FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

ASSIGNMENT OF MASTER’S THESIS

Title:	Facial landmarks detection for the purpose of automated speech therapy
Student:	Bc. Laura Klimešová
Supervisor:	Ing. Lukáš Brchl
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2021/22

Instructions

The aim of this work is to design and implement algorithms that detect suitable landmarks in the video of the face for the purpose of automated speech therapy. Focus on the landmarks related to the tongue and mouth.

- Research existing solutions and explore appropriate speech therapy exercises.
- Design the minimum requirements for the camera and evaluation system of the task.
- Design and implement facial landmark detection algorithms using computer vision algorithms.
- Select a basic set of speech therapy exercises and implement methods for their automated evaluation.
- Evaluate the achieved results and suggest future improvements.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague September 29, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Facial landmarks detection for the purpose of automated speech therapy

Bc. Laura Klimešová

Department of Applied Mathematics
Supervisor: Ing. Lukáš Brchl

January 7, 2021

Acknowledgements

I would like to thank Ing. Lukáš Brchl for supervising my thesis, his guidance, support and advice. Secondly, I wish to thank Ing. Anna Čudáková, Jan Němeček and Michael Pavlista from SIT Port for providing me with the thesis topic, as well as their support and cooperation in obtaining the exercise descriptions and testing video data. I also wish to thank the ImproLab team of *FIT ČVUT* for lending me the equipment I used for this thesis. Lastly, I would like to thank my family and friends who supported me throughout this work and also volunteered to participate as test subjects.

Declaration

Citation of this thesis

Klimešová, Laura. *Facial landmarks detection for the purpose of automated speech therapy.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstrakt

Tato diplomová práce se zabývá detekcí obličejových landmarků s cílem vytvořit systém pro automatizované ohodnocování logopedických cviků. Důležitou součástí takového systému je schopnost detekce a sledování pohybu jazyka. K tomuto účelu je použita hloubková kamera Intel RealSense D415. Výsledná aplikace, která byla napsaná v jazyce Python, umí ohodnotit několik typických logopedických cviků ze živého záznamu z kamery či z předtočených videí ve formátu *.bag*. Byla provedena experimentální analýza dvou vybraných cviků s využitím video dat několika dobrovolníků zahrnujících dospělé i děti. V průměru byl systém schopen detekovat 65% provedených opakování cviků přičemž žádný ze cviků nebyl vyhodnocen předčasně.

Klíčová slova detekce obličejových landmarků, detekce jazyka, automatizace logopedie, strojové vidění, Python, hloubková kamera, Intel RealSense

Abstract

This diploma thesis focuses on facial landmark detection with the aim to create a system for automated speech therapy. Important part of this system is the ability to detect and track tongue movement. The Intel RealSense D415 depth camera is used for this purpose. The resulting application written in the Python language is able to evaluate several typical speech therapy exercises either from live camera footage or prerecorded videos in the *.bag* format. An experimental analysis of two selected exercises was performed using video data of several volunteers including both adults and children. On average the system was able to recognize 65% of the performed exercise repetitions without any of the exercises getting evaluated prematurely.

Keywords facial landmark detection, tongue detection, speech therapy automation, computer vision, Python, depth camera, Intel RealSense

Contents

List of Figures

List of Tables

Introduction

Speech disorders are a common problem for many people who can struggle with them throughout their lives. Children often undergo speech therapies to identify and remedy these issues. Part of the process is to strengthen facial muscles and their mobility through various exercises using their lips and tongue. The positive impact of these exercises is heavily dependant on frequency of performing them. While the most important part of speech therapy are training sessions with specialists, many patients could benefit from being able to do these exercises on their own with getting feedback on the precision of performed actions.

The aim of this thesis is to design and implement software to support automated speech therapy application. The outcomes are focused on landmark detection, tracking and exercise evaluation. The user interface of the application is being developed independently under the patronage of SIT Port in Pilsen.

The first two chapters of this thesis are dedicated to research of relevant terms and related works. The third chapter focuses on analysis of detecting relevant facial landmarks which includes benchmarks of existing models and comparison of different approaches of detecting the tongue. Outputs of these chapters will serve the following chapter which is focused on the practical part of this thesis – design of the solution and its implementation. The final chapter contains the experimental evaluation of practical outcomes of this work.

The resulting system will be beneficial not only for supporting automated speech therapy for children but also for patients recovering from strokes or different nerve damage diseases. The outcomes of this work can also serve as a support for systems built for human computer interaction using movements of lips and tongue.

Chapter 1

Theoretical background

This chapter serves as an introduction to the important research terms which are relevant for topic of this thesis. The following chapter 2 uses these terms further to explain what methodologies were used to build related solutions.

1.1 Computer vision

Computer vision is a field which focuses on analysis and processing of visual data in order to model the ability of human sight and translate it for the computers. Different kinds of sensors and cameras can be used for this purpose. However, compared to the human eye and brain, these sensors have only a projected information about the reality [1].

One of the goals of computer vision is automation of visual inspections which are prone to human error and bias. This affects various fields ranging from factory production lines to toll gates and border security [2].

1.2 Input data

Computer vision works with two dimensional digital image or video data as the input. Simply put, an image is a matrix of pixels. This can be written mathematically as a function of two variables defined as

$$\mathrm{I}[x, y] : \mathbb{N}^2 \to \mathbb{R}^n,$$

where $n = 1$ represents luminance (pixel intensity) and $n = 3$ represents color [3].

1.3 Image processing methods

The image data collected for solving computer vision tasks often is not ready for use from the start. Sometimes the image can contain defects such as low contrast or noise. Sometimes it is needed to enhance or even suppress certain parts of an image so that the subsequent work is easier. These are just some of the reasons to preprocess the image data before using it. There are several types of methods for preprocessing. The types differ mostly in the way of processing certain parts of input image in order to get a pixel value of the output image. The most basic operations use only one input pixel at a time to produce the output pixel value. Other operations use a local input pixel neighbourhood to generate the output pixel value. The latter processing technique is used with special operators, which are referred to as *image filters*. Image filters can enhance an image in different ways. Among the most well known cases are noise suppression filters and edge detection filters [1, p. 56-108].

Filters are applied to an image using an operation known as *convolution*. This operation is defined as

$$(F * G)[s,t] = \sum_{x} \sum_{y} F[x][y] \cdot G[s-x][t-y],$$

where F is the input image and G is a kernel of the image filter [4].

Kernel of an image filter in theory is defined as an n-dimensional mathematical function. One of the most well known examples is the Gaussian kernel which is uses the Gaussian probability distribution function for noise suppression. In practice, the kernels can be estimated by a simple 2D matrix of various sizes [1, p. 68-71].

When detecting edges in an image it is often useful to look at gradient of the image as its magnitude (the Euclidean norm of the gradient) can be used to pinpoint the edge location by pixel intensity changes. An important filter, which is used to estimate the second derivative of an image and detect its edges, is called a Laplacian filter [5].

Another interesting example is the Gabor filter. This filter is used for edge detection, texture analysis or optical flow estimation. Gabor filter is a product of Gaussian and sinusoidal signal [6]. The figure 1.1 showcases visualization of this filter in 3D space.

1.4 Optical flow

Optical flow (OF) is a term linked to the object motion analysis. It is used to represent motion changes in an image between frames. By computing OF it is possible to determine and track both direction and velocity of an object over time. For computation of optical flow there are two assumptions – brightness of points is constant over time and nearby points in the image move with a similar velocity. Firstly, a dynamic image as a function of position and time

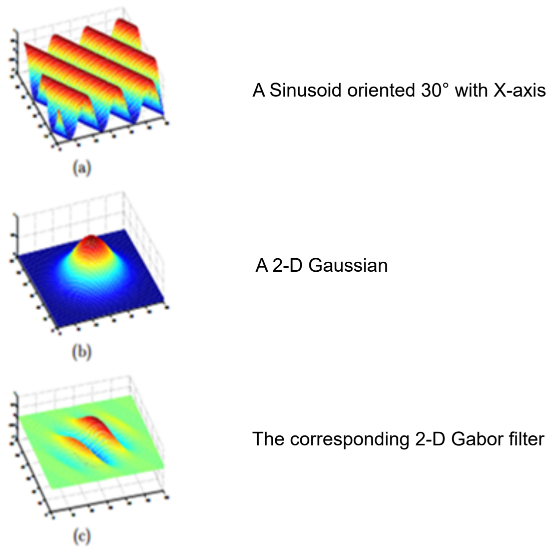


Figure 1.1: A 2D Gabor filter obtained by modulating the sine wave with a Gaussian [7]

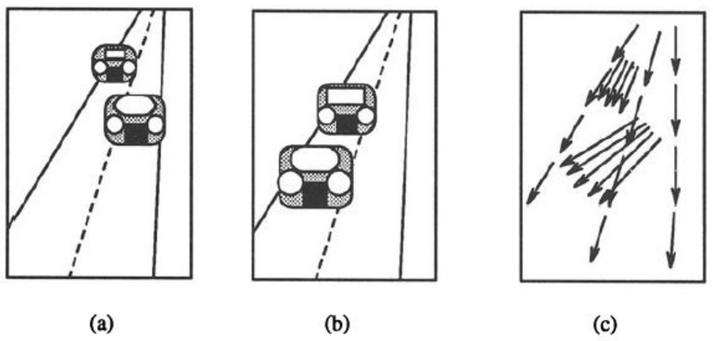


Figure 1.2: Example of optical flow computation: (a) Frame at time t_1 (b) Frame at time t_2 (c) Computed optical flow [1, p. 522]

is defined from the input image as

$$
\begin{aligned}
f(x+dx, y+dy, t+dt) &= f(x,y,t) + \frac{\partial f}{\partial x}dx + \frac{\partial f}{\partial y}dy + \frac{\partial f}{\partial t}dt + \mathcal{O}(\partial^2) \\
&= f(x,y,t) + f_x dx + f_y dy + f_t dt + \mathcal{O}(\partial^2),
\end{aligned}
$$

where $f(x, y, t)$ refer to a state of image in (x, y) in time t. The optical flow is determined by using an iterative method on two consecutive dynamic images [1, p. 507-513]. An example result of optical flow computation can be seen in the figure 1.2.

To simplify and speed up the computational process, several methods were designed to estimate the optical flow result. Some methods focus on computing

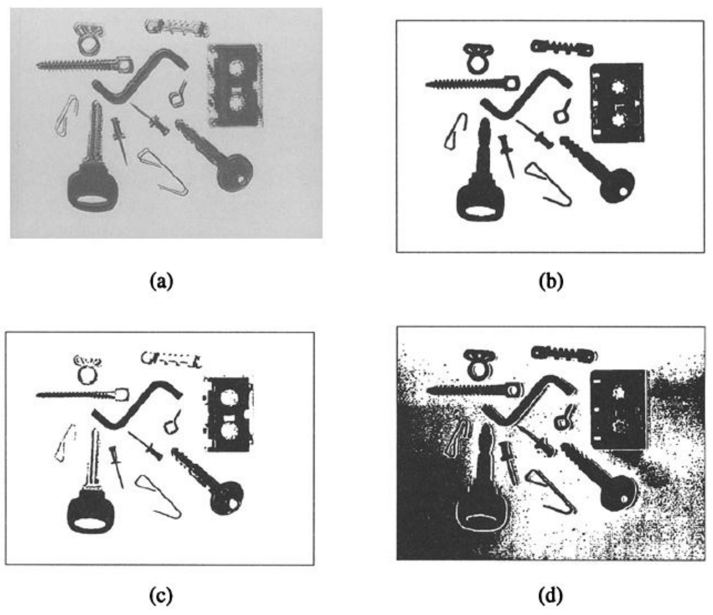


Figure 1.3: Example of segmentation using thresholding: (a) Original image, (b) threshold segmentation, (c) threshold too low, (d) threshold too high. [1, p. 113]

a sparse velocity field instead of a dense one for a subset of interesting points in an image which needs to be defined prior to the computation [1, p. 524].

Slower, but more accurate are the dense methods for estimating OF. One of such algorithms is named after its founder – Gunnar Farneback's optical flow method. The algorithm estimates disposition between two consecutive frames using the polynomial expansion transform [8].

1.5 Image segmentation

Image segmentation is an important task in processing of image data prior to their analysis. It aims to locate and divide an object in the image from its surroundings. The most basic method of separating an object from its gray image background is thresholding. This can work well with images which are taken under good conditions and have a simple background. The output image will only contain results which are above the selected threshold value. Selecting the optimal value for threshold is very important for this method as can be seen in the figure 1.3 [1, p. 112-113].

Edge-based segmentation methods are another approach to the image segmentation problem. These methods rely on edge detectors, for example edge detection image filters [1, p. 123].

Region growing segmentation is a method which works better with noisy images where edge detection algorithms can have problems. The general idea is to assign pixels in image to regions based on pre-defined conditions. These can range from pixel intensity levels to texture structures. The goal is to maximize each of the regions area [1, p. 164].

1.6 Methodology for solving computer vision tasks

Computer vision tasks can be solved by using various kinds of models which can be trained using the collected image or video data in order to gain insights.

Ensemble methods are using multiple classifiers together to share insights among themselves and classify samples together. There are two approaches to these methods. Bagging is considered to be the simpler approach. The idea behind bagging is to have multiple classifiers used in parallel. The group reaches the final decision by the majority of votes [p. 173][9].

Adaptive Boosting algorithm or *AdaBoost* is an ensemble method which was created as a better alternative to bagging algorithms. AdaBoost trains its weak learners using different subsets of training data, which are chosen for each learner probabilistically. Each classifier has a weighted vote and together with all the classifiers a final prediction is made based on the weighted majority [9, p. 179].

Support-vector machines (SVMs) are usually used in classification problems. The goal is to find a function which is able to separate samples of different classes. The classification accuracy tends to be more precise the greater the size of the margin separating the classes [9, p. 84-85].

With the popularity of deep learning, neural networks are often a choice for solving computer vision tasks. The *convolutional neural networks* (CNNs) are used in particular. Their architecture is designed to be able to process image data using much less parameters than classic fully connected feed-forward networks. As a result of this, CNNs can have more layers. One of the core concepts is the convolution layer. Each convolution layer has a predefined number of different image filters. Each of these filters will be convolved with the input image. Due to the feature extracting capabilities of the Gabor filter, they are often a good choice for the filters. The output features are represented by the filtered images. In one CNN there are usually multiple convolution layers. Convolution layers are usually paired with pooling layers which aim to gradually downsample the number of features while still retaining valuable information [10, p. 85-129].

A special case of CNNs are the *hourglass networks.* They are able to capture features of the input image at different scales. The architecture of the network is built using two mirrored CNNs. These kinds of networks are sometimes referred to as *encode-decode* networks or *convolution-deconvolution* networks. Convolution and pooling layers are used to change the output sizes. Each resolution scale will have its counterpart in the mirrored section of the

		Labels predicted by a classifier	
		pos	*neg*
True labels	*pos*	N_{TP}	N_{FN}
	neg	N_{FP}	N_{TN}

Table 1.1: The basic quantities used in the definitions of performance criteria [9, p. 212]

network. Firstly, a downsampling network is used to collect features at different resolutions. This is then followed by an upsampling network which collects and combines the features available from all the scales [11].

Several frameworks can be used to create and train a neural network model. Among the most popular are *TensorFlow* [12] and *PyTorch* [13]. TensorFlow has many variants for different use cases. For example, *TensorFlow.js* can be used for creating and deploying models in the browser [12].

1.7 Methodology for evaluation of classification performance

Classification is one of the most typical machine learning tasks. A classifier model can be trained to evaluate and assign one or more target classes to an input sample [9].

To validate how well can a trained model perform, some evaluation metrics must be defined. While a simple accuracy metric can be used as a ratio of correctly classified samples against the number of all samples in the classification, this metric tends to be not as informative, especially in a case of having more than one target class. Similarly defined error rate, which takes into account the number of all misclassified samples, can also give the wrong idea about the performance the more classes are used for classification. Another aspect to keep in mind is the balance of class samples used for testing the performance. In the table 1.1 a set of basic quantities is defined. N_{TP} and N_{TN} are the number of correctly classified positive samples and correctly classified negative samples respectively. N_{FP} and N_{FN} on the other hand represent the misclassified samples from positive and negative samples respectively [9, p. 212].

The table 1.1 can be referred to as the *confusion matrix* of a binary classifier [14]. For a model with more than one target classes, the situation is a little more complex as suddenly there are more than one value for each of the number of positive and negative quantities. The table 1.2 showcases a confusion matrix for a 3-class classifier. All the values on the diagonal are considered to be the number of correctly classified positive samples per each class. For a number of correctly classified negative samples per class, only the values which are not in the class's row and column are counted [15].

		Predicted		
		Greyhound	Mastiff	Samoyed
Actual	Greyhound	P_{GG}	P_{MG}	P_{SG}
	Mastiff	P_{GM}	P_{MM}	P_{SM}
	Samoyed	P_{GS}	P_{MS}	P_{SS}

Table 1.2: Confusion matrix for a multi-class classification model [15]

More informative and detailed measures, compared to the accuracy and error rate, are precision and recall. *Precision* marks the probability of the classifier being correct by labeling positive sample. It is calculated as

$$\text{Precision} = \frac{N_{TP}}{N_{TP} + N_{FP}}.$$

Recall measures the probability of the positive sample being recognized by the classifier. Similarly, it can be calculated as

$$\text{Recall} = \frac{N_{TP}}{N_{TP} + N_{FN}}.$$

Some use cases benefit from higher value of precision, other from higher value of recall. Precision can be a more valuable information when evaluating a recommending system for users. Recall on the other hand will be important when classifying medical diagnosis so that the patients are classified by the correct diagnosis [9, p. 211-218].

Another way to measure performance using the metrics defined above is to combine them together. This measure can be defined as

$$\text{F1 score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}},$$

and it stands for F1 score. This metric is helpful when it is not obvious whether to weigh precision or recall as the more relevant metric [9, p. 219].

Another interesting measure is the *Hamming loss* which refers to the average number of times an incorrect class is predicted by the classifier. It is normalized by the number of target classes. It is defined as

$$\text{Hamming loss} = \frac{1}{n_{\text{labels}}} \sum_{i=0}^{n_{\text{labels}}-1} l(\hat{y}_i \neq y_i),$$

where $l(x)$ indicates if the predicted value $\hat{y}_i$ matches the truth value y_i [16].

1.8 Face detection

Detection tasks in general can be viewed as a classification problem which aims to classify whether an object is or is not present in an image. In case the object is present then the location of the found object should be returned. Objects in images can be described by their salient features, such as their shape, color or texture. When in the context of locating object in one image, the process can be described as *object detection*. If the input data is a video sequence, then the goal can be to follow the detected object's position. This task is referred to as *object tracking* [17].

Face detection is a specialized case of an object detection task with the key focus on detecting human faces from images. It is a very challenging task as the human face appearance can take many different shapes and forms even if the perfect lighting and background conditions are assumed [18].

The following subsections focus on describing a selection of methods used for face detection.

1.8.1 Haar cascades

Haar cascades are a concept which was introduced in [19]. Firstly, the image features are obtained by applying different Haar filters on a training set. Different filters amplify different edge features. The training set contains both positive and negative samples. The method further introduces a variant of AdaBoost algorithm, which is used to first select a subset of samples and then train a classifier on the selection.

This method is implemented in the *OpenCV* framework [20].

1.8.2 Histogram of oriented gradients

Histograms of oriented gradients (HOG) were defined in [21]. This method aims to define a feature set to robustly describe an object to be detected. The local image gradients are used to model these features as they have good characteristic capabilities around important edges in the image. HOG descriptors are used on a set of training images which contain the object to look for. These are then combined with images which don't contain the objects and together they are used to train an SVM classifier with a Gaussian kernel.

This method is implemented in the *Dlib* toolkit [22].

1.8.3 Single shot MultiBox detector

This method for detecting objects in images was introduced in [23]. The implementation leverages a single CNN which returns a collection of same sized bounding boxes of an object along with scores of the object presence. The base of the CNN is appended with multiple feature layers with decreasing size. This allows for predicting detections at multiple scales. Multiple bounding boxes of different scales are defined along the ground truth bounding boxes. The network can learn many more different variations of each object class because of this.

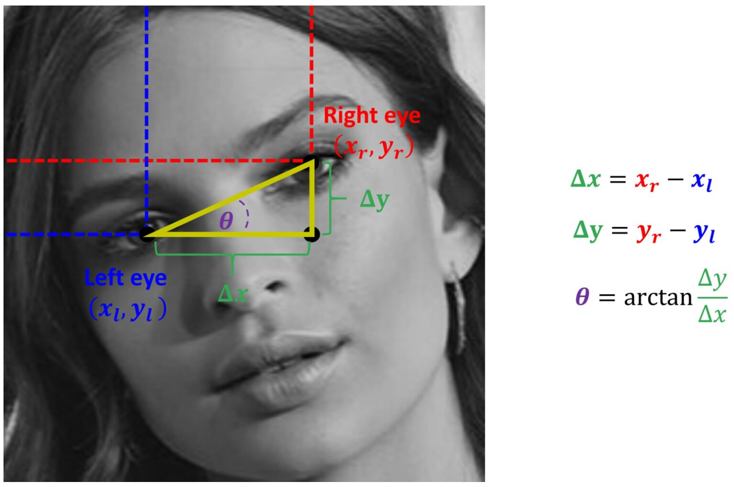


Figure 1.4: Calculation of angle for rotation in a face alignment task [26]

The final detection is preceded by a *non-maximum suppression* step. This processing technique was designed to merge multiple bounding boxes which may relate to the same area of the image. The greater the overlap of bounding boxes, the higher the score is for that image area. In the end the areas with scores above the threshold will selected and the windows overlapping with this area will be eliminated [24, p. 145].

A modified version of this model was created and trained to detect human faces. The output is a *TensorFlow.js* model named *Blazeface* [25].

1.9 Face alignment

Face alignment is a computer vision task which is often used for improving accuracy of face recognition or facial landmark detection pipelines. The aim is to find the optimal transformations of the input image to convert it in a standardized format. These transformations include for example scaling by a parameter or rotating by an angle [26]. The methodology of how the angle can be calculated is showcased in the figure 1.4.

1.10 Facial landmark detection

Facial landmarks can be described as key characteristics on a human face. Among the commonly detected landmarks are for example eye pupils, nose tip or mouth corners. The analysis of these key points is beneficial for various computer vision use cases including biometry (e.g. face recognition for secure authentication), face tracking or emotion detection. Typically the facial landmark detection process is preceded by several sub-steps, most notably detection of a face region, face alignment, and color and illumination preprocessing [27].

There are different ways to categorise facial landmarking algorithms. In [28] they divide the algorithms into holistic, regression-based and Constraint Local Model methods.

In the recent years, the use of deep neural networks to solve facial landmarking have been becoming more popular. With the right amount of computational power these methods can outperform the traditional methods in both processing time and accuracy. Their biggest advantage is their greater robustness in facial variety as well as working around more challenging conditions related to lighting, facial occlusions and steeper head angles [29].

I choose a selection of various methods which I explore in more detail. These methods have different approaches to the landmark computation. The selected methods are further explored in the chapter 3, where I perform benchmarks and comparisons. The sections 1.13, 1.14 and 1.15 describe methods using more traditional machine learning algorithms. In the sections 1.16, 1.17, 1.18 and 1.19 I explore some example methods using deep learning.

Before exploring the details behind each method, the sections 1.11 and 1.12 showcase how landmarks can be mapped in an image and some state-of-the-art datasets which can be used to train and test the solution's performance.

1.11 Landmark markups

Various methods output different facial landmarks. Some methods use only main facial landmarks, such as eye pupils, note tip and mouth corners, while others output a larger amount of detail. One of the most notable example is the 68 facial landmark markup which is the common format for many state-of-the-art datasets [30]. It can be seen in the figure 1.5.

1.12 Datasets for face-related tasks

While methods and technologies used are a backbone of a facial landmark detection system, these methods need data to train. There are several state-of-the-art datasets which are often used for training and testing landmark detectors or face alignment pipelines. Their variety in subjects, angles and luminance greatly improve robustness of such trained models as well as thoroughly test how this method would handle difficult conditions. Among the notable examples are *HELEN* with 194 annotated landmarks [32], *LFPW* with 35 annotated landmark points [33].

To unify data across multiple databases as well as to standardize the landmark markup scheme, the *300W* dataset combines selection of other datasets like LPFW and HELEN. The resulting database contains all of the samples annotated with 68 landmark markup [34].

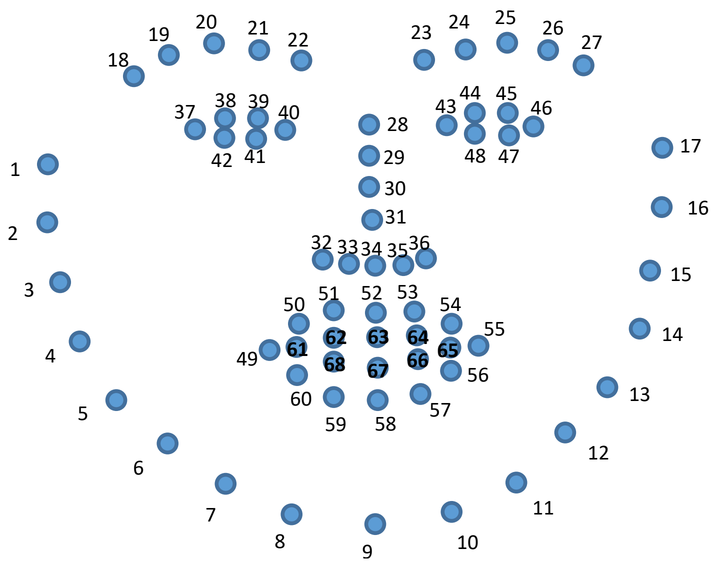


Figure 1.5: 68 facial landmark markup [31]

Datasets are not limited to 2D space only. *300W-LP* dataset is a synthetically created expansion of the 300W. Each sample is annotated with coefficients of the 3D Morphable Model which is used to map a 2D image of a human face into 3-dimensional space [35].

1.13 Active Shape Model

Active Shape Model (ASM) was introduced in [36]. It belongs to a branch of methods known as flexible models. These models are typically used to describe cases of objects which have high variety of shapes, such as the human face or body organs. Similar methods to ASM are the Active Contour models (also known as *snakes*). There are differences between the two, most notably ASM employ a global shape constraint. This means that a shape can only deform in ways which were found in the training set.

The method focuses on statistics of landmark coordinates. In order to be able to compare the statistic properties of landmarks across multiple examples from the training set, an alignment must be performed first to standardize the samples. This is done iteratively by rotating, scaling and translating each shape in the training set by a calculated mean shape from previously aligned shapes. Once all samples have been aligned a point clouds of all the landmarks in the set can be used by a Point Distribution Model to determine variance of coordinates within the individual clouds. An important thing to note is that the landmarks usually don’t move independently but are partially correlated with one another [36].

1.14 Active Appearance Model

Dating back to the year 1998, [37] introduced the Active Appearance Model (AAM) as an extension of the original ASM. It obtains the shape model using the Point Distribution Model just like ASM. Additionally, the model preserves image information and combines it in the statistical model along with the shape. That is done in such a way that after a mean shape of the training set is obtained, each example image in the training set is aligned to match the mean shape. All grey images in the training set are also normalized so that the differences in lighting conditions are suppressed as much as possible to not intervene with the statistical properties of the model. The normalized image vectors as well as the shape vectors are combined under one linear model using the principal component analysis.

A pre-trained instance of this model, which was trained on multiple datasets including LFPW and HELEN, is available through the *Menpo* library [38].

1.15 Ensemble of regression trees

One of the most well known tools for facial landmark detection is provided by the *Dlib* toolkit. It uses HOG for face detection and an ensemble of regression trees as a for pose estimation. The Dlib's version of the model was trained on the 300-W dataset to output 68 facial landmarks [39].

The details of this procedure were described in [40]. Let $\mathbb{S}$ be the shape vector containing 2D landmarks of image I. The current estimate of this vector is noted as $\hat{\mathbb{S}}^{(t)}$. Each regressor $r_t(\cdot,\cdot)$ in the cascade updates the vector using the following formula:

$$\hat{\mathbb{S}}^{(t+1)} = \hat{\mathbb{S}}^{(t)} + r_t(I, \hat{\mathbb{S}}^{(t)}).$$

Each regressor makes its prediction using the features of the original image. The regressors are trained using gradient tree boosting algorithm with a sum of squares loss. To work around missing landmarks during the training phase, a weight coefficients are introduced. Missing landmarks are simply marked with weight of 0 [40].

1.16 Joint Multi-task Cascaded Convolutional Networks

The Joint Multi-task Cascaded Convolutional Networks (MTCNN) were first described in [41]. It is a method which aims to solve face detection and face alignment as well as detection of 5 major facial landmarks – eye pupils, nose tip and mouth corners. In order to unify the detection and alignment tasks this solution combines three cascaded CNNs in its framework. This framework is designed as lightweight as possible to enable realtime performance.

Firstly, an image pyramid is built from the input by making several scaling operations. All these images are fed as an input into Stage 1 network (P-Net for Proposal Network). This CNN outputs bounding boxes for each of the inputs. Before the move to the second stage, there are calibration and merging operations happening to eliminate bounding boxes with too many overlaps. In the Stage 2 network (R-Net for Refine Network) the candidate bounding boxes get even more refined by reusing the same methodology as in Stage 1. Finally in Stage 3, the network (O-Net for Output Network) returns the detected facial landmarks along with the final bounding box [41].

1.17 Deep Alignment Network

The Deep Alignment Network (DAN) introduced in [42] is a robust deep learning-based face alignment method. It was trained on the 300W dataset. The method describes a novelty approach for solving face alignment by processing entire images rather than patches extracted around certain landmarks.

The initial input into the network is a warped image. The network refines the detected landmark location over time by chaining layers into several stages. Each stage has a Connection layer which outputs a landmark heatmap, estimate of the input image transformation and a feature image. These three are then fed as input into the consecutive stage's feed-forward neural network for further refinement. The landmark heatmap is a special concept of the solution which marks the individual landmark location with intensity so that the closer the landmark is, the higher the intensity value is in the heatmap. With this approach the landmark estimates can be used and refined by the subsequent network stages as the previous positions are reflected by the heatmap [42].

The figure 1.6 shows how the outputs look after going through the stage process. The first column marks the initial input image, the second column shows the image transformation estimate, the third column contains the landmark heatmap estimate and the fourth column marks the feature image estimate [42].

1.18 Face Alignment Network

The Face Alignment Network (FAN) was introduced in [43]. This method can be used for both 2D and 3D face alignment. The network was built using 4 stacked hourglass networks, however instead of a bottleneck block there is a custom hierarchical multi-scale residual block.

The network for 2D and 3D face alignment is using the same architecture. The training was done using the 300W-LP dataset which was divided into 2D and 3D versions based on their annotations [43].

The paper further explores a way to create a 3D face alignment dataset using a network which will convert data of a 2D dataset into projected 3D

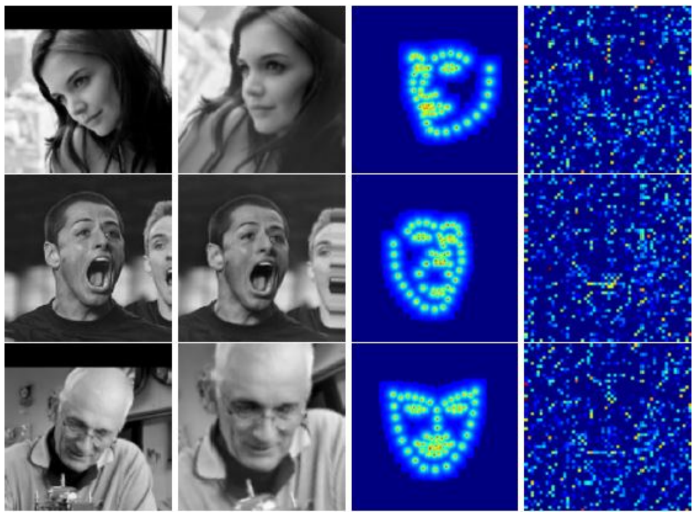


Figure 1.6: Outputs of a DAN stage [42]

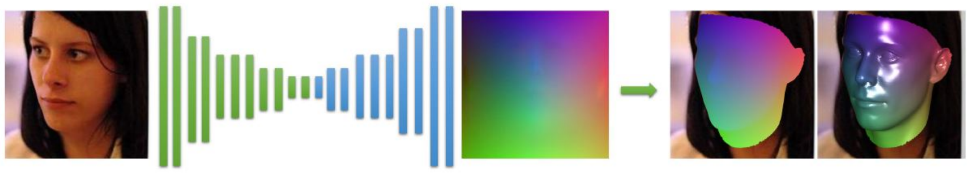


Figure 1.7: The architecture of PRNet [45]

landmarks. The network's design is based on the design of the FAN and it was also trained on the 300W-LP which contains both 2D and 3D annotations. The resulting network was used to generate a whole new 3D face alignment dataset – *LS3D-W*. This synthetic dataset consists of approximately 230000 images [43].

Face detection is a part of the FAN implementation along with the facial landmark detection model. There are different options for face detectors included, notably *Dlib* and *Blazeface* [44].

1.19 Position map regression network

Position map regression network (PRNet) was introduced in [45]. This end-to-end method was designed to predict dense alignment out of which a 3D face shape is reconstructed. A trained encoder-decoder network firstly maps a 2D image into 3D coordinate space using a UV position map which was designed for this solution specifically. The process of the method is highlighted in the figure 1.7. The used dataset for training was the 300W-LP. The resulting network takes about 9.8 ms to process a 256×256 input RGB images.

1.20 Tongue image processing

Difficulties of working with tongue image data and trying to analyze the outputs are likely linked to the unique physiology of the tongue. Tongues have no skeletal structure, they can bend, protrude and twist freely which makes them difficult to track for a specific shape [46].

There are different tasks in computer vision for which tongue images are used. Two main processing techniques which are relevant for the topic of this thesis are tongue image segmentation and tongue tracking. Processing of tongue images has various uses from detecting illnesses to pattern recognition for biometric purposes.

One example of tracking tongue movement was described in [47]. This system uses the Gunnar Farneback algorithm for estimating dense optical flow. It uses the entire face frame to compute the points where the flow magnitude is over a certain threshold. The application is implemented in Matlab.

Chapter 2

Related work

In this chapter I focus on describing existing applications and research fields that are relevant for the topic of this thesis. The findings of these works help me derive the most suitable solution for the expected outcomes of this thesis. While there are several applications dedicated to the topic of automated speech therapy I am also considering other use cases which deal with processing image data of lips and tongue.

2.1 Traditional Chinese medicine

The traditional Chinese medicine (TCM) is a discipline of life science originating from ancient China. It focuses on research of human health and disease prevention. One of the key elements of TCM is tongue diagnosis. It is used to evaluate patient's health based on tongue features such as color or texture [48].

For the purpose of automation there are several applications which focus on either preprocessing tongue image data or combining computer vision algorithms to help analyze the input. The data used for this purpose are mostly images of fully protruded tongues from frontal view.

2.1.1 Color based tongue segmentation using active contour

The solution described in [49] tackles the tongue segmentation by using active contour. The initial contour is approximated by analyzing the tongue image in HSV space. The most prominent features were located in the Hue and Value component of the image. After separating the selected components, a binary image is created for each of these subset images. Binary image simply creates a black and white image from the input based on intensity of each pixel. If the pixel intensity reaches a threshold then in the output there will be a white pixel assigned. For pixels under the selected threshold there will be a black pixel assigned in the output. Fusing the hue and value binary images gives

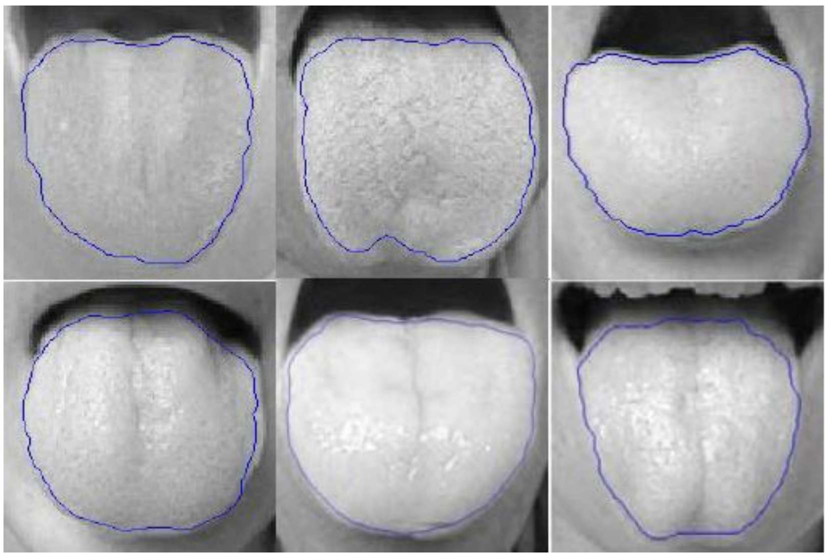


Figure 2.1: Results of found tongue contours [49]

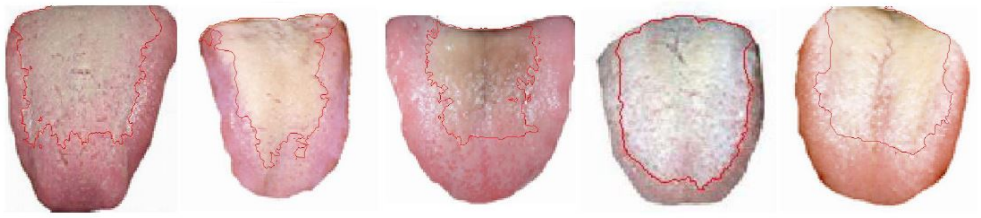


Figure 2.2: Results of tongue coating segmentation [50]

the initial contour for later processing. The initial contour is being evolved by minimizing an energy function for separating tongue area and skin. Sample results can be seen in the figure 2.1.

2.1.2 Separation framework of tongue coating

The same research team used this algorithm further for separating tongue coating from tongue image as described in [50]. The sample results are showcased in the figure 2.2.

2.2 Human computer interfaces

A human computer interface (HCI) is an assistive technology that serves as means of communication between a computer and its user. In the past the most common ways of communicating with computers were hardware machines which transmitted user's input into computer instructions such as track

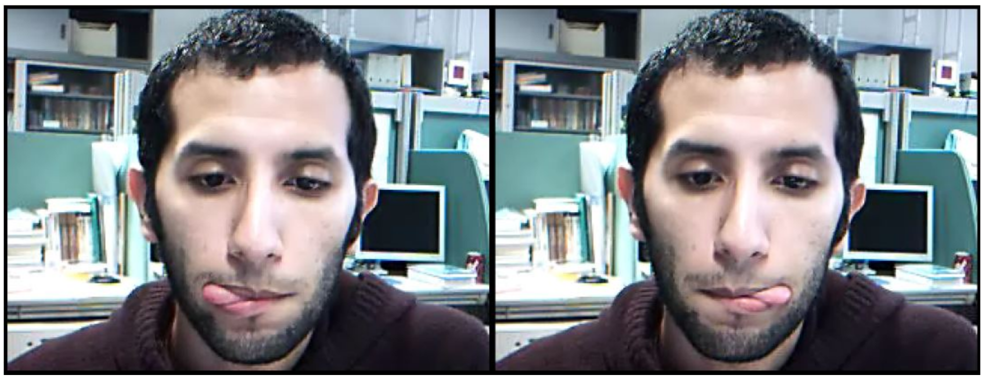


Figure 2.3: Gestures for detection [52]

pads, touch screens and keyboards. The problem with using these devices is when the user has limited control over their motor functions. In order to enable people with disabilities to make use of electronic devices, researchers shifted their focus to alternative methods of controlling computers. These methods mimic traditional input from hardware devices by implementing various gesture detection systems which can be taught to detect subtle facial expressions such as eye-blinking [51].

In this section I focus on HCI solutions which use mouth and tongue detection as a part of their workflow.

2.2.1 Tongue side protrusion gestures detector

For the purpose of human computer interaction, [52] designed a system for evaluating tongue protrusion gestures from a simple web camera. Two different gestures are being detected – tongue protruded on left and right side of mouth as shown in the figure 2.3. The system requires for the mouth to be closed during evaluation. Initially, for segmenting the mouth region this solution uses Haar cascades to detect face and then cropping bottom third of the detected face. Width is estimated by the detected position of the eyes. The mouth region is further processed with a Gabor filter to locate the mouth line connecting the corners. This initial mouth line is then used for tracking of the mouth. The protrusion is located by comparing normalized correlation coefficient value with both halves of initial mouth template over time.

2.2.2 Camera-based tongue computer interface

The solution described by [53] implements an assistive technology which enables the users to control a joystick using 6 different tongue and mouth gestures – closed mouth, opened mouth, tongue up, tongue down, tongue left and tongue right. As a first step, the mouth region of interest is located and transformed so that the input images match in size. The AdaBoost algorithm is used to classify state of mouth. Each gesture has its own strong classifier. This prototype was tested in a first phase of evaluation of the interface. For the second phase of testing there was an addition of training samples collected from user during the evaluation process. The previously trained strong classifiers output top 3 most probable gestures. These findings are evaluated by a final classifier which is trained on data captured by the user. While the accuracy of the gesture improved with the second phase of evaluation, there has been an increased latency of the classification.

2.3 Automated speech therapy

There have been solutions built in the past for the use case of speech therapy automation. I keep a strict focus on applications analyzing image data only while omitting the ones that perform their analyses on audio data. There are also applications using contact devices for the user to be worn or attached to their mouth such as magnetic sensors placed on the tongue. These solutions will also not be included in this section as they are not as relevant to the use case of this thesis.

2.3.1 sMilestone

This medical software was designed for post-stroke patients in order to speed up the process of speech recovery. The solution uses the Microsoft Kinect 1.0 sensor to capture depth features of patient's face. Along with the Kinect sensor, the application leverages the Microsoft's Software Development Kit (SDK) including most notably the Face Tracking SDK. It enables to track 87 facial landmarks in real-time. For tracking the tongue tip the application first locates the mouth area and searches for a point closest to the sensor in this bounded space. With further image data readings the location of the tip is refined even more by searching just the area where the previous closest point was found. The application also takes into account that the depth field roughness of the Kinect. In order to keep the detection smooth the localized tip changes only once every few frames. As a proof of concept the authors also developed a game which uses this tongue tip tracking system. The application requires the user to stick their tongue out as much as possible in order to find the correct tongue tip position [54].

Important outcome of this work is its focus on depth information and how it can accurately generalize tracking of tongue tip. With the use of the Kinect sensor the authors were able to track tongue tip in realtime but with a major constraint by needing the users to protrude their tongue fully out of their oral cavity before controlling the application. However, the speech therapy exercises often include more subtle movements of the tongue.

The desktop version of Kinect devices is not being sold anymore [55] which further limits the application's use.

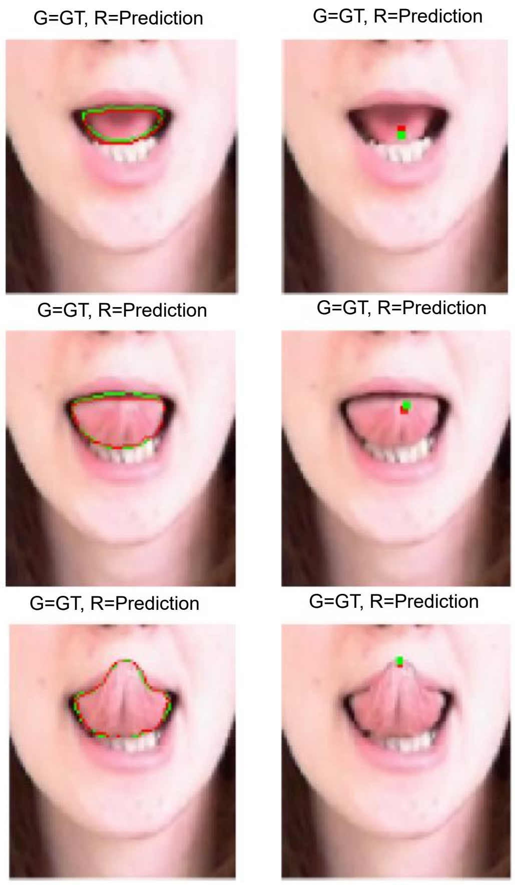


Figure 2.4: Results of tongue segmentation and tongue tip localization [58]

2.3.2 AssistLT

This project was funded by the Technology Agency of the Czech Republic with the task to develop a system for automatic speech therapy evaluation. The project started in the year 2018 and ended in the year 2019 [56].

Technical details of the solution were published in [57]. Since the intended user base of the application are children, there was a focus on augmented reality features that make the speech therapy more engaging. For face landmark detection the *Dlib* library is used. Using landmarks there are several features being calculated on top of them to validate mouth states such as closed mouth, opened mouth, smile, etc. For tongue related exercises the solution implements a CNN which outputs segmented tongue area mask and a tongue tip are mask out of single image input. Resulting outputs of the implemented CNN are showcased in the figure 2.4 where ground truth area is marked green and predicted area is marked red.

The solution is still under active development. One of the newer features is teeth detection for use with additional speech therapy exercises. Another addition is a human computer interface mode which leverages the current tongue detection system for controlling keyboard using tongue movements [58].

Chapter 3

Analysis

In this chapter I analyze possibilities of solving each of the works's sub-tasks based on both their performance and usability in order to design the most suitable solution in the following chapter.

3.1 Facial landmark detection

In order to find the most suitable approach for landmark detection I have performed benchmarking of several landmarking models. I have chosen *Dlib* [39] and pretrained AAM model from *Menpofit* [38] library for Python as the more traditional approaches to facial landmark detection. From deep learning models I have chosen *MTCNN* [59], *DAN* [60], *FAN* [44] and *PRNet* [61].

3.1.1 Selected datasets

The methods I selected for testing were trained on different state-of-the-art facial landmarking datasets, such as *300W* and *LFPW*. To evaluate used algorithms I looked for datasets which weren't part of the training process and still maintained decent level of difficulty in terms of lighting and angles.

First dataset I chose was the BioID Face database [62]. This dataset consists of 1571 gray images. Each image is manually annotated with 20 landmarks. The mapping of the landmarks is shown in the figure 3.1. The resolution of all the images is 384×286. The dataset was built on the facial data of 23 different individuals (including people of different races, genders and facial features such as beards and glasses) in various background surroundings.

Second dataset I chose came from Kaggle [63]. This dataset contains over 7000 images with up to 15 marked landmarks. However, not all of these images have all the 15 landmarks annotated. I created a selection of images containing all the available landmarks. There was 2140 in the final image subset. Each image has a fixed size of 96x96 pixels. Several sample images of this dataset can be seen in the figure 3.2 including the ground truth landmarks.

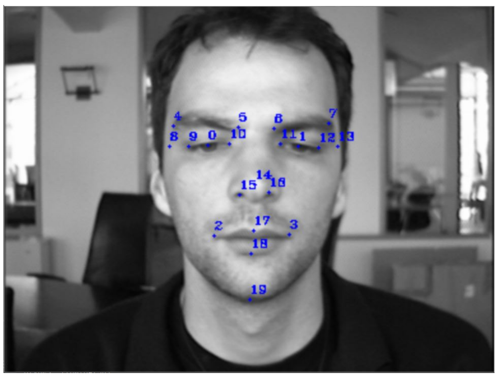


Figure 3.1: Ground truth markup of BioID dataset

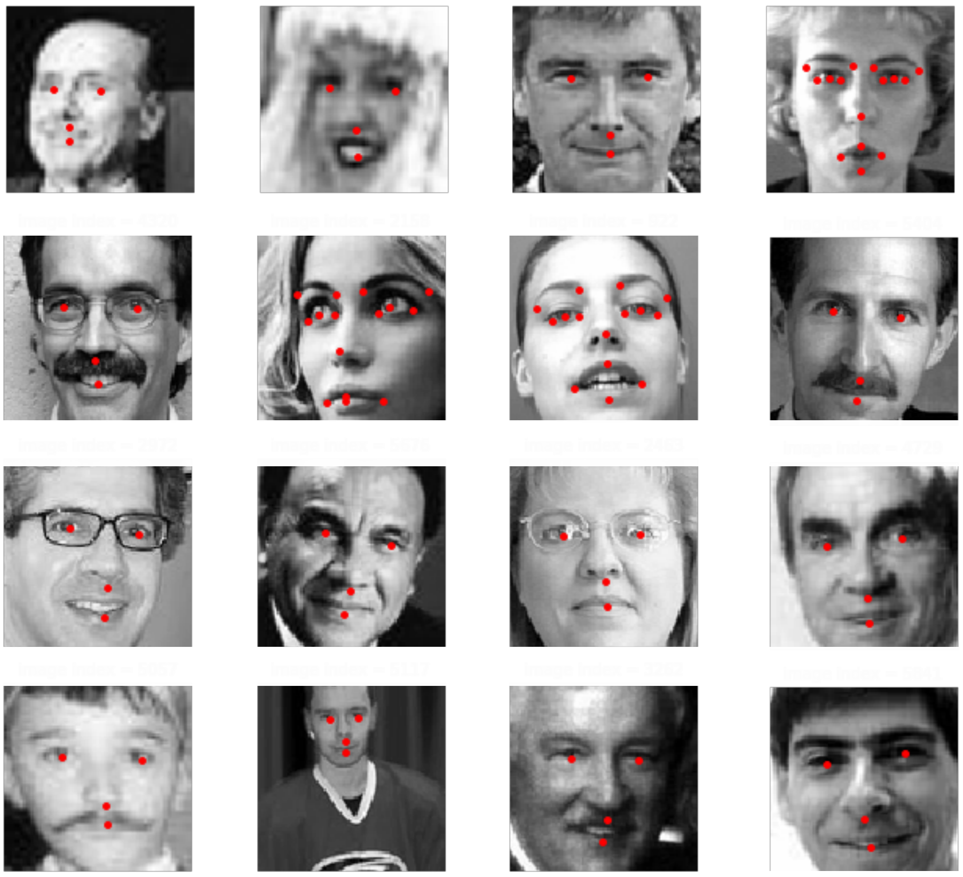


Figure 3.2: Sample images of the Kaggle dataset including ground truth landmarks [64]

3.1.2 Error metric

From the selection of models I measured their accuracy while considering that not all of the chosen models output the same amount of facial landmarks. The other thing to consider was also the amount of landmarks that were available for the selected datasets as well as their order in the output. I settled on 5 key facial points – pupils, nose tip and mouth corners. This mapping is used by the *MTCNN* model.

On the selected point subsets I calculated the normalized mean error [65] which can be defined as

$$NME = \frac{1}{N}\sum_{i=1}^{N}\frac{\sqrt{\Delta x_i^2 + \Delta y_i^2}}{d},$$

where N is the number of landmarks and d is the normalizing coefficient for which I used inter-pupil distance. Δx_i and Δy_i denote the difference between the ground truth and predicted coordinates of the i-th landmark. The numerator part of the fraction is the formula for Frobenius norm.

3.1.3 Accuracy comparison

After saving the computed features from all models and datasets, I used the *Pandas* library to process and analyze the outputs. Then I used the *Matplotlib* library to create the plots. The calculated error distribution for each of the benchmarked methods can be seen in the figure 3.3. The *DAN* model was the most accurate from the selection with 86 % of errors on selected datasets were under 0.25 in terms of point-to-point error. While *PRNet* was outperformed by the other models in tests on selected datasets it has shown much better performance on testing videos of people practicing the speech therapy exercises captured for this thesis. They were taken with a phone low quality front camera without fixed face position and angle of the subjects. The model showcased better ability in locating lip landmarks in difficult conditions of tongue being fully protruded falling third only to the *DAN* and *FAN* models.

3.1.4 Processing time comparison

All experiments were run on a machine with the following specifications:

- OS: Ubuntu 20.04.1 LTS 64-bit
- CPU: Intel(R) Core(TM) i5-7600K @ 3.80 GHz
- RAM: 32 GB
- GPU: NVidia GeForce RTX 2080 Ti (11 GB GDDR6)
- CUDA version: 10.1

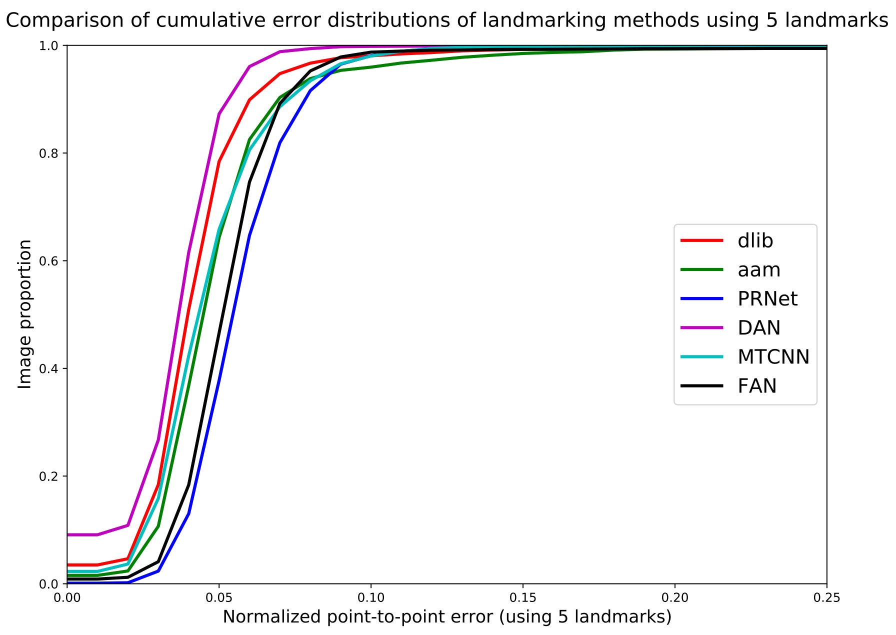


Figure 3.3: Normalized point-to-point error distribution using 5 landmarks

As there were images in the tested datasets which included different image sizes I had to factor this into the final result. For each image I calculated runtime per pixel. For every landmarking method I computed the average across all the images in both datasets. The comparison results are showcased in the table 3.1. In the same table I estimated how long it would take to process an image of size 640 × 480 pixels using each of the methods from the measured value. *Dlib* turned out to be the fastest of the chosen methods in terms of processing runtime per pixel. However the model's performance was much slower when tested on videos of speech therapy exercises. In that setting, the *FAN* model was the fastest performer.

3.2 Tongue tip localization

In this section I aim to analyze possibilities for locating tongue tip which is the most important information I need for evaluating tongue related exercises. While not each of these options can determine the tongue tip location, they can still be used as a stepping stone in evaluating different gestures. I focus on the positives and negatives each of these options can bring.

Table 3.1: Comparison of average runtime per image pixel for benchmarked methods

Method	**Average runtime per pixel [ms]**	**Estimate runtime on** 640 × 480 **picture [s]**
Dlib	4.35×10^{-4}	0.13
AAM	4.01×10^{-2}	12.31
PRNet	6.56×10^{-3}	2.01
DAN	1.31×10^{-3}	0.40
MTCNN	6.90×10^{-3}	2.12
FAN	2.22×10^{-3}	0.68

3.2.1 Optical flow

For tracking using OF there are options to use either sparse or dense methods. While dense methods can be more accurate, they are much more computationally demanding than sparse methods. For sparse tracking there would be a selection of points in the oral cavity which would be used to analyze tongue movement. Based on the vectors I would estimate the direction of the tongue's movement.

+ No prior data training & annotating
− Needs to save the previous frame data to work
− Does not locate the tongue tip by itself
− Computationally demanding
− Accuracy is dependant on the user's movement and camera equipment

3.2.2 Detecting tongue gestures using a CNN

This option would require to train a CNN which would classify image gestures such as tongue tip located up, down, left, right. Similar approach was described by [66].

+ More robust and precise than OF
+ Needs only the current frame to work
+ Annotation of training data would be simpler with labels rather than coordinates
+ Can train for special states of mouth e.g. opened without tongue visible, closed

– Exercise evaluation too broad e.g. cannot react to improvements of patients, exercise evaluated as a false positive

– Would require a lot of training data

3.2.3 Detecting tongue tip coordinates using a CNN

Instead of detecting broad gestures the CNN could learn to detect one landmark on tongue image – tongue tip. Dataset for training would require manual annotation of estimated coordinates. In order to ensure the necessary robustness there would have to be a large number of images of various groups of people.

+ More robust and precise than OF

+ Needs only the current frame to work

+ Exercise evaluation could be very precise – calculation of tongue tip distance from a lip point

– Annotation of data would be a complex manual work

– Would require a very large amount of training data to be robust enough

3.2.4 Detecting tongue tip coordinates using a depth sensor

Using this method I would use depth information of the input to locate tip of the tongue. Based on location of the found tip coordinates the implementation would estimate the corresponding gesture. A reliable face landmark detector would be needed to locate mouth area as accurately as possible. Locating and tracking a minimum point in a segmented region would be less expensive computation compared to calculating OF. For users it would mean to make a prior investment into a depth sensing camera device however there would no longer be dependence on the user's camera quality.

+ No prior data training/annotating

+ Needs only the current frame to work

+ Exercise evaluation could be very precise – calculation of tongue tip distance from a lip point for detecting the true location

+ Would not depend on user's camera set up at all

– Additional investment into a depth sensor needed (both for testing & for application users)

– Output may not show tongue tip but the most protruded tongue part

- Sensitive to the user's head tilt angle
- Performance and quality loss due to the physical connection of the sensor

3.2.4.1 Depth sensors' comparison

There are several options for external depth sensor cameras which could be used for this particular problem. I am including only the devices which are available to buy which is why I am not including Microsoft Kinect[1]. I am comparing them based on their capabilities, specifications and costs.

Intel RealSense D415 The D415 model is priced for \$149. The depth technology used is active infrared stereo. The camera's built-in infrared projector enables the sensor to pick up finer details in the scene [67]. It supports depth resolution up to 1280×720 pixels and frame rate up to 90 FPS. The depth field of view is given at $65° \pm 2° \times 40° \pm 1°$. The depth sensor can capture range from 0.16 metres to 10 metres. The RGB sensor uses the rolling shutter and supports resolution up to 1920×1080 pixels with 30 FPS frame rate. The field of view of the RGB sensor is $69° \pm 1$Ö$42° \pm 1$ [68].

Intel RealSense D435 While the depth technology used by this model is also active infrared stereo like the previous model, there are several differences. Compared to the D415 model the depth sensor has a greater field of view with $86° \times 57°(\pm 3°)$. Additionally it can detect depth from minimal distance of 0.106 metres. These differences allow for better capturing of a wider space as well as the option to capture elements much closer to the sensor. The parameters regarding the RGB sensor are identical to the D415 model. The price of the device is \$179 [68].

Orbbec Astra Series This product line consists of Astra, Astra S and Astra Pro with all three products priced at \$149.99 [69]. While the Astra S has a range between 0.4 and 10 metres, the Astra and Astra Pro have a range between 0.6 to 8 metres. All three models have the field of view of 60°H × 49.5°V × 73°D and they also share the depth resolution specification of 640×480 pixels at 30 FPS frame rate. The Astra Pro's RGB sensor supports resolution up to 1280×720 pixels at 30 FPS while the Astra and Astra S support up to 640×480 pixels at 30 FPS [70].

The devices by Intel are compatible with the software development kit which is available across different platforms and can be used by different programming languages. Another part of this kit is the Intel RealSense Viewer application which provides a user interface for the camera. Besides testing

[1] See subsection 2.3.1.

different settings and filters of the camera stream, it can also be used to capture videos to *.bag* compressed files and play them back [71]. The cameras can also be used with an Android phone or tablet by connecting a USB On-the-go (OTG) device using an input connector suitable for the target device. The authors of the *librealsense* library have released a sample application in the Google Play store, which showcases communication between the external camera and an Android device via OTG connection [72].

The Astra device line can be used programmatically with the Astra SDK which is supported on numerous platforms. There are also additional features available such as the Orbbec Body Tracking library which needs to be purchased separately for a license fee of $74.99 [73].

Chapter 4

Design and implementation

The Intel RealSense D415 depth camera was selected for capturing image and depth data stream. Its depth sensor's technical parameters are sufficient for the use case of this thesis. The RealSense SDK will be used to capture and process RGB and depth data programmatically.

4.1 Requirements and prerequisites

In order to be able to use the application there is a couple of requirements on the user's setup so that the optimal quality of the output is assured.

The following list of components must be used:

- Intel RealSense D415 camera
- USB-C to USB-B 3.2 cable (included with the camera)
- a tripod with a 1/4-20 UNC mounting thread

There are also requirements for how the user should proceed when using the solution:

- User should be using the application in a reasonably lit room. There does not need to be a special lighting in place directed at user but the face should not be occluded with too much shadow.
- The camera should be seated on a tripod. The camera must be leveled with the ground in order to capture face from 90° angle. The camera needs to be placed approximately between 40 cm and 45 cm away from user's nose tip. The camera height should be set in parallel with the user's lips. The set up is further showcased in the figure 4.1.
- User should sit straight and keep the head movements to the minimum.

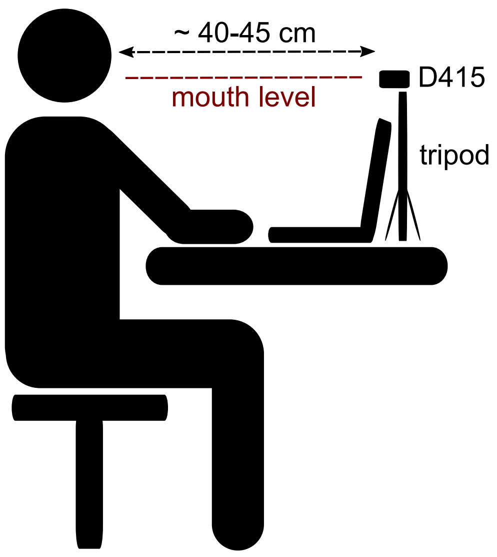


Figure 4.1: Camera setup guidelines (vector image edited from [74])

- Each exercise should be performed as slowly as possible. This way the exercise is most effective as well as easier for the application to process.
- When performing exercises with the tongue, the mouth should be opened as much as possible.

4.2 Workflow of the application

The application will be performing the following set of subtasks:

1. Load current image and depth data
2. Face detection
3. Facial landmark detection
4. Mouth localization
5. Calculation of mouth measurements
6. Detection of mouth state (closed/open)
7. Detection of tongue gesture
8. Update of exercise sequence based on detected gesture

4.3 Detecting facial landmarks

The first key step in the application's workflow is to detect user's face and the facial landmarks. The most important landmarks for the application logic are around the lips. Following the analysis of different algorithms I have chosen the FAN network as means to detect facial landmarks. Upon detecting and locating these landmarks I calculate several measurements such as lip height and height of the oral cavity. Based on these measurements, specifically if the height of oral cavity is larger than height of the thicker lip, I estimate whether the mouth is opened or closed.

4.4 Detecting and tracking tongue

For estimating tongue tip movements I follow a set of constraints which can be described by pseudo code 1.

Algorithm 1: Tongue tip tracking

```
Data: Color and depth image stream
Result: Tongue tip estimation point
initialization;
while stream is running do
    find landmarks;
    check if mouth is opened;
    if mouth is open then
        if minimum point is not being tracked then
            find minimum in oral cavity;
            if found minimum is close to depth threshold then
                save new found minimum for next frame;
            end
        else
            find new minimum in previous minimum point's area;
            if new minimum is in mouth bounds then
                save new found minimum for next frame;
            else
                release the current minimum point from tracking;
            end
        end
    end
end
```

When locating the initial tongue tip, the algorithm is looking for a depth minimum within the oral cavity. In order not to misdetect teeth as tongue tip candidate the initial area of search is located near the mouth center and the radius of the area is approximately 1/4 of located oral cavity. A minimum is considered to be the tracked tongue tip once the depth value reaches a threshold close to a minimum depth value of the lip area. While the initial minimum found may not be the tongue tip precisely, the estimate eventually converges towards this point by locating a new local minimum in the previously found area of interest. This area is is marked by points surrounding previously found minimum point. Sometimes it can happen that there are multiple points in the region of interest which have the same depth value. If that is the case then a center of these points is estimated by their bounding box and this center will be the next found minimum point. This allows for more controlled estimate of the found minimum if a larger tongue area is protruded in the current frame. Tracking point gets reset if the tip estimation reaches set mouth bounds or mouth gets detected as closed.

There are several parameters to consider when setting up the tracking algorithm. If the search area between the last located point is too small, the location may get stuck in a local minimum. If the search window is too big, the location may converge towards a lip point closer to the depth camera instead of following the tongue. If either of these cases should occur, the tracking should reset in order to try to repair itself. For this purpose it's also important to define bounds of the mouth where the search for a local minimum can still happen.

4.5 Exercise description and key gestures

For this thesis I chose a subset of exercises which were shown to me by a speech therapy expert. Most of the exercises focus on tongue movements. A description as well as a sequence of steps and key points of tracking are described in the table 4.1.

The tongue is found in a key position based on the closest lip point in the $[x, y]$ coordinate space. If the found minimum depth is close enough to the lip point based on a selected threshold, the tongue is set to cover this lip point. Special cases are the mouth corners which are usually located slightly further away from the depth sensor than the rest of the lips. In order for the detected position to be the mouth corner, the area around each of the corners is checked. If the found minimum is in a proximity of a corner and the mean depth of points around the corner is less than the mean depth of points around the other mouth corner, it is assumed that the mouth corner is covered by the tongue.

4.6 Exercise evaluation workflow

Exercises are defined as key gestures performed in a sequence. While the full sequence is not detected, exercise should not be validated as successfully

Table 4.1: Descriptions and key gestures of selected speech therapy exercises

Exercise ID	**Description**	**Gesture sequence**
1	The user slowly moves tongue across the top palate to the upper lip and back again.	Tongue out, tongue top center, tongue out
2	The user slowly licks their lower lip from right to left and back.	Tongue right corner, tongue bottom center, tongue left corner, tongue bottom center, tongue right corner
3	The user slowly licks their upper lip from right to left and back. Tongue tip is pointed towards the nose.	Tongue right corner, tongue top center, tongue left corner, tongue top center, tongue right corner
4	The user slowly licks their lips in circular motion.	Tongue right corner, tongue top center, tongue left corner, tongue bottom center, tongue right corner
5	The user has their mouth opened and moves tongue from left corner to the right corner without moving the jaw.	Tongue right corner, tongue out, tongue left corner

performed. Additionally, some gestures within the exercise can be more important and should therefore be performed for a longer period of time. Example of this is the exercise with ID 1 which was described in table 4.1. The key gesture in that case is *tongue top center*. In order to enforce the importance of this gesture it should be held in correct position for a longer period of time compared to the other gestures within the sequence.

After a tongue gesture is identified, the current sequence state is updated. Once the correct gesture was performed for the defined count, the active gesture to evaluate changes to a following gesture of a sequence. The exercise can be enforced to be repeated multiple times so while this count is not achieved, the sequence simply restarts itself upon completion.

4.7 Implementation

All of the code for the application was written in Python 3.6. The main libraries which I used to load and process the data from the depth camera were *OpenCV* and *Pyrealsense2*. Additional packages were based on the requirements of facial landmark detection model which is implemented using

PyTorch framework. Operations with vectors and matrices were implemented using the *Numpy* package. The complete list of packages which were needed to be installed on top of a clean Python environment were:

- Pyrealsense2 v2.40.0
- OpenCV v4.1.0
- PyTorch v1.7.1 (tested with versions for *CUDA v10.1* and *CPU only*)
- face-alignment v1.3.3

While there are not any specific OS restrictions listed with these packages, I only tested the application on Linux.

4.7.1 Input data description

I have been working with the resolution of 640×480 pixels and frame rate of 30 FPS from both depth and RGB sensor which proved to be sufficient settings for the proof of concept. Pyrealsense2 provides a wrapper for many functionalities to process incoming camera streams. The stream can be accessed both by using pre-recorded *.bag* files and using data coming from the camera directly. When recording data through the Intel RealSense Viewer application the recording preset needs to be changed to *High Accuracy*. These presets are prepared by Intel and they are optimized for the device type and resolution. Alternatively these presets can be downloaded in a JSON format [75]. The relevant file I have used in my application is the `MidResHighAccuracyPreset.json`.

I have used post-processing filters for the depth data in order to enhance the incoming depth image stream. The importance of this step can be seen in the figure 4.2. The visible holes in the raw image would drastically change the outcomes of the detection process but it is remedied by the use of filters. The filters which I am using are *depth to disparity*, *temporal filter*, *hole-filling filter* and *disparity to depth* in their default setting.

The data captured by the RealSense camera device can be further transformed with Pyrealsense2's library functions in order to used seamlessly by the Numpy package. While the image data are defined as 3-channel matrices as one would expect, the depth data are instead given as a 2D matrix where each value is represented by depth value of the point. In order to get the actual metric distance between camera and a point, the data needs to be scaled by a constant depth scale parameter which can be accessed by a library function.

4.7.2 Core of the application

The script `rs_fan_pipeline.py` contains the main logic of the application. It can be run in two different modes. It either uses realtime data pulled from the D415 camera which needs to be plugged in the target device during runtime,

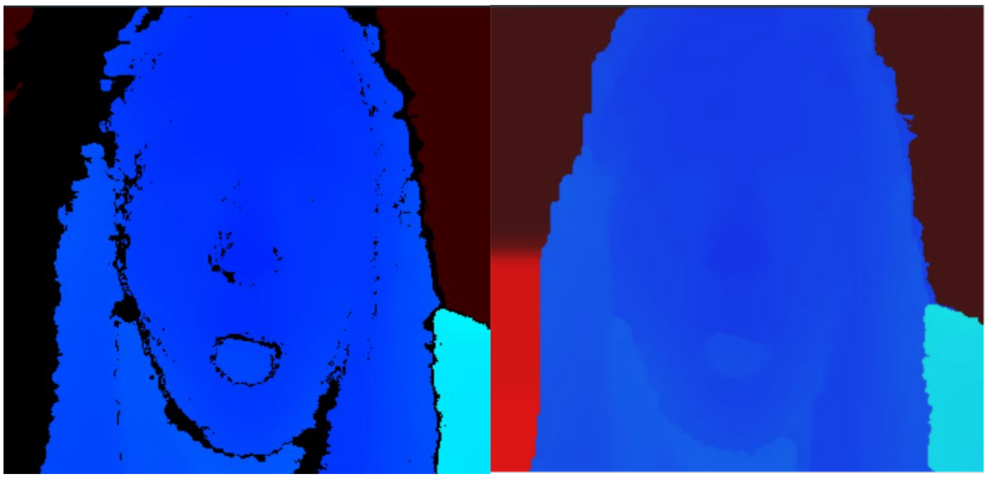


Figure 4.2: Comparison of raw depth image data and depth image after post-processing using filters

or it can load a compressed *.bag* file which was recorded through the *Intel RealSense Viewer* application.

There are several parameters to specify when running the script:

`--realtime` This flag indicates that the input will be taken directly from a camera plugged in the computer.

`--visible` This flag indicates that the output of the application will be rendered in a separate window on top of being saved to a video file. The stream can be closed by pressing the *esc* key or the *q* key on the keyboard.

`--input` ***string*** This parameter expects a string specifying the path to a *.bag* file which will be used as an input. If the realtime mode is toggled on then this input parameter is ignored.

`--exercise` ***int*** This parameter specifies which exercise will be evaluated during execution. If no exercise ID is specified or the ID is not valid then no exercise will be evaluated and the application will only track movement and predict gestures.

`--reps` ***int*** This parameter specifies how many times should the exercise be performed.

`--winsize` ***int*** This parameter defines size of the search window when tracking tongue based on previous minimum location. The figure 4.3 shows how this parameter helps define the search area.

`--tdtol` ***float*** This parameter defines how big of a depth value difference between the minimum point in the oral cavity and a weighted minimum lip point there has to be in order to start tracking tongue in the oral cavity. If the minimum values are not within the specified threshold then the detected tongue gesture is *tongue_in*.

`--mctol` ***float*** This parameter defines how big of a depth value difference between points around mouth corners there needs to be to detect the tongue covering one of the corners. If the minimum values are within this specified tolerance then tongue cannot be detected as covering mouth corner. The smaller the parameter value, the greater the lenience when detecting tongue covering mouth corner gesture.

`--tcscale` ***float*** This parameter defines a scale of depth tolerance when determining if tongue covers a specific lip point. The higher this value is, the more likely will the tongue be classified as covering a lip. Moreover the smaller this value is, the more likely the detected gesture will result in *tongue_out*.

`--lcscale` ***float*** This parameter defines how much will the lip area be scaled by. This defines how far can the minimum point be tracked up the mouth before the tracking gets reset and a new search will happen. If the parameter value is too small, the tracking will get reset more often. If the parameter value is too large, it can happen that the tracking will follow a lip point rather than the tongue.

4.7.3 Utilities

The script `func_utils.py` provides important functions for supporting main application logic. There are four main areas of functionalities that are implemented in this script:

- Calculating mouth features such as width and height for different parts
- Finding minimum location either as initial search or around the previously found minimum area
- Evaluating tongue coverage of different lip parts
- Lip and mouth mask operations

Mask operations are especially useful when locating minimum. By using a mask instead of a regular bounding box I have much better control over which points are to be considered to be in an area of interest.

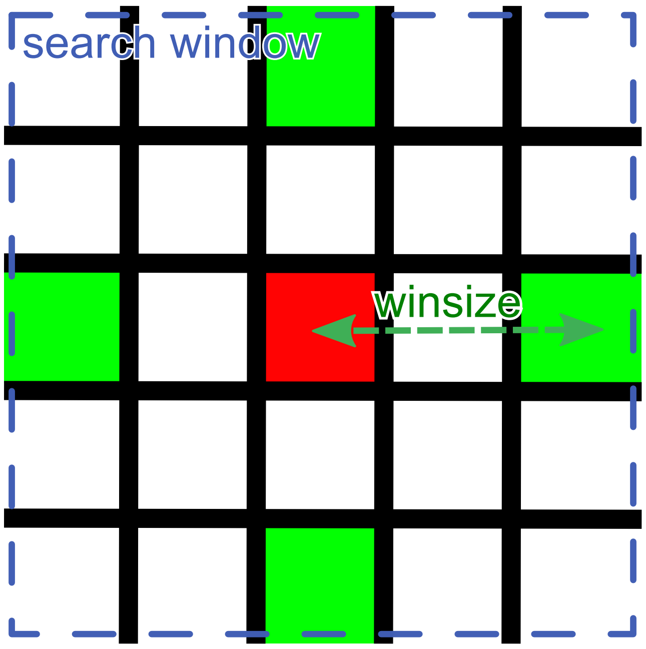


Figure 4.3: The `winsize` parameter value and how it affects the search bounding box around the previously found minimum value (marked as the red pixel)

4.7.4 Exercise definition

Exercises are defined as a constant using a Python dictionary. Each dictionary key marks the index of an exercise step. The value is comprised of a list which includes exercise step gesture name and a number of frames marking the length of performing this gesture. This allows for defining important parts of the exercise which should be performed for a longer period of time. It also reduces the risk of gesture misclassification affecting the proper exercise performance. All exercises from table 4.1 were defined in a file `constants.py`. Some of the exercise gesture sequences were simplified for the experimental evaluation which will be further discussed in the subsequent chapter 5.

Chapter 5

Experiments and evaluation

For the evaluation of the design I described in the previous chapter I have used recordings of several individuals performing the speech therapy exercises with the selected depth camera. The recordings for each of the exercises were created using the same settings, however they were not necessarily taken in the same conditions described in section 4.1. I decided to focus in greater detail on exercises 1 and 5 which are the simpler ones in terms of gesture sequence complexity. While these gestures are also relatively easy for the users to perform without any expert guidance, the outputs have shown that there was a lot of room for individual variety in how the exercises were performed. This kind of user input helps to uncover different weaker points of the designed solution as well as the important aspect of how important is the correct setup of the user's environment while using the application. For each of these exercises I am describing the used input data as well as individual parameters set for processing these recordings using my application in order to get the most exercise repetitions evaluated.

All experiments were performed on the same machine[2] which was used for benchmarking landmark detectors. The evaluation metrics and plots were computed using the *scikit-learn* library. The frame rate for each of the input videos was around 19 to 22 FPS when running without visible the output window.

5.1 Exercise 1

During this exercise the goal is to move the tongue tip slowly across the top palate inside the mouth and continuing this movement towards the top lip which should get fully covered by the tongue. The sequence of gestures is shown in the figure 5.1.

[2]See 3.1.4

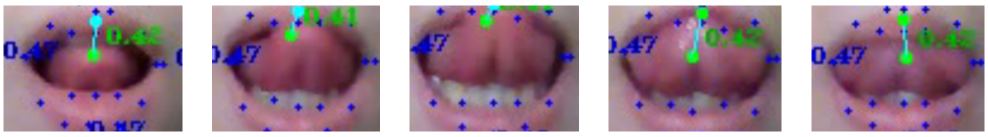


Figure 5.1: Sample sequence of exercise 1

This exercise should by its definition include the tongue moving inside the oral cavity. However, this kind of movement cannot be reasonably tracked with the depth camera, so the gesture sequence had to be simplified to not include the *tongue_in* gesture. As stated in the previous chapter, the key gesture in this sequence is *tongue_top_center* which is why in the exercise evaluation and the labeling of generated image data I paid a greater attention to the tongue being placed as high as possible over the top lip.

5.1.1 Description of input videos

Adult 1 Approximately 47 cm away from the camera. Head was facing the camera. Performed 4 repetitions. Mouth was opened wide during tongue protrusion. Closest to the exercise description.[3] Parameters: `--winsize` 2, `--tdtol` 7.5×10^{-3}, `--mctol` 2.0, `--tcscale` 0.225, `--lcscale` 0.65

Adult 2 Approximately 50 cm away from the camera. Performed 2 repetitions. Head was slightly tilted so that right mouth corner was closer to the camera than the left mouth corner (approximately 0.5 cm). Lips were very close together during the tongue protrusion. Parameters: `--winsize` 2, `--tdtol` 2.5×10^{-2}, `--mctol` 1.0, `--tcscale` 0.1, `--lcscale` 0.7

Adult 3 Approximately 35 cm away from the camera. Performed 5 repetitions. Head was facing the camera. On first repetition the lips were close together when protruding tongue. For the rest of the repetitions the lips were further apart. Parameters: `--winsize` 2, `--tdtol` 7.5×10^{-3}, `--mctol` 1.0, `--tcscale` 0.175, `--lcscale` 0.5

Child 1 Approximately 40 cm away from the camera. Performed 4 repetitions. Head was tilted so that right mouth corner was closer to the camera than the left mouth corner (approximately 1 cm). Tongue movement was very fast. Tongue was only moving between mouth center and the top lip, the movement never went in and out the oral cavity. Parameters: `--winsize` 2, `--tdtol` 8.5×10^{-3}, `--mctol` 3.0, `--tcscale` 0.17, `--lcscale` 0.55

[3]Link to the evaluated video: `https://youtu.be/PMxhPDdGhBE`

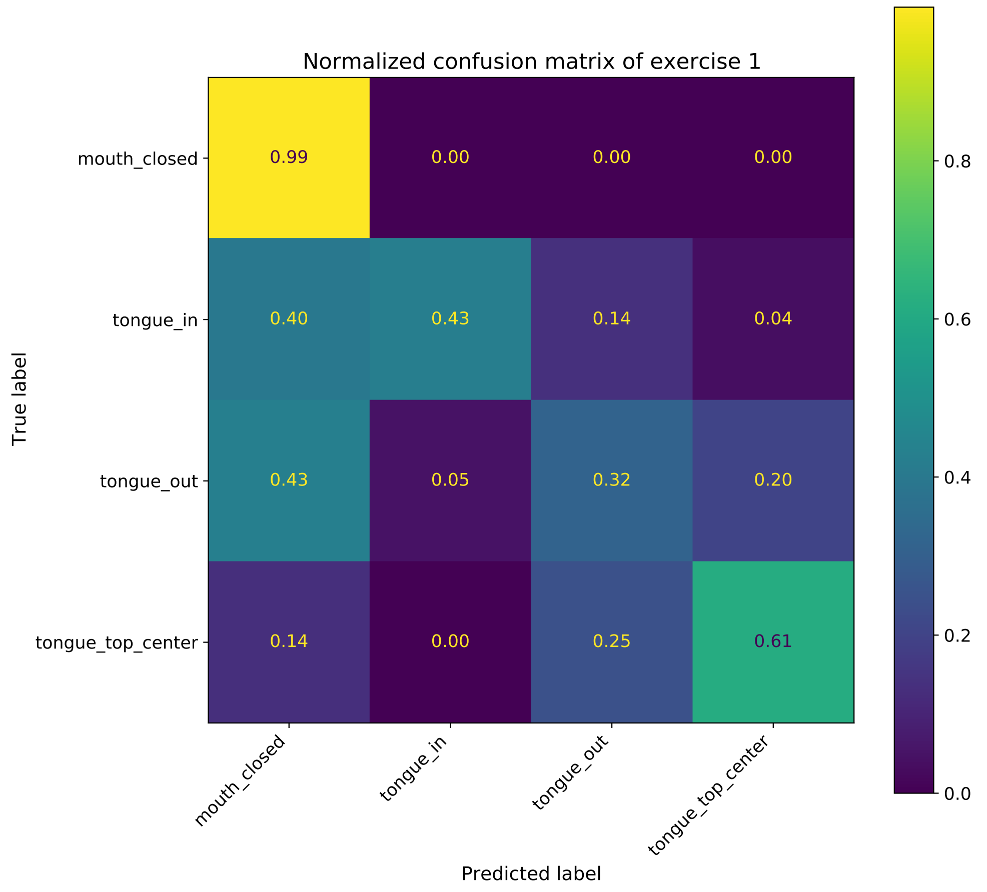


Figure 5.2: Normalized confusion matrix of gestures in exercise 1

Child 2 Approximately 35 cm away from the camera. Performed 4 repetitions. Head was facing the camera. During one repetition the lips were close together while the tongue was protruded. Parameters: `--winsize` 2, `--tdtol` 1.0×10^{-2}, `--mctol` 1.0, `--tcscale` 0.2, `--lcscale` 0.6

Child 3 Approximately 32 cm away from the camera. Performed 2 repetitions. A lot of head movement during the exercise. Lips were very close together during the tongue protrusion, mouth was not opening at all during the exercise. Parameters: `--winsize` 2, `--tdtol` 2.5×10^{-2}, `--mctol` 3.5, `--tcscale` 0.15, `--lcscale` 0.7

5.1.2 Evaluation of gesture classification

The confusion matrix shown in the figure 5.2 contains true and predicted gestures for all of the input videos.

From this data I have calculated the following quality metrics:

Table 5.1: Results of evaluation of exercise 1

Video Subject	**Repetitions made**	**Repetitions detected**
Adult 1	4	4
Adult 2	2	1
Adult 3	5	4
Child 1	4	3
Child 2	4	3
Child 3	2	1

- precision = 0.7
- recall = 0.62
- F1 score = 0.62
- Hamming loss = 0.38

5.1.3 Evaluation of exercise sequence detection

The table 5.1 describes how many repetitions of the exercise were evaluated as successful based on each of the input videos. The percentage of total detected repetitions out of all performed was **76%** with no prematurely evaluated repetitions.

5.1.4 Experiment conclusion

The biggest problem with evaluating this exercise was with the different approach the users were taking while performing it. While the algorithm was able to track tongue movement when the mouth was open as much as possible, it had problems detecting movement when the lips were very close together as the tracking sequence does not trigger when the mouth is detected as closed. The figure 5.3 shows the manifestation of this problem. This has been a problem during testing of the other exercises as well. The logic of the algorithm in this case should stay as is, but a further incentive for the users is needed so they know how much should they open their mouth or warn them when they keep the lips too close together.

Other weaker point of the exercise evaluation happened while the tongue was supposed to move from the top lip back in the oral cavity. For the majority of users this movement was much more subtle and faster than moving the tongue tip upwards. That was usually the cause of why the sequence was not registered fully or with a considerable delay.

For future research it would be needed to think more about the sequence of this exercise. The sequence could be even more strict and enforce the user to close the mouth entirely after finishing the repetition. This would allow to

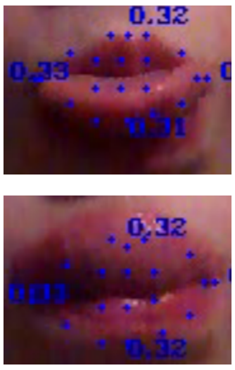


Figure 5.3: Problems with gesture detection during exercise 1

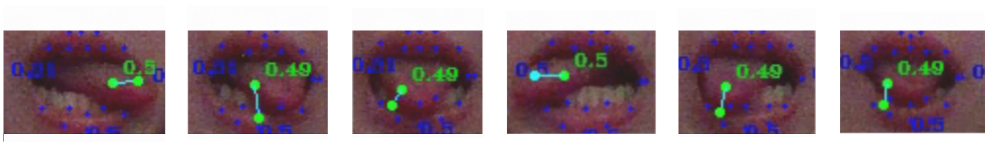


Figure 5.4: Sample sequence of exercise 5

separate the start and end of the exercise and potentially increase the exercise detection.

5.2 Exercise 5

During this exercise the user moves their tongue from left mouth corner to the right while keeping their jaw in a fixed position. It is not required for the tongue to hold the corner position for a prolonged amount of time but the corners should be considered the key gestures in this case and therefore have a bigger priority over the movement which happens between the two corners.

Different users started from different mouth corners. For evaluation the exercise repetition was counted when they moved their tongues from the left corner to the right corner and then made a move from the right corner towards the left corner again. The figure 5.4 shows how the sequence should look like.

5.2.1 Description of input videos

Adult 1 Approximately 47 cm away from the camera. Head was facing the camera. Performed 4 repetitions starting from left mouth corner. Mouth was opened during tongue protrusion. The tongue rarely touched the lips during the transition between mouth corners. Closest to the exercise description.[4] Parameters: `--winsize` 2, `--tdtol` 9.0×10^{-3}, `--mctol` 0.25, `--tcscale` 0.25, `--lcscale` 0.7

[4]Link to the evaluated video: `https://youtu.be/p5iINEdr3Z4`

Adult 2 Approximately 48 cm away from the camera. Performed 3 repetitions. Head was tilted so that left mouth corner was closer to the camera than the right mouth corner (approximately 1 cm). Mouth was opened during the tongue protrusion. Tongue was moving in a slow pace. Tongue was overlapping the bottom lip slightly when moving between mouth corners. Parameters: `--winsize` 3, `--tdtol` 1.0×10^{-2}, `--mctol` 0.1, `--tcscale` 0.1, `--lcscale` 0.7

Adult 3 Approximately 40 cm away from the camera. Performed 3 repetitions. Head was facing the camera initially but it was moving during the exercise and got tilted so that the right corner was closer to the camera than the left mouth corner (approximately 1 cm). Mouth was opened during the tongue protrusion. Tongue was moving at a rapid pace. Tongue did not touch lips between movements from corner to corner. Parameters: `--winsize` 6, `--tdtol` 1.5×10^{-2}, `--mctol` 0.1, `--tcscale` 0.3, `--lcscale` 0.7

Child 1 Approximately 33 cm away from the camera. Performed 10 repetitions. Head was facing the camera. Mouth was opened a little just so that the tongue's border was close to the lips. Tongue movement was very fast and mouth corners were not touched for a longer period of time. Tongue was not touching the lips while moving from corner to corner. Parameters: `--winsize` 4, `--tdtol` 5.0×10^{-2}, `--mctol` 0.01, `--tcscale` 0.575, `--lcscale` 0.7

Child 2 Approximately 33 cm away from the camera. Performed 10 repetitions. Head was slightly tilted so that the right mouth corner was closer to the camera than the left mouth corner (approximately 0.5 cm). Mouth was opened wide. Tongue was overlapping the bottom lip during the movement between mouth corners. Tongue movement was very fast. Parameters: `--winsize` 5, `--tdtol` 2.5×10^{-2}, `--mctol` 2.5×10^{-2}, `--tcscale` 0.1, `--lcscale` 0.65

5.2.2 Evaluation of gesture classification

The confusion matrix shown in the figure 5.5 contains true and predicted gestures for all of the input videos.

From this data I have calculated the following quality metrics:

- precision = 0.58
- recall = 0.28
- F1 score = 0.33
- Hamming loss = 0.72

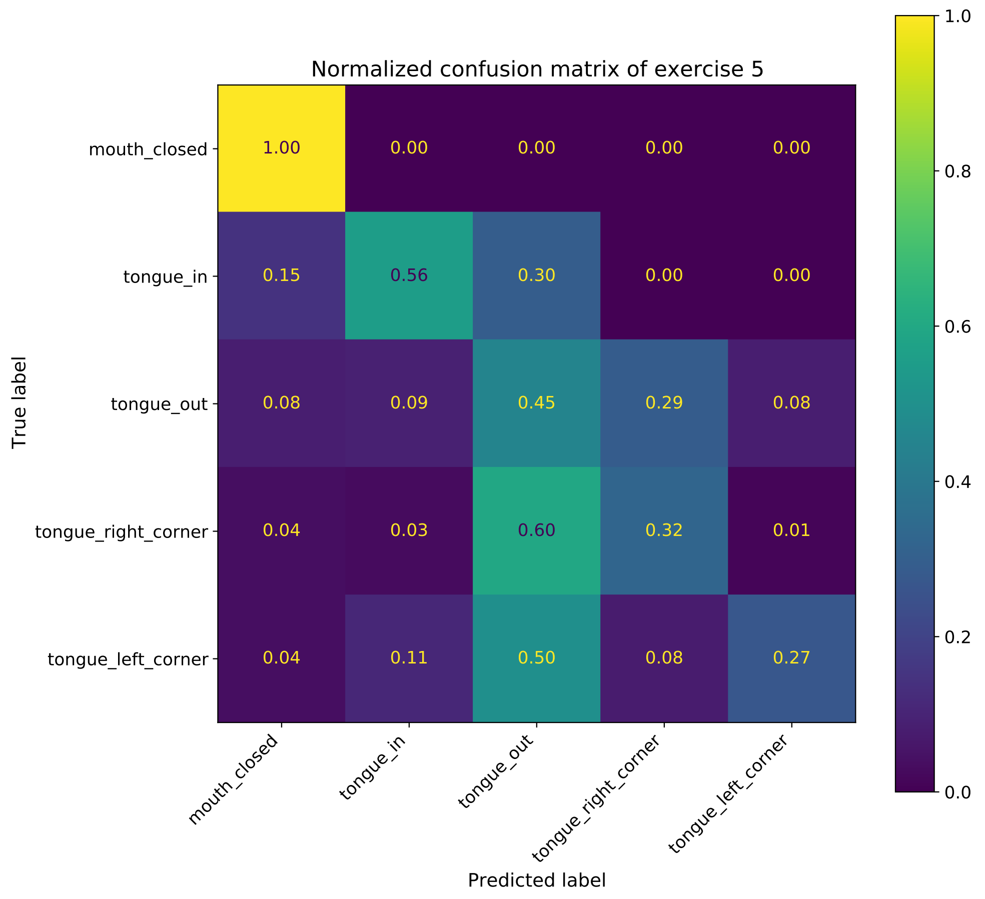


Figure 5.5: Normalized confusion matrix of gestures in exercise 5

Table 5.2: Results of evaluation of exercise 5

Video Subject	**Repetitions made**	**Repetitions detected**
Adult 1	4	4
Adult 2	3	2
Adult 3	3	2
Child 1	10	2
Child 2	10	5

5.2.3 Evaluation of exercise sequence detection

The table 5.2 describes how many repetitions of the exercise were evaluated as successful based on each of the input videos. The percentage of total detected repetitions out of all was **53%** with no prematurely evaluated repetitions.

5.2.4 Experiment conclusion

For experimental purposes it was not checked whether the users moved their jaw in any way during the performance. In the future this would need to be actively checked by the application during runtime according to the location of jaw landmarks.

This exercise has proved very sensitive in cases when the user's head was tilted in one direction. Even a 1 cm difference between corners' depths often resulted in delayed detection of the corner gesture or gesture not getting detected at all. In the future this problem would have to be dealt with either by guiding users how to use the application properly or by find a way of compensating for the bigger depth values based on the head tilt.

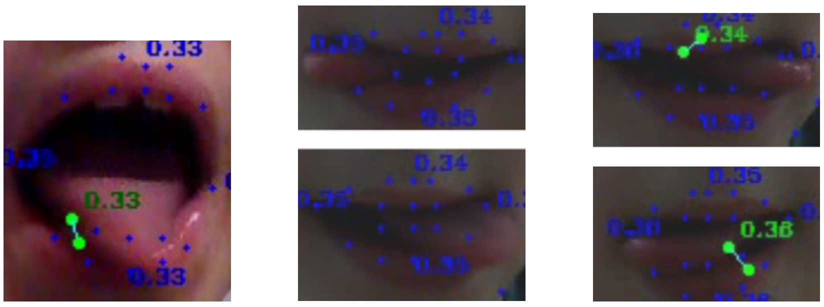
Another problematic aspect was speed of the tongue movement. It was possible to track fast tongue movement by increasing the `winsize` parameter but if the user did not have fully opened mouth this often broke the sequence as the minimum point got tracked towards the lip where depth values were much smaller than in the tongue area. While this could have been balanced by reducing the `lcscale` parameter, the tracking would loose a lot of valuable information of user's tongue movement. Combined with the tilt of the head this proved to be a big problem for detecting the whole exercise sequence.

In this exercise it is expected that the gesture being detected during movement in between mouth corners would be *tongue_out*. In theory this would work best with user pointing the tongue tip forward in order not to overlap the top or bottom lip. In practice the users often pointed their tongue tip downwards which resulted in detecting the gesture *tongue_bottom_center* instead. For the algorithm it was the logical outcome as the detected minimum depth of oral cavity was technically close to the depth of the bottom lip by a selected threshold. The potential fix of the logic would be to try to estimate if the tongue truly covers the lip physically or only hovers over it. This could prove difficult to implement due to different thicknesses of tongues as well as the individual users' ability to protrude tongue forward. Another way of going about solving this particular problem would be to separate key gestures and the movement in between. This would allow for different gestures being counted as a part of the sequence while keeping the important gestures enforced. It would need to be consulted with speech therapists in order not to make the system too lenient to be helpful for the users.

The figure 5.6 shows typical problems of misdetection happening during evaluation of exercise 5.

5.3 Future improvements

This application still has areas for improvement before being released to the general public. I have analyzed several points of improvement for the current application logic in the dedicated subsections of each experiment. In this



(a) Tongue hovering over lip during movement

(b) Lips too close together

(c) Head tilting

Figure 5.6: Problems with gesture evaluation of exercise 5

section I would focus more on different areas of improvement, most notably related to user experience.

While this working prototype is able to track and evaluate speech therapy exercises, the application would benefit from being easier to use. At this stage it is needed to set up multiple parameters individually for the best possible evaluation performance. This could be changed by implementing machine learning methods to either help set up the optimal parameters automatically or train a classifier to evaluate tongue gesture state based on the mouth features and depth map.

One possible approach for increasing the evaluation accuracy would be to properly enforce how a user behaves while using the application, for example how is their head tilted or how quickly they perform the exercise. This could be a challenge from the user interface perspective and it would require a further cooperation with speech therapists. The benefit would be that the users would be lead towards doing the exercises in the proper way.

An interesting addition for the application would be to add an augmented reality component to make the exercises more fun and enjoyable especially for younger children.

Conclusion

The goal of this thesis was to design and implement software for detecting facial landmarks to support an automated speech therapy application. The biggest and most challenging aspect of this application was the detection and tracking of tongue movement.

I have conducted a research of existing solutions and technologies which were used to solve similar use cases. The analysis of the positives and negatives of various approaches has led me to design an application which leverages the Intel RealSense D415 depth camera.

I have performed an experimental evaluation of the design on two selected speech therapy exercises. Despite the lower exercise sequence detection which in most cases was caused by head tilting, rapid tongue movement and mouth being opened too little, the depth camera has proven to be a reliable method for tongue tracking.

With the cooperation and support of SIT Port in Pilsen, where the user interface of the application is under active development, I will work further on improving the current solution to be more reliable and easier to use. The end goal is to release the application to the general public and enable users to practice different speech therapy exercises without active supervision of a speech therapy expert.

Bibliography

[1] Sonka, M.; Hlavac, V.; et al. *Image Processing, Analysis and Machine Vision.* Boston, MA: Springer US, 1993, ISBN 0412455706;1489932178;9780412455704;9781489932174;.

[2] Jiřina, M.; Novák, J.; et al. Strojové vidění a zpracování obrazu (BI-SVZ): Strojové vidění a fyzikální podstata. `https://courses.fit.cvut.cz/BI-SVZ/lectures/files/bi-svz-01-strojove-videni-a-fyzikalni-podstata.pdf`, 2020, [Accessed: 2020–07-12].

[3] Sýkora, D. Digital Image Processing: Lecture 1 – Monadic operations. `https://dcgi.fel.cvut.cz/home/sykorad/dzo/slides/dzo-l01.pdf`, 2020, [Accessed: 2020–05-01].

[4] Sýkora, D. Digital Image Processing: Lecture 3 – Convolution. `https://dcgi.fel.cvut.cz/home/sykorad/dzo/slides/dzo-l03.pdf`, 2020, [Accessed: 2020–05-01].

[5] Dey, S. *Hands-On Image Processing with Python: Expert Techniques for Advanced Image Analysis and Effective Interpretation of Image Data.* Birmingham: Packt Publishing, Limited, 2018, ISBN 1789343739;9781789343731;.

[6] Mehrotra, R.; Namuduri, K.; et al. Gabor filter-based edge detection. *Pattern Recognition*, volume 25, no. 12, 1992: pp. 1479 – 1494, ISSN 0031-3203, doi:https://doi.org/10.1016/0031-3203(92)90121-X. Available from: `http://www.sciencedirect.com/science/article/pii/003132039290121X`

[7] Shah, A. Through the eyes of Gabor filter. Medium.com [online], 2018, [Accessed: 2020-10-11]. Available from: `https://medium.com/@anuj_shah/through-the-eyes-of-gabor-filter-17d1fdb3ac97`

[8] Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, Springer, 2003, pp. 363–370.

[9] Kubat, M. *An Introduction to Machine Learning*. Cham: Springer International Publishing AG, second edition, 2017, ISBN 9783319639123;3319639129;.

[10] Habibi Aghdam, H.; Jahani Heravi, E. *Guide to Convolutional Neural Networks: A Practical Application to Traffic-Sign Detection and Classification*. Cham: Springer International Publishing AG, 2017, ISBN 9783319575490;331957549X;.

[11] Newell, A.; Yang, K.; et al. Stacked Hourglass Networks for Human Pose Estimation. In *Computer Vision – ECCV 2016*, edited by B. Leibe; J. Matas; N. Sebe; M. Welling, Cham: Springer International Publishing, 2016, ISBN 978-3-319-46484-8, pp. 483–499.

[12] Tensorflow team. TensorFlow: An end-to-end open source machine learning platform. Tensorflow.org [online], [Accessed: 2020-10-05]. Available from: `https://www.tensorflow.org/`

[13] PyTorch team. PyTorch: From research to production. Pytorch.org [online], [Accessed: 2020-10-05]. Available from: `https://pytorch.org/`

[14] Wisdom, B. D. Understanding the Confusion Matrix. Dev.to [online], [Accessed: 2020-12-10]. Available from: `https://dev.to/overrideveloper/understanding-the-confusion-matrix-2dk8`

[15] Wisdom, B. D. Understanding the Confusion Matrix (II). Dev.to [online], [Accessed: 2020-12-10]. Available from: `https://dev.to/overrideveloper/understanding-the-confusion-matrix-264i`

[16] Scikit-learn team. Metrics and scoring: quantifying the quality of predictions. Scikit-Learn.org [online], [Accessed: 2020-12-10]. Available from: `https://scikit-learn.org/stable/modules/model_evaluation.html#hamming-loss`

[17] Cyganek, B.; Books24x7, I. *Object detection and recognition in digital images: theory and practice*. Somerset: WILEY, 2013, ISBN 0470976373;9780470976371;9781118618370;1118618378;.

[18] Hjelmås, E.; Low, B. K. Face Detection: A Survey. *Computer vision and image understanding*, volume 83, no. 3, 2001: pp. 236–274.

[19] Viola, P.; Jones, M. Rapid Object Detection using a Boosted Cascade of Simple Features. 02 2001, ISBN 0-7695-1272-0, pp. I–511, doi:10.1109/CVPR.2001.990517.

[20] Itseez. Open Source Computer Vision Library. `https://github.com/itseez/opencv`, 2015.

[21] Dalal, N.; Triggs, B. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, 2005, pp. 886–893 vol. 1, doi:10.1109/CVPR.2005.177.

[22] King, D. E. Dlib-ml: A Machine Learning Toolkit. *Journal of Machine Learning Research*, volume 10, 2009: pp. 1755–1758.

[23] Liu, W.; Anguelov, D.; et al. SSD: Single Shot MultiBox Detector. *Lecture Notes in Computer Science*, 2016: p. 21–37, ISSN 1611-3349, doi:10.1007/978-3-319-46448-0_2. Available from: `http://dx.doi.org/10.1007/978-3-319-46448-0_2`

[24] Minichino, J.; Howse, J. *Learning OpenCV 3 Computer Vision with Python - Second Edition.*, *Community Experience Distilled*, volume Second edition. Packt Publishing, 2015, ISBN 9781785283840. Available from: `http://search.ebscohost.com.ezproxy.techlib.cz/login.aspx?direct=true&db=e000xww&AN=1074862&lang=cs&site=ehost-live`

[25] Tensorflow development team. Blazeface detector. GitHub.com [online], [Acessed: 2020-10-19]. Available from: `https://github.com/tensorflow/tfjs-models/tree/master/blazeface`

[26] datahacker.rs. #010 How to align faces with OpenCV in Python. DataHacker.rs [online], 2020, [Accessed: 2020-11-05]. Available from: `http://datahacker.rs/010-how-to-align-faces-with-opencv-in-python/`

[27] Celiktutan, O.; Ulukaya, S.; et al. A comparative study of face landmarking techniques. *EURASIP Journal on Image and Video Processing*, volume 2013, 03 2013, doi:10.1186/1687-5281-2013-13.

[28] Wu, Y.; Ji, Q. Facial Landmark Detection: a Literature Survey. *CoRR*, volume abs/1805.05563, 2018, 1805.05563. Available from: `http://arxiv.org/abs/1805.05563`

[29] Chateau, T.; Duffner, S.; et al. A survey of deep facial landmark detection. 06 2018.

[30] Intelligent Behaviour Understanding Group (iBUG). Facial point annotation. `https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/`, [Accessed: 2020-10-01].

[31] vigneshsuresh4499. OpenCV – Facial Landmarks and Face Detection using dlib and OpenCV. GeeksForGeeks.org [online], 2020, [Accessed: 2020-10-10]. Available from: `https://www.geeksforgeeks.org/opencv-facial-landmarks-and-face-detection-using-dlib-and-opencv/`

[32] Le, V.; Brandt, J.; et al. Interactive Facial Feature Localization. In *Computer Vision – ECCV 2012*, edited by A. Fitzgibbon; S. Lazebnik; P. Perona; Y. Sato; C. Schmid, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33712-3, pp. 679–692.

[33] Belhumeur, P. N.; Jacobs, D. W.; et al. Localizing Parts of Faces Using a Consensus of Exemplars. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 35, no. 12, 2013: pp. 2930–2940, doi: 10.1109/TPAMI.2013.23.

[34] Sagonas, C.; Antonakos, E.; et al. 300 Faces In-The-Wild Challenge: database and results. *Image and Vision Computing*, volume 47, 2016: pp. 3 – 18, ISSN 0262-8856, doi:https://doi.org/10.1016/j.imavis.2016.01.002, 300-W, the First Automatic Facial Landmark Detection in-the-Wild Challenge. Available from: `http://www.sciencedirect.com/science/article/pii/S0262885616000147`

[35] Zhu, X.; Lei, Z.; et al. Face Alignment Across Large Poses: A 3D Solution. *CoRR*, volume abs/1511.07212, 2015, `1511.07212`. Available from: `http://arxiv.org/abs/1511.07212`

[36] Cootes, T.; Taylor, C.; et al. Active Shape Models-Their Training and Application. *Computer Vision and Image Understanding*, volume 61, no. 1, 1995: pp. 38 – 59, ISSN 1077-3142, doi:https://doi.org/10.1006/cviu.1995.1004. Available from: `http://www.sciencedirect.com/science/article/pii/S1077314285710041`

[37] Cootes, T. F.; Edwards, G. J.; et al. Active appearance models. In *Computer Vision — ECCV'98*, edited by H. Burkhardt; B. Neumann, Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, ISBN 978-3-540-69235-5, pp. 484–498.

[38] Menpo development team. Source code for menpofit.aam.pretrained. `https://menpofit.readthedocs.io/en/stable/_modules/menpofit/aam/pretrained.html`, [Accessed: 2020-10-01].

[39] Dlib development team. Face Landmark detection example code. dlib.net [online], [Accessed: 2020-10-01]. Available from: `http://dlib.net/face_landmark_detection_ex.cpp.html`

[40] Kazemi, V.; Sullivan, J. One millisecond face alignment with an ensemble of regression trees. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1867–1874, doi:10.1109/CVPR.2014.241.

[41] Zhang, K.; Zhang, Z.; et al. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks. *CoRR*, volume abs/1604.02878, 2016, 1604.02878. Available from: http://arxiv.org/abs/1604.02878

[42] Kowalski, M.; Naruniec, J.; et al. Deep Alignment Network: A convolutional neural network for robust face alignment. *CoRR*, volume abs/1706.01789, 2017, 1706.01789. Available from: http://arxiv.org/abs/1706.01789

[43] Bulat, A.; Tzimiropoulos, G. How far are we from solving the 2D & 3D Face Alignment problem? (and a dataset of 230,000 3D facial landmarks). In *International Conference on Computer Vision*, 2017.

[44] Bulat, A. Face Recognition. GitHub.com [online], [Accessed: 2020-10-10]. Available from: https://github.com/1adrianb/face-alignment

[45] Feng, Y.; Wu, F.; et al. Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network. In *ECCV*, 2018.

[46] Snider, L. Anatomy and Physiology: The Terrific Tongue. Visible Body Blog [online], 2019, [Accessed: 2020-11-05]. Available from: https://www.visiblebody.com/blog/anatomy-and-physiology-the-terrific-tongue

[47] Balamuralidhar, N.; Saksena, S. K.; et al. 3D Tongue tip tracking. GitHub.com [online], 2019, [Accessed: 2020-10-02]. Available from: https://github.com/dronefreak/3D-tongue-tip-tracking

[48] Hu, D.; Tsinghua University Press, T. U. *Traditional Chinese Medicine : Theory and Principles*. Berlin/Boston, GERMANY: De Gruyter, Inc., 2015, ISBN 9783110417661. Available from: http://ebookcentral.proquest.com/lib/techlib-ebooks/detail.action?docID=4338516

[49] Li, W.; Hu, S.; et al. Towards the objectification of tongue diagnosis: Automatic segmentation of tongue image. In *2009 35th Annual Conference of IEEE Industrial Electronics*, 2009, pp. 2121–2124, doi:10.1109/IECON.2009.5415334.

[50] Li, W.; Hu, S.; et al. The separation framework of tongue coating and proper in Traditional Chinese Medicine. In *2009 7th International Conference on Information, Communications and Signal Processing (ICICS)*, 2009, pp. 1–4, doi:10.1109/ICICS.2009.5397473.

[51] Królak, A.; Strumio, P. Eye-blink detection system for human-computer interaction. *Universal Access in the Information Society*, volume 11, no. 4, 11 2012: pp. 409–419, copyright -

Springer-Verlag Berlin Heidelberg 2012; Last update - 2012-12-10. Available from: `http://ezproxy.techlib.cz/login?url=https://www-proquest-com.ezproxy.techlib.cz/scholarly-journals/eye-blink-detection-system-human-computer/docview/1115061571/se-2?accountid=119841`

[52] Sapaico, R.; Laga, H.; et al. Detection of Tongue Protrusion Gestures from Video. *IEICE Transactions on Information and Systems*, volume E94D, 08 2011: pp. 1671–1682, doi:10.1587/transinf.E94.D.1671.

[53] Niu, S.; Liu, L.; et al. Tongue-able Interfaces: Prototyping and Evaluating Camera Based Tongue Gesture Input System. *Smart Health*, 03 2018: pp. –, doi:10.1016/j.smhl.2018.03.001.

[54] Suciu, D. M.; Pop, B. A.; et al. Non-intrusive Tongue Tracking and Its Applicability in Post-stroke Rehabilitation. In *On the Move to Meaningful Internet Systems: OTM 2014 Workshops*, edited by R. Meersman; H. Panetto; A. Mishra; R. Valencia-García; A. L. Soares; I. Ciuciu; F. Ferri; G. Weichhart; T. Moser; M. Bezzi; H. Chan, Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, ISBN 978-3-662-45550-0, pp. 504–513.

[55] Seppala, T. J. Microsoft isn't selling Kinect for Windows anymore. Engadget [online], 2015, [Accessed: 2020-10-02]. Available from: `https://www.engadget.com/2015-04-02-rip-kinect-for-windows.html`

[56] ASSISLT - Automatizovaný Softwarový Systém pro Logopedickou Terapii. TAČR Starfos [online], 2018, [Accessed: 2020-08-13]. Available from: `https://starfos.tacr.cz/cs/project/TJ01000181`

[57] Bílková, Z.; Novozámský, A.; et al. Automatic Evaluation of Speech Therapy Exercises Based on Image Data. In *Image Analysis and Recognition*, edited by F. Karray; A. Campilho; A. Yu, Cham: Springer International Publishing, 2019, ISBN 978-3-030-27202-9, pp. 397–404.

[58] Bílková, Z.; Novozámský, A.; et al. Human Computer Interface Based on Tongue and Lips Movements and its Application for Speech Therapy System. *Electronic Imaging*, volume 2020, no. 1, 2020: pp. 389–1–389–5, doi:doi:10.2352/ISSN.2470-1173.2020.1.VDA-389. Available from: `https://www.ingentaconnect.com/content/ist/ei/2020/00002020/00000001/art00007`

[59] de Paz, I. C. MTCNN. GitHub.com [online], [Accessed: 2020-09-20]. Available from: `https://github.com/ipazc/mtcnn`

[60] Kowalski, M. Deep Alignment Network. GitHub.com [online], [Accessed: 2020-09-20]. Available from: `https://github.com/MarekKowalski/DeepAlignmentNetwork`

[61] Feng, Y. Joint 3D Face Reconstruction and Dense Alignment with Position Map Regression Network. GitHub.com [online], [Accessed: 2020-09-25]. Available from: `https://github.com/YadiraF/PRNet`

[62] BioID. The BioID Face Database [dataset]. `https://www.bioid.com/facedb/`, [Accessed: 2020-09-30].

[63] Goldstein, O. Face Images with Marked Landmark Points [dataset]. `https://www.kaggle.com/drgilermo/face-images-with-marked-landmark-points`, 09 2017, [Accessed: 2020-09-30].

[64] Goldstein, O. Exploring the images and facial landmarks [online]. `https://www.kaggle.com/drgilermo/exploring-the-images-and-facial-landmarks`, 2017, [Accessed: 2020-09-30].

[65] Lai, S.; Chai, Z.; et al. Enhanced Normalized Mean Error loss for Robust Facial Landmark detection. In *BMVC*, 2019.

[66] Anaraki, J. R.; Orlandi, S.; et al. A Deep Learning Approach to Tongue Detection for Pediatric Population. 2020, 2009.02397.

[67] librealsense development team. Depth from Stereo. GitHub.com [online], [Accessed: 2020-10-15]. Available from: `https://github.com/IntelRealSense/librealsense/blob/master/doc/depth-from-stereo.md`

[68] Intel. Compare Cameras. IntelRealSense.com [online], [Accessed: 2020-10-15]. Available from: `https://www.intelrealsense.com/compare-depth-cameras/`

[69] Orbbec 3D Tech. Orbbec Online shop. Orbbec 3D Tech shop [online], [Accessed: 2020-10-15]. Available from: `https://shop.orbbec3d.com/`

[70] Orbbec 3D Tech. Astra Series. Orbbec 3D Tech [online], [Accessed: 2020-10-15]. Available from: `https://orbbec3d.com/product-astra-pro/`

[71] Intel. Intel® RealSense™ SDK 2.0 – Start building your own depth applications. IntelRealSense.com [online], [Accessed: 2020-10-15]. Available from: `https://www.intelrealsense.com/developers/`

[72] librealsense development team. RS Camera. Google Play [online], [Accessed: 2020-10-20]. Available from: `https://play.google.com/store/apps/details?id=com.intel.realsense.camera&hl=en_US&gl=US`

[73] Orbbec 3D Tech. Develop with Orbbec – Astra SDK. Orbbec 3D Tech [online], [Accessed: 2020-10-15]. Available from: `https://orbbec3d.com/develop/`

[74] Katzenbauer. Počítač Uživatele Osoba Stolní Strany Laboratoře Piktogram Lan. Vector.me [online], [Accessed: 2020-12-17]. Available from: https://cz.vector.me/browse/145993/computer_user_person_desktop_party_lab_pictogram_lan

[75] librealsense development team. D400 Series Visual Presets. GitHub.com [online], [Accessed: 2020-10-15]. Available from: https://github.com/IntelRealSense/librealsense/wiki/D400-Series-Visual-Presets

Appendix A

Acronyms

AAM	Active appearance model
ASM	Active shape model
CNN	Convolutional neural network
DAN	Deep Alignment Network
FAN	Face Alignment Network
FPS	Frames-per-second
HCI	Human computer interface
HOG	Histogram of oriented gradients
MTCNN	Multi-task Cascaded Convolutional Networks
NME	Normalized mean error
OF	Optical flow
OTG	USB On-the-go
PRNet	Position map regression network
SDK	Software development kit
SVM	Support-vector machine
TCM	Traditional Chinese medicine

Appendix B

Contents of enclosed CD

```
README.md ............ the file with CD contents description
src .................. the directory of implementation sources
  demos .............. the directory of sample demo notebooks
  outputs ............ the directory of sample output videos
text ................. the thesis text directory
  sources ............ the directory of LaTeX source codes of the thesis
  thesis.pdf ......... the thesis text in PDF format
```