHAPTER 5

Data Analysis

Data analysis is performed for audio (a) and video (b) data and follows different paths:



5.1 Audio analysis

Audio data is cropped to the last 4 seconds (arbitrary, can be changed). The landmarks for acquisition and synchronization are detected using several peak finding algorithms (see code). The audio data that corresponds to the video data is subsequently analyzed (*raw signal* in above Figure, panel a).

Quantitative parameters are computed on the signal. Currently, these are:

- mean-jitter
- jitter-percent
- mean-shimmer
- shimmer-percent
- fundamental frequency (F0) and standard deviation (STD)
- Harmonics-to-Noise-Ratio (HNR)
- Cepstral Peak Prominence (CPP)

5.2 Video analysis

The glottal area is segmented using a deep convolutional neural network that was trained on the *BAGLS https://www.bagls.org/* dataset. Next, we use the segmented glottal area to

- compute the glottal area waveform (GAW)
- **estimate the glottal midline to**
  - compute GAW for left and right vocal fold
  - compute phonovibrogram (PVG)

With that, we follow the analysis pipeline as shown in the above Figure, panel b, to finally compute quantitative parameters. Currently, these are:

- Open Quotient (OQ)
- Closing Quotient (CQ)
- Asymmetry Quotient (AQ)
- Rate Quotient (RQ)
- Speed Quotient (SQ)
- Speed Index (SI)
- Fundamental frequency (F0)
- Amplitude perturbation factor (APF)
- Amplitude perturbation quotient (APQ)
- Glottis gap index (GGI)
- Amplitude Quotient (AQ)
- Stiffness
- Amplitude Symmetry Index (ASI)
- Phase Asymmetry Index (PAI)

5.3 Analysis

Parameters are shown in a separate window using a tabular view. All computed parameters can be exported as CSV file.

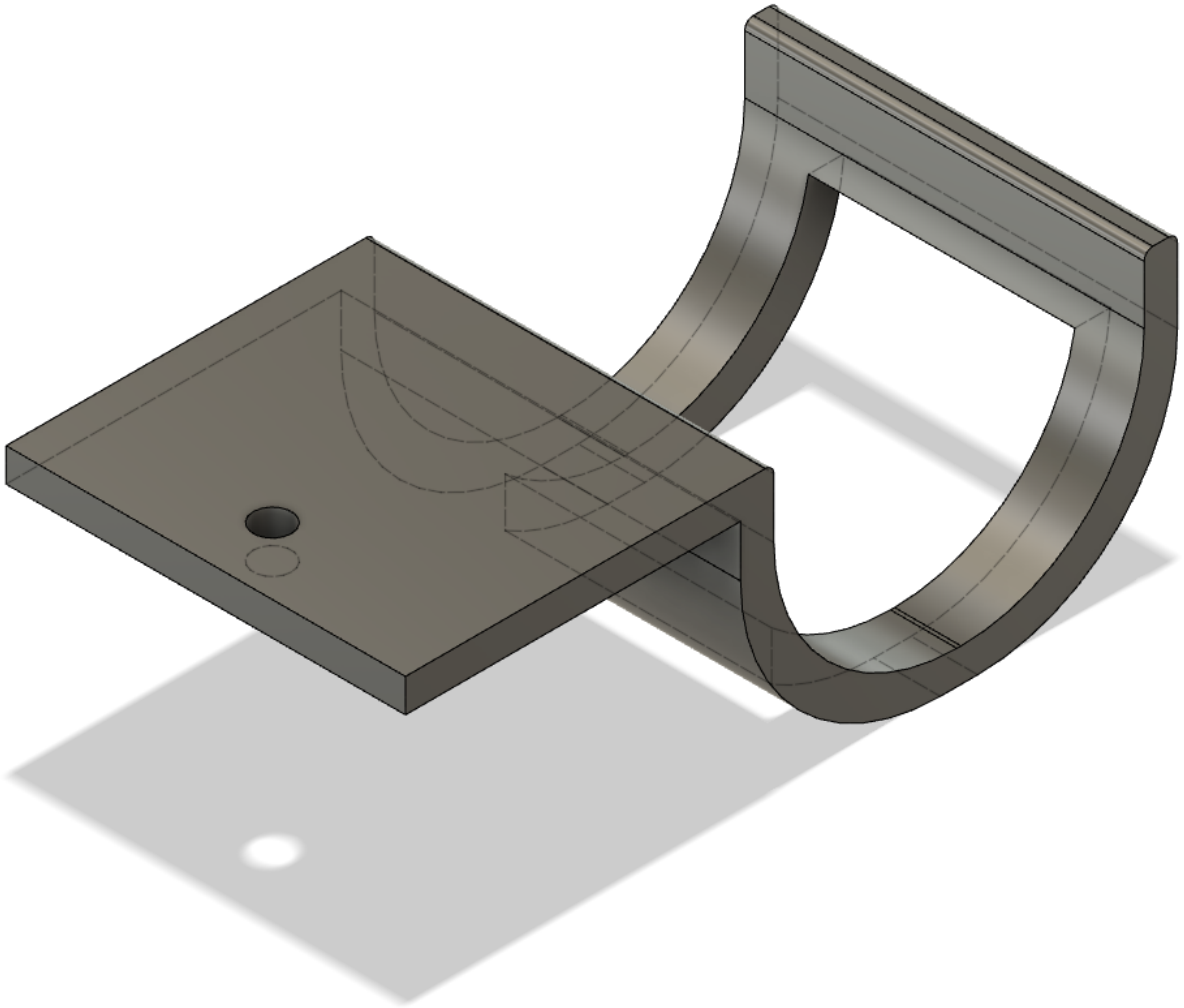
CHAPTER 6

3D printed parts

We 3D-printed the following parts to fit our needs. Everything was printed in Prusa PLA using the Prusa Mini FDM 3D-printer. We use typically an infill of 15% and 200 um layer size. All parts were designed in Fusion 360.

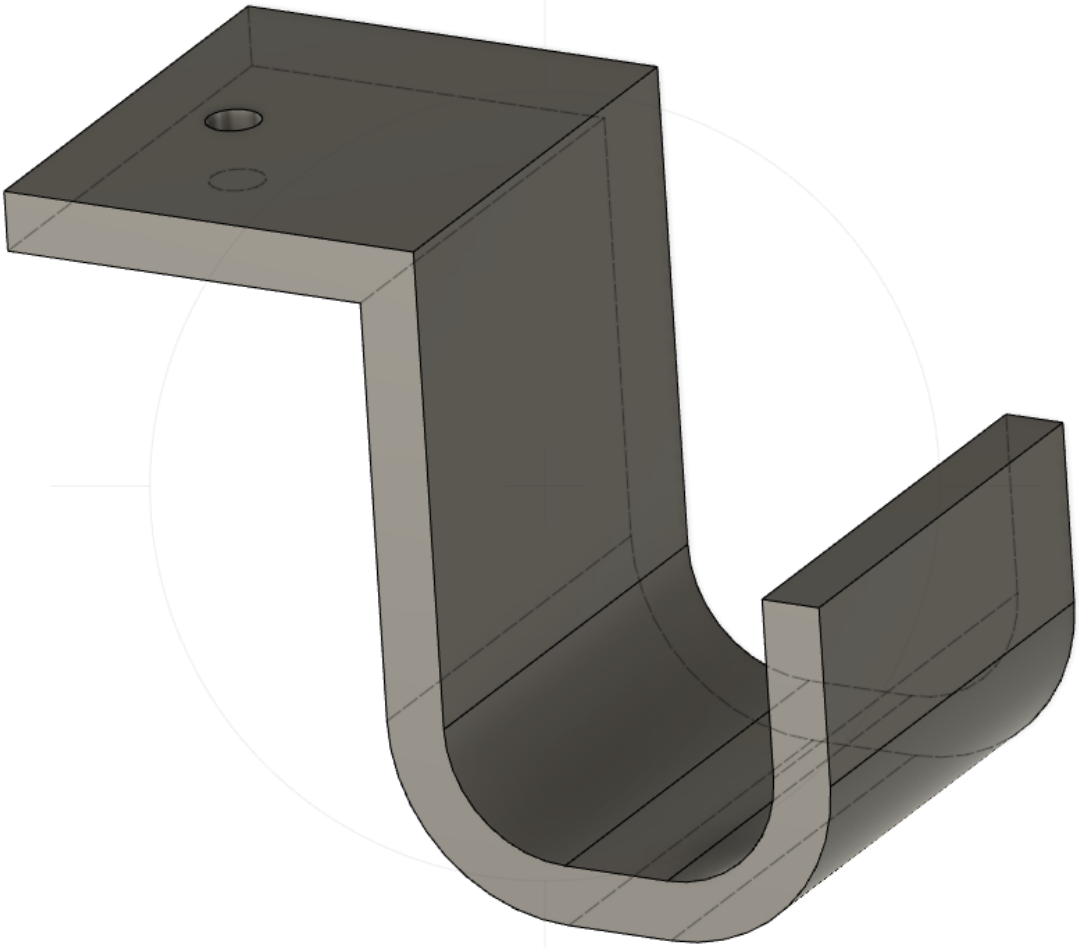
6.1 Endoscope holder

Endoscope holder to store the imaging unit.



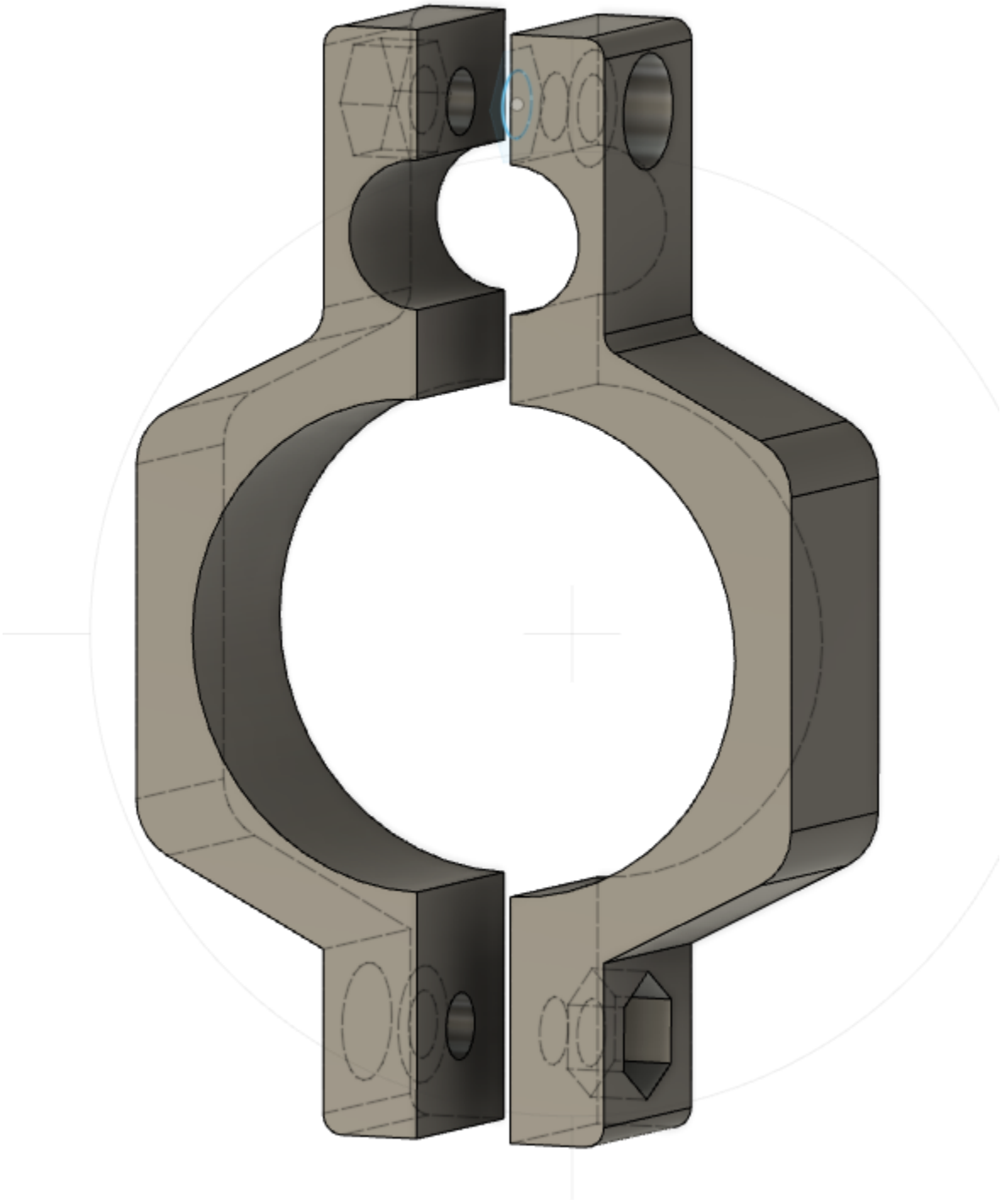
6.2 Cable holder

Cable holder for light guide and other cables.



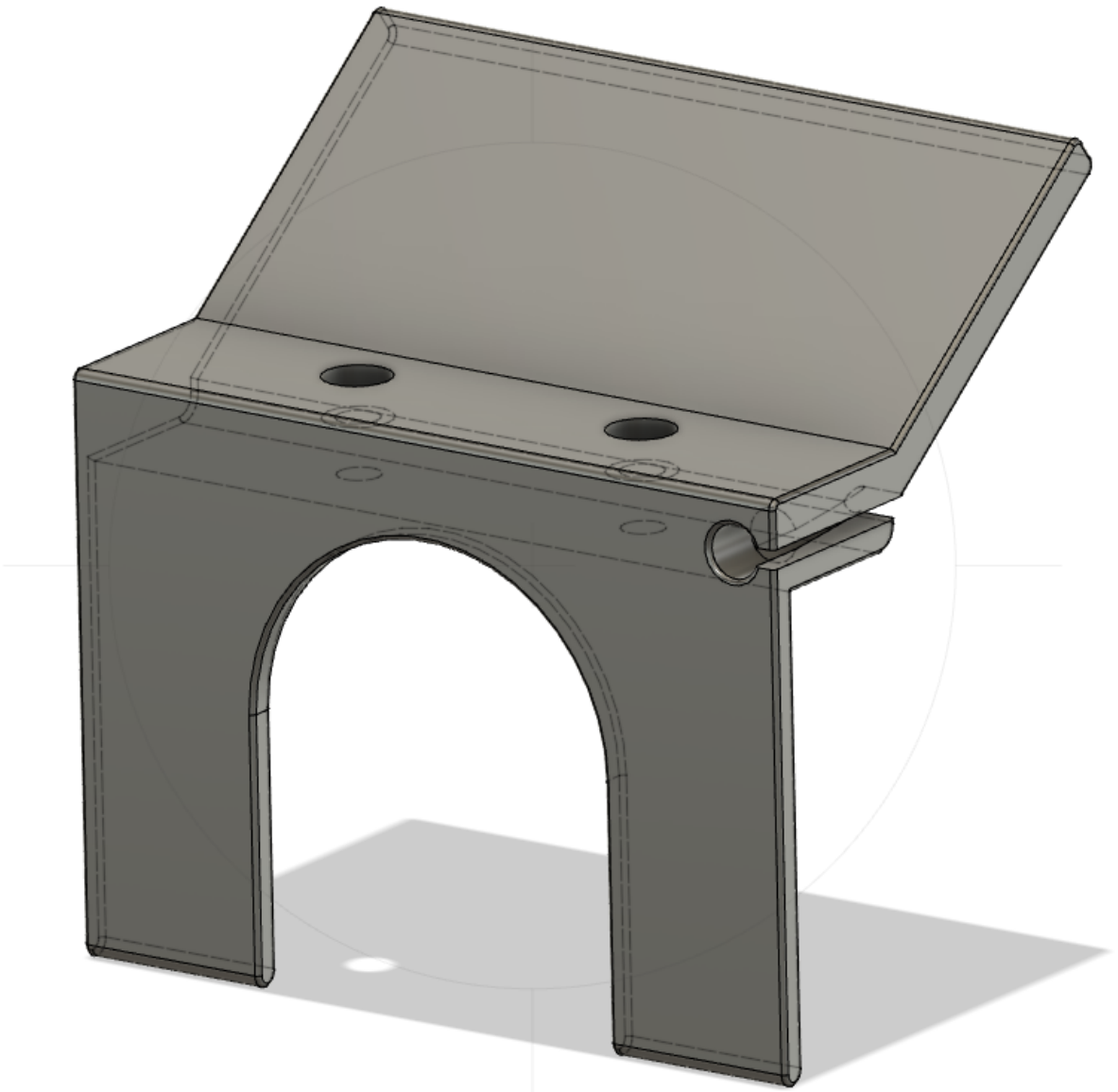
6.3 Microphone mount

The microphone mount consists of two parts: Part1 and Part2. You further need M3 screws and nuts to assemble it. Depending on your endoscope and the examiner this mount may cause some discomfort in handling the imaging unit.



6.4 Droplet protection shield

Also known as *Corona shield*. It is mounted with M4 screws on the IDT camera to protect the camera. On the side we provide an integrated microphone mount. The microphone cable enters on the side and is then pushed inwards.



CHAPTER 7

openhsv

7.1 openhsv package

7.1.1 Submodules

7.1.2 openhsv.main module

7.1.3 Module contents

class openhsv.**OpenHSV**(*app*, *base_folder='C:/openhsv'*, *verbose=False*)

Bases: PyQt5.QtWidgets.QWidget

OpenHSV is the main class for recording high-speed videoendoscopy footage and audio. It interacts with the audio interface and the camera, performs deep neural network based analysis and saves the data.

Parameters

- **app** (*QtWidgets.QApplication*) – To init OpenHSV, you only need to pass the QApplication instance
- **base_folder** (*str, optional*) – Location where data is stored
- **verbose** (*boolean, optional*) – Prints additional information to the Python console, defaults to False.

findpatient()

Opens a window to select patient from database.

showMaximized()

shows the window maximized and updates the range indicator

screenshot()

Takes a screenshot from the current camera image and saves it as png file.

settings()

Opens settings dialog and saves settings

initSettings(*exposureTime=245*, *videoSamplingRate=4000*, *audioSamplingRate=80000*, *audioBlockSize=4000*, *audioBufferSize=3*, *baseFolder=''*, *saveRaw=True*)

Initializes camera, audio and saving settings

Parameters

- **exposureTime** (*int, optional*) – camera exposure time in us, defaults to 245
- **videoSamplingRate** (*int, optional*) – frames per second, defaults to 4000
- **audioSamplingRate** (*int, optional*) – audio sampling rate in Hz, defaults to 80000
- **audioBlockSize** (*int, optional*) – audio block size transmitted from interface, defaults to 4000
- **audioBufferSize** (*int, optional*) – audio buffer size (multiples of block size), defaults to 3
- **baseFolder** (*str, optional*) – base folder for data saving, defaults to ''
- **saveRaw** (*bool, optional*) – if raw video data should be saved as lossless compressed mp4, defaults to True

patient()

Opens interface for patient information

updateRangeIndicator()

updates the range indicator that shows the subselection of the video for download or analysis

initCamera(*force_init=False*)

Initializes camera connection. Open camera, do basic configuration and set advanced settings, such as exposure time and video sampling rate (fps). If camera connection could be established, enable further buttons.

Parameters **force_init** (*bool, optional*) – forces (re-)initialization of camera, defaults to False

playStop()

setImage(*im*, *restore_view=True*, *restore_levels=False*)

Shows image in the camera preview window. It further can restore the previous view, i.e. zoom and location, as well as the levels (contrast, brightness) of the previous image. Currently, the restoring level feature is by default deactivated, because in the examination procedure it is quite common that there is no signal (away from patient) or oversaturated (very close to the mouth/tongue).

Parameters

- **im** (*numpy.ndarray*) – image to be shown in preview
- **restore_view** (*bool, optional*) – if view should be restored from previous image, defaults to True
- **restore_levels** (*bool, optional*) – if contrast/brightness should be restored from previous image, defaults to False

nextFrame()

initAudio()

initialize audio recorder and empties the audio queue and data list. It selects the first audio interface found (in OpenHSV the Focusrite Scarlet 2i2), selects both channels (by default, channel 1 is the camera reference signal and channel 2 the actual audio signal). Every audio data block is passed to the callback function. The callback works already on a separate thread, no need to move it to a different one. It also immmediately starts the recorder.

stopAudio()
Stops audio recording and saves data from queue to internal memory

F0(*channel_for_F0=1*, *intensity_threshold=5*)
Calculates fundamental frequency from audio signal. It further saves the audio data to internal memory.

Parameters

- **channel_for_F0** (*int, optional*) – selected audio channel for F0 calculation. In our setting, channel 0 is for the reference signal, channel 1 for the audio signal, defaults to 1
- **intensity_threshold** (*int, optional*) – intensity threshold for calculating F0, defaults to 5

startCamera()
Starts camera (and audio) feed. If grabbing is already active, it stops grabbing from the camera and stops streaming audio data. A full screen preview is shown to provide maximum view. It starts fundamental frequency calculation and saves audio data in memory.

analyze()
Analyzes the selected range of video data. The selected frames will be downloaded from the camera and subsequently processed, i.e. segmented by the neural network.

save(*save_last_seconds=4*)
Saves the recorded and selected data. In particular, we save the metadata, including audio, video and patient metadata, audio data together with camera reference signal and video data.

Parameters **save_last_seconds** (*int*) – the last seconds from recording end to be saved. We record one second after the *stop*-trigger, and we usually record one second of footage, thus, we need at least two seconds to ensure saving all relevant audio data. To adjust for some uncertainties, we recommend recording a few more seconds. Defaults to 4.

Note: Saving the data in an appropriate format is not trivial. We both need to consider portability, cross-functionality and quality. Therefore, we save metadata as structured JSON file format, a common file format that can be opened and viewed with any text editor, but easily processed by a variety of data analysis software.

Further, audio data is saved as common wav files, as well as packed as HDF5 file. HDF5 is a very common container format that allows storing of complex data in a very efficient and convenient way.

Video data, however, is saved as `mp4` file format, as this is highly portable and can be viewed with a common video viewers. The h264 codec also allows saving the video data in a lossless file format, needed for accurate data analysis while keeping the file size at a reasonable level and still ensure the ability to preview the video.

If there's any segmentation already available, the segmentation maps are stored as well in HDF5 file format as binary maps.

---

close(*self*) → bool

CHAPTER 8

openhsv.analysis package

8.1 Module contents

8.2 Submodules

8.3 openhsv.analysis.nn module

class openhsv.analysis.nn.**Analysis**(*app=None*)
Bases: PyQt5.QtWidgets.QWidget

Analysis widget that shows the segmentation process of the neural network.

Parameters

- **QWidget** (*PyQt5.QtWidgets.QWidget*) – Inherits from QWidget
- **app** (*PyQt5.QtWidgets.QWidget, optional*) – QApplication, needed to process events to avoid freezing of the GUI, defaults to None

segmentSequence(*ims*, *normalize=True*, *reinit=True*)
segments an image sequence, such as a video, frame by frame.

Parameters

- **ims** (*list of numpy.ndarray, or numpy.ndarray*) – collection of images
- **normalize** (*bool, optional*) – normalize 0..255 to -1..1, defaults to True
- **reinit** (*bool, optional*) – deletes any previous segmentation information, defaults to True

segment(*im*)
Segments an endoscopic image using a deep neural network

Parameters **im** – np.ndarray (HxWx3)

Returns

setAudio(*audio*)

syncAudio(*start_frame*, *end_frame*, *total_frames*, *debug=False*)

computeParameters(*dt_audio=1.25e-05*, *dt_video=0.00025*, *debug=False*)
Compute parameters from GAW

Parameters

- **dt_audio** (*float, optional*) – audio sampling time in seconds, defaults to 1/80000
- **dt_video** (*float, optional*) – video sampling time in seconds, defaults to 1/4000
- **debug** (*bool, optional*) – shows debugging information and plots, defaults to False

get()
returns GAW and segmentation maps for video

Returns GAW and segmentations

Return type tuple(list, list(numpy.ndarray))

class openhsv.analysis.nn.**ROIDialog**(*ims*)
Bases: PyQt5.QtWidgets.QDialog

8.4 openhsv.analysis.parameters module

openhsv.analysis.parameters.**movingAverage**
computes moving average from x with a window of n

Parameters

- **x** (*numpy.ndarray*) – input array
- **n** (*int, optional*) – window, defaults to 3

Returns filtered array

Return type numpy.ndarray

openhsv.analysis.parameters.**detectMaximaMinima**(*s*, *distance=5*, *rel_height=0.01*, *use_prominence=True*, *clean_f0=None*)
Detect maxima and minima from a signal s.

Parameters

- **s** (*numpy.ndarray*) – signal
- **distance** (*int, optional*) – distance between two peaks in samples, defaults to 5
- **rel_height** (*float, optional*) – minimum relative height of a found peak, defaults to .35
- **use_prominence** (*bool, optional*) – uses peak prominence for peak detection, defaults to True

Returns tuple of maxima and minima locations

Return type tuple(numpy.ndarray, numpy.ndarray)

openhsv.analysis.parameters.**detectOpeningAndClosingEvents**(*signal*, *p_max*, *t=0.02*)
Detects glottis opening and closing events relative to maximum opening events.

Parameters

- **signal** (*numpy.ndarray*) – glottal area waveform
- **p_max** (*list of indexes*) – maxima of glottal area waveform
- **t** (*float, optional*) – threshold for finding signal bottom, defaults to 0.02

Returns opening and closing point for each maximum

Return type tuple(list(int), list(int))

openhsv.analysis.parameters.**computeOpenAndClosedIntervals**(*t*, *opening*, *closed*)
computes the opened and closed intervals during each cycle.

Parameters

- **t** (*numpy.ndarray*) – time
- **opening** (*list(int)*) – indices of opening points
- **closed** (*list(int)*) – indices of closing points

Returns duration of opening and closed phases

Return type tuple(list(float), list(float))

openhsv.analysis.parameters.**computeOCandCOTransitions**(*t*, *opening*, *closed*, *p_max*)
Computes Open->Closed (OC) and Closed-Open (CO) transitions.

Parameters

- **t** (*numpy.ndarray*) – time
- **opening** (*list(int)*) – indices of opening points
- **closed** (*list(int)*) – indices of closing points
- **p_max** (*numpy.ndarray*) – indices of cycle maxima

Returns CO and OC durations

Return type tuple(numpy.ndarray, numpy.ndarray)

openhsv.analysis.parameters.**F0fromCycles**(*T*, *verbose=False*, *epsilon=1e-09*)
determine fundamental frequency (F0) based on period lengths

Parameters

- **T** (*numpy.ndarray*) – periods
- **verbose** (*bool, optional*) – prints F0 mean and standard deviation, defaults to False

Returns mean of F0 and std of F0

Return type tuple(float, float)

openhsv.analysis.parameters.**F0fromFFT**(*fft*, *freqs*, *freq_lower=75*, *freq_higher=500*)
fundamental frequency from power spectrum determined assuming that fundamental frequency is equal to the dominant frequency. Frequency is determined in physiological range (75-500 Hz).

Parameters

- **fft** (*numpy.ndarray*) – fast fourier transformation of signal
- **freqs** (*numpy.ndarray*) – frequencies corresponding to FFT
- **freq_lower** (*int, optional*) – lower border for F0 prediction, defaults to 75
- **freq_higher** (*int, optional*) – higher border for F0 prediction, defaults to 500

Returns fundamental frequency (F0)

Return type float

openhsv.analysis.parameters.**F0fromAutocorrelation**(*signal*, *freq=40000*)
Fundamental frequency using autocorrelation.

Steps:

1) rFFT of signal
2) Corresponding frequencies
3) Autocorrelation
4) Find peaks in autocorrelation
5) Remove first unrelevant peaks
6) Use first peak as fundamental frequency

Parameters

- **signal** (*numpy.ndarray*) – Signal (e.g. audio trace or GAW)
- **freq** (*int*) – sampling frequency, defaults to 40000 Hz

Returns fundamental frequency (F0)

Return type float

openhsv.analysis.parameters.**asymmetryQuotient**(*CO*, *OC*)
Asymmetry Quotient (AsyQ)

$$\text{AsyQ} = \frac{1}{N}\sum_{i=1}^{N}\frac{\frac{CO_i}{OC_i}}{1+\frac{CO_i}{OC_i}}$$

Parameters

- **CO** (*numpy.ndarray*) – Closed->Open transitions
- **OC** (*numpy.ndarray*) – Open->Closed transitions

Returns asymmetry quotient (AQ), a.u.

Return type float

openhsv.analysis.parameters.**closingQuotient**(*CO*, *t_open*)
Closing Quotient (CQ)

$$\text{CQ} = \frac{1}{N}\sum_{i=1}^{N}\frac{CO_i}{t_{open,i}}$$

Parameters

- **CO** (*numpy.ndarray*) – Closed->Open transitions
- **t_open** – Open interval

Returns closing quotient (CQ), a.u.

Return type float

openhsv.analysis.parameters.**openQuotient**(*t_open*, *t_closed*)
Open Quotient (OQ)

$$\mathrm{OQ} = \frac{1}{N}\sum_{i=1}^{N}\frac{t_{open,i}}{t_{open,i}+t_{closed,i}}$$

Parameters

- **t_open** (*numpy.ndarray*) – Open intervals
- **t_closed** (*numpy.ndarray*) – Closed intervals

Returns open quotient (OQ), a.u.

Return type float

openhsv.analysis.parameters.**rateQuotient**(*CO*, *OC*, *t_closed*)
Rate Quotient (RQ)

$$\mathrm{RQ} = \frac{1}{N}\sum_{i=1}^{N}\frac{t_{closed,i}-CO_i}{OC_i}$$

Parameters

- **CO** (*numpy.ndarray*) – Closed->Open transitions
- **OC** (*numpy.ndarray*) – Open->Closed transitions
- **t_closed** (*numpy.ndarray*) – closed intervals

Returns rate quotient (RQ), a.u.

Return type float

openhsv.analysis.parameters.**speedIndex**(*CO*, *OC*, *t_open*)
Speed Index (SI)

$$\mathrm{SI} = \frac{1}{N}\sum_{i=1}^{N}\frac{CO_i-OC_i}{t_{open,i}}$$

Parameters

- **CO** (*numpy.ndarray*) – Closed->Open transitions
- **OC** (*numpy.ndarray*) – Open->Closed transitions
- **t_open** (*numpy.ndarray*) – open intervals

Returns speed index (SI), a.u.

Return type float

openhsv.analysis.parameters.**speedQuotient**(*CO*, *OC*)
Speed Quotient (SQ)

$$\mathrm{SQ} = \frac{1}{N}\sum_{i=1}^{N}\frac{CO_i}{OC_i}$$

Parameters

- **CO** (*numpy.ndarray*) – Closed->Open transitions
- **OC** (*numpy.ndarray*) – Open->Closed transitions

Returns speed quotient (SQ), a.u.

Return type float

openhsv.analysis.parameters.**meanJitter**(*T*)

Calculating the mean jitter in ms from signal periods

$$\text{mean-Jitter} = \frac{\sum_{i=1}^{N-1} |T_i - T_{i-1}|}{N-1}$$

Parameters **T** (*numpy.ndarray or list*) – The signal periods

Returns mean jitter in ms

Return type float

openhsv.analysis.parameters.**jitterPercent**(*T*)

Calculating the jitter in percent from signal periods.

Returns jitter in percent

Return type float

openhsv.analysis.parameters.**meanShimmer**(*A*, *epsilon=1e-09*)

Calculating the mean shimmer in dB from the signal amplitude maxima.

$$\text{mean-Shimmer [db]} = \frac{20}{N-1} \sum_{i=0}^{N-2} |\log_{10}\left[\frac{A_i}{A_{i+1}}\right]|.$$

Parameters **A** (*numpy.ndarray or list*) – The signal amplitude maxima

Returns mean shimmer in db

Return type float

openhsv.analysis.parameters.**shimmerPercent**(*A*, *e=1e-05*)

Calculating the shimmer in percent from the signal amplitude maxima.

Returns shimmer in percent

Return type float

openhsv.analysis.parameters.**periodPerturbationFactor**(*T*)

Calculating the Period Perturbation Factor (PPF) in arbitrary units using the signal periods.

$$\text{PPF} = \frac{1}{N-1} \sum_{i=1}^{N-1} |\frac{T_i - T_{i-1}}{T_i}| \cdot 100$$

Parameters **T** (*list*) – periods
Returns PPF in percent
Return type float

openhsv.analysis.parameters.**glottalGapIndex**(*signal*, *opening*, *epsilon=1e-09*)

Glottal Gap Index (GGI) that computes the relation between minimum and maximum glottal area in each glottal cycle.

$$GGI = \frac{1}{N} \sum_{i}^{N} \frac{\min(a_i)}{\max(a_i)}$$

Parameters
- **signal** (*numpy.ndarray*) – glottal area waveform
- **opening** (*list*) – points where the glottis opens
- **epsilon** (*float, optional*) – numerical stability, defaults to 1e-9

Returns glottal gap index mean and std
Return type tuple(float, float)

openhsv.analysis.parameters.**amplitudePerturbationFactor**(*A*)
Amplitude perturbation factor.

$$APF = \frac{1}{N}\sum_{i=1}^{N-1}|\frac{A_i - A_{i-1}}{A_i}| \cdot 100$$

Parameters **A** (*list or numpy.ndarray*) – amplitudes in each cycle
Returns APF mean and std
Return type tuple(float, float)

openhsv.analysis.parameters.**amplitudePerturbationQuotient**(*A*, *k=3*)
Amplitude perturbation quotient

$$l = \frac{k-1}{2}$$

$$APQ_k = \frac{1}{N-k}\sum_{i=l}^{N-l-1}|1 - \frac{k \cdot A_i}{\sum_{j=-l}^{l} A_{i+j}}| \cdot 100$$

Parameters
- **A** (*list or numpy.ndarray*) – amplitudes in each cycle
- **k** (*int, optional*) – range, defaults to 3

Returns APQ mean and std
Return type tuple(float, float)

openhsv.analysis.parameters.**amplitudeQuotient**(*signal*, *opening*)
Amplitude Quotient

$$AQ = \frac{A_i}{|\min_j \frac{d}{dj} f_i(j)|}$$

Parameters
- **signal** (*numpy.ndarray*) – audio or GAW signal
- **opening** (*list*) – indices where glottis starts to open

Returns AQ mean and std
Return type tuple(float, float)

openhsv.analysis.parameters.**stiffness**(*signal*, *opening*)
Stiffness

$$Stiffness = \frac{|\max_j |\frac{d}{dj} f_i(j)||}{A_i}$$

Parameters
- **signal** (*numpy.ndarray*) – audio or GAW signal
- **opening** (*list*) – indices where glottis starts to open

Returns stiffness mean and std
Return type tuple(float, float)

openhsv.analysis.parameters.**harmonicNoiseRatio**(*signal*, *freq*, *freq_lower=50*, *freq_higher=450*, *filter_autocorrelation=False*, *epsilon=1e-09*)

Computes Harmonic-Noise-Ratio (HNR) using autocorrelation approximation. First, it computes the fundamental frequency using the power spectrum of `signal`. Next, it computes the autocorrelation in Fourier space. Then, local maxima in the autocorrelation are found, the HNR computed and the maximum HNR and the corresponding frequency is returned.

$$R_{xx} = \frac{1}{N} \sum_{k=l}^{N-1} x[k]x[k-l]$$

$$HNR = \frac{R_{xx}[T_0]}{R_{xx}[0] - R_{xx}[T_0]}$$

Parameters

- **signal** (*numpy.ndarray*) – audio signal
- **freq** (*int*) – sampling rate/frequency, e.g. 44100
- **freq_lower** (*int, optional*) – lower frequency cut-off, defaults to 50
- **freq_higher** (*int, optional*) – higher frequency cut-off, defaults to 350

Returns HNR [dB], F0_FFT [Hz], F0_Autocorr [Hz]

Return type tuple(float, float, float)

openhsv.analysis.parameters.**cepstralPeakProminence**(*signal*, *freq*, *freq_lower=70*, *freq_higher=350*, *plot=False*)

Computes cepstral peak prominence from signal using Fourier transformations.

Steps:

1) Compute FFT from signal
2) Compute fundamental frequency from power spectrum
3) Compute cepstrum from FFT, filter with moving average (window = 3)
4) Find maximum peak in cepstrum
5) Find corresponding quefrency
6) Fit line to cepstrum
7) Compute distance from peak to line –> Cepstral Peak Prominence

Parameters

- **signal** (*numpy.ndarray*) – audio signal
- **freq** (*int*) – sampling rate/frequency, e.g. 44100
- **freq_lower** (*int, optional*) – lower frequency cut-off, defaults to 70
- **freq_higher** (*int, optional*) – higher frequency cut-off, defaults to 350
- **plot** (*bool, optional*) – plots the cepstrum, the line and the peak prominence, defaults to False

Returns CPP [dB], F0_FFT [Hz], F0_Cepstrum [Hz]

Return type tuple(float, float, float)

openhsv.analysis.parameters.**phaseAsymmetryIndex**(*left_gaw*, *right_gaw*, *opening*)

Phase Asymmetry Index (PAI)

$$PAI = \frac{|argmax_j GA_i^L(j) - argmax_j GA_i^R(j)|}{N_i}$$

Note: We use here the maximum instead of the minimum, as it is more likely that there are multiple time points where the value is minimal or close to the minimum (i.e. when the glottis is closed for prolonged time).

Parameters

- **left_gaw** (*numpy.ndarray*) – GAW of left vocal fold
- **right_gaw** (*numpy.ndarray*) – GAW from right vocal fold
- **opening** (*list*) – indices where glottis starts to open (i.e. new cycle)

Returns PAI mean and std
Return type tuple(float, float)

openhsv.analysis.parameters.**amplitudeSymmetryIndex**(*left_gaw*, *right_gaw*, *opening*, *epsilon=1e-05*)

Amplitude Symmetry Index (ASI)

$$ASI = \frac{\min(\max_j(GA_i^L(j)), \max_j(GA_i^R(j)))}{\max(\max_j(GA_i^L(j)), \max_j(GA_i^R(j)))}$$

Parameters

- **left_gaw** (*numpy.ndarray*) – GAW of left vocal fold
- **right_gaw** (*numpy.ndarray*) – GAW from right vocal fold
- **opening** (*list*) – indices where glottis starts to open (i.e. new cycle)
- **epsilon** (*float, optional*) – numerical stability, defaults to 1e-5

Returns ASI mean and std
Return type tuple(float, float)

class openhsv.analysis.parameters.**Signal**(*raw_signal*, *dt=0.00025*, *debug=True*)

Bases: object

Inits the signal class with the raw signal, e.g. audio data or glottal area waveform.

Parameters **raw_signal** (*numpy.ndarray*) – Audio signal, GAW or alike

computeFFT(*use_filtered_signal=True*, *use_hanning=True*, *lowpass_filter=20*)

computeCepstrum()

filterSignal(*cutoff_frequency=0.1*)

detectCycles(*method='peaks'*, *peak='max'*, *use_filtered_signal=True*)

Detects cycles using different methods.

Parameters

- **method** (*str, optional*) – method to detect cycles, defaults to 'peaks'
  - ***peaks*** Using a peak finding algorithm on the raw signal
  - ***autocorrelation*** Detects cycles and period using autocorrelation
- **use_filtered_signal** (*bool, optional*) – uses filtered signal, if available.

detectPhases(*use_filtered_signal=True*)

Detects opening and close phase in each cycle.

Parameters **use_filtered_signal** (*bool, optional*) – Event detection on raw (False) or filtered (True) signal.

getPowerSpectrum()

Returns power spectrum from signal

Returns Frequencies and Amplitude
Return type tuple(np.ndarray, np.ndarray)

getCepsturm()

Returns cepstrum from signal

Returns quefrencies and cepstrum
Return type tuple(np.ndarray, np.ndarray)

class openhsv.analysis.parameters.**Audio**(*raw_signal*, *dt=1.25e-05*, *use_filtered_signal=True*, *use_hanning=True*, *debug=False*)

Bases: *openhsv.analysis.parameters.Signal*

The **Audio** class handles audio data to compute respective parameters.

Parameters

- **raw_signal** (*numpy.ndarray*) – the raw audio signal
- **dt** (*float, optional*) – time between samples in seconds, defaults to 1/80000
- **use_filtered_signal** (*bool, optional*) – use filtered signal for computations, defaults to True
- **use_hanning** (*bool, optional*) – use hanning window for FFT, defaults to True
- **debug** (*bool, optional*) – enable debugging mode, defaults to False

detectCycles(*method='peaks'*, *peak='max'*, *use_filtered_signal=True*)

Detects cycles using different methods.

Parameters

- **method** (*str, optional*) – method to detect cycles, defaults to 'peaks'
  - ***peaks*** Using a peak finding algorithm on the raw signal
  - ***autocorrelation*** Detects cycles and period using autocorrelation
- **use_filtered_signal** (*bool, optional*) – uses filtered signal, if available.

filterSignal(*freq_range=20*)

computeParameters(*use_filtered_signal=False*)

class openhsv.analysis.parameters.**GAW**(*raw_signal*, *dt=0.00025*, *use_filtered_signal=False*, *use_hanning=True*, *debug=False*)

Bases: *openhsv.analysis.parameters.Signal*

The **GAW** class handles the glottal area waveform (GAW) to compute respective parameters.

Parameters

- **raw_signal** (*numpy.ndarray*) – raw GAW signal
- **dt** (*float, optional*) – time between samples in seconds, defaults to 1/4000
- **use_filtered_signal** (*bool, optional*) – use filtered signal for computations, defaults to False
- **use_hanning** (*bool, optional*) – use hanning window for FFT, defaults to True
- **debug** (*bool, optional*) – enable debugging mode, defaults to False

setLeftRightGAW(*left_gaw*, *right_gaw*)

computeParameters()

class openhsv.analysis.parameters.**AnalysisPlatform**(*raw_signal*, *dt*)

Bases: PyQt5.QtWidgets.QWidget

8.5 openhsv.analysis.midline module

class openhsv.analysis.midline.**Midline**(*seg*, *maxima=None*)

Bases: object

Midline prediction module.

Midline is predicted based on segmentation from neural net for each peak. Midline is interpolated between peaks. Computations are based on Kist et al., biorxiv 2020.

Parameters

- **seg** (*numpy.ndarray*) – segmentation masks (T x H x W)
- **maxima** (*list, optional*) – detected maxima in GAW, defaults to None

predict(*method='pca'*, *time_range=5*)

Predicts midline with given method on each GAW peak.

Parameters

- **method** (*str, optional*) – 'pca' or 'moments', defaults to 'pca'
- **time_range** (*int, optional*) – time range around peak to improve prediction, defaults to 5

side()

Returns left and right GAW based on midline in each frame

Returns left and right GAW as array T x 2

Return type numpy.ndarray

pvg(*steps=64*)

Computes PVG in discrete steps for each side.

Parameters **steps** (*int, optional*) – resolution along each axis, defaults to 64

Returns phonovibrogram as T x steps*2

Return type numpy.ndarray

openhsv.analysis.midline.**imageMoments**(*im*, *transpose=True*, *angle_correction=1.5707963267948966*)

Predicts midline using image moments.

Parameters

- **im** (*[type]*) – [description]
- **transpose** (*bool, optional*) – [description], defaults to True
- **angle_correction** (*[type], optional*) – [description], defaults to +np.pi/2

Returns [description]

Return type [type]

openhsv.analysis.midline.**principalComponents**(*im*, *use_2nd=False*)

Midline prediction using principal component analysis.

Parameters

- **im** (*numpy.ndarray*) – input image
- **use_2nd** (*bool, optional*) – use second principal component, defaults to False

Returns slope and intercept of midline

Return type tuple(float, float)

8.6 openhsv.analysis.pvg module

openhsv.analysis.pvg.**get_labels**(*x_low*, *x_high*, *coef*, *intercept*, *image_shape*, *steps=64*)

Function for getting left/right step-id'd labels based on the AP axis

Parameters

- **x_low** (*float*) – lower x coordinate
- **x_high** (*float*) – higher x coordinate
- **coef** (*float*) – coefficient of linear regression AP
- **intercept** (*float*) – intercept of linear regression AP
- **image_shape** (*tuple(int, int)*) – image size (HxW) for label
- **steps** (*int*) – steps between A and P

Returns label map

Return type numpy.ndarray

openhsv.analysis.pvg.**compute_pvg**(*s*, *labels*, *steps=64*)

Calculates Phonovibrogram based on labels.

Parameters

- **s** (*numpy.ndarray*) – segmented area (TxYxX)
- **labels** (*numpy.ndarray*) – labelled image
- **steps** (*int, optional*) – PVG resolution, defaults to 64

Returns PVG, time x 2*steps

Return type numpy.ndarray

CHAPTER 9

openhsv.gui package

9.1 Submodules

9.2 openhsv.gui.misc module

class openhsv.gui.misc.**fullScreenPreview**
Bases: PyQt5.QtWidgets.QWidget

Full Screen Preview widget

setImage(*im*)
Sets image in central ImageView
Parameters **im** (*numpy.ndarray*) – image to be shown

9.3 openhsv.gui.patient module

class openhsv.gui.patient.**Patient**(*base_folder*)
Bases: PyQt5.QtWidgets.QDialog

Patient dialog
Parameters **base_folder** (*str*) – base folder where data will be saved

close(*self*) → bool

get()

9.4 openhsv.gui.settings module

class openhsv.gui.settings.**Settings**(*exposure*, *fps*, *audioSamplingRate*, *audioBlockSize*, *audioBufferSize*, *save_raw*, *base_folder*)
Bases: PyQt5.QtWidgets.QDialog

Define settings for OpenHSV operation, especially camera, audio and save settings.

selectBaseFolder()

get()

saveAndClose()

9.5 Module contents

CHAPTER 10

openhsv.hardware package

10.1 Submodules

10.2 openhsv.hardware.camera module

class openhsv.hardware.camera.**Camera**(*verbose=True*)

Bases: abc.ABC

An abstract camera class

Parameters

- **ABC** (*object*) – abstract class
- **verbose** (*bool, optional*) – prints to console, defaults to True

openCamera()

opens camera

configCam(**args*, ***kwargs*)

configures camera

setSettings(*exposure*, *fps*, **args*, ***kwargs*)

sets camera settings

Parameters

- **exposure** (*int*) – exposure time
- **fps** (*int*) – frames per second

isIdle()

returns if camera is recording or not (boolean)

startGrab()

Starts acquisition on camera

stopGrab()

stops acquisition on camera

live()
returns live image preview as numpy array

updateTriggerPosition()
Updates the trigger position for internal memory view

getMemoryFrame(*frame_index*, *by_trigger*)
gets memory frame from camera onboard memory
Parameters **frame_index** (*int*) – frame index in memory

closeCamera()
closes camera connection

class openhsv.hardware.camera.**DummyCamera**(*is_color=True*, *verbose=True*)
Bases: *openhsv.hardware.camera.Camera*

openCamera()
opens camera

configCam()
configures camera

setSettings(**args*, ***kwargs*)
sets camera settings
Parameters
- **exposure** (*int*) – exposure time
- **fps** (*int*) – frames per second

isIdle()
returns if camera is recording or not (boolean)

startGrab()
Starts acquisition on camera

stopGrab()
stops acquisition on camera

live()
returns live image preview as numpy array

getMemoryFrame(*frame_index*, *by_trigger=True*)
gets memory frame from camera onboard memory
Parameters **frame_index** (*int*) – frame index in memory

closeCamera()
closes camera connection

updateTriggerPosition()
Updates the trigger position for internal memory view

class openhsv.hardware.camera.**IdtCamera**(*verbose=True*)
Bases: *openhsv.hardware.camera.Camera*

The IdtCamera class uses the abstract `Camera` class to interact with the IDT high-speed camera API. In particular it starts and stops recording, fetches frames from the internal camera memory, and sets settings, such as exposure time and framerate.
Parameters **verbose** (*bool, optional*) – Additional information is printed to the command window. Maybe important for debugging purposes. Defaults to True.

openCamera()
Searches for attached cameras and opens the first found one
Returns success in opening the camera

Return type bool

configCam(*px_gain=1*, *camera_gain=1*)
Basic camera configuration, such as gain.
Parameters
- **px_gain** (*int, optional*) – Pixel gain (selects 8 from 10 bits, lower (0), middle (1) and upper (2) 8 bits), defaults to 1
- **camera_gain** (*int, optional*) – Camera gain, defaults to 1

setSettings(*exposure*, *fps*, *roi=(1024*, *1024)*, *rec_mode=1*, *sync=True*)
Sets camera settings.
Parameters
- **exposure** (*int*) – Exposure time in us (microseconds)
- **fps** (*int*) – Camera sampling rate in frames per second
- **roi** (*tuple or None, optional*) – if camera image should cropped to ROI (i.e. may run faster). If not desired, set roi=None, otherwise (height, width). Defaults to (1024, 1024)
- **rec_mode** (*XsCamera.XS_REC_MODE, optional*) – Recording mode, defaults to XsCamera.XS_REC_MODE.XS_RM_CIRCULAR
- **sync** (*bool, optional*) – If recording should be synchronized to trigger, defaults to True

getStatus()
Returns camera status. Status indicates if the camera is recording on a circular buffer or the camera is in idle mode or the recording is done.
Returns business and status
Return type bool, XsCamera.XS_STATUS

isIdle()
Determines if camera is not recording via XS_STATUS

startGrab()
Starts recording images on camera using previous set settings

stopGrab()
Stops data acquisition
Returns number of recorded frames
Return type int

live()
Returns live image using XsMemoryPreview :return:

correctForTrigger(*frame_index*)
Correct frame_index for trigger index, otherwise indexing is not historically
Parameters **frame_index** (*int*) – absolute frame index
Returns frame index relative to trigger
Return type int

getMemoryFrame(*frame_index*, *by_trigger=True*)
Creates a buffer and retrieves a camera frame by index from the camera's onboard memory.
Parameters
- **frame_index** (*int*) – The frame index
- **by_trigger** (*bool, optional*) – If the frame index should be relative to the trigger, defaults to True

Returns the camera frame in RGB (height, width, 3) or grayscale (height, width, 1)
Return type numpy.ndarray

updateTriggerPosition()
updates internally trigger position
Returns trigger position
Return type int

closeCamera()
Closes camera handle

10.3 Module contents

Python Module Index

o

Index

G

H

I

J

L

M

N

O

P

R

S

U