Radboud University Nijmegen

AI Bachelor Thesis

Automatic detection of facial tics in Gilles de la Tourette Syndrome patients

Author:
Flip van Rijn
(s4050614)

Supervisors:
Dr. L.G. Vuurpijl
Prof. P.W.M. Desain

External contact HSK Group:
Drs. J. van de Griendt

SOW-BKI300 AI Bachelor Thesis
Artificial Intelligence
Faculty of Social Sciences
Radboud University Nijmegen
Januari 30, 2014

Abstract

This project pursues automatic facial tic detection in video recordings of Gilles de la Tourette syndrome patients. Current diagnostic methods using a severity measurement scales and tic counts require huge effort from the therapist and the patient. By automating these diagnostic methods, this would alleviate these efforts. A study by Chappell et al. shows that manually counting tics in videotape recordings is feasible[9]. Only few studies used artificial intelligence to automate the process of tic detection. A study by M. Bernabei is the most prominent one and shows that automatic tic detection with triaxial accelerometers is possible[3]. In the current project, distinctive features used by experts have been explored for their use in a system using computer vision and machine learning techniques. An implementation for this system is made and the performance is benchmarked. This project explored whether this approach was feasible and yields recommendations for further improvements. Finally, results and guidelines for the design of a system that uses automated tic detection for psychotherapeutic feedback and exercises are given.

1 Introduction

Patients with the Gilles de la Tourette Syndrome (GTS) have various kinds of tics. These tics are involuntary movements or sounds which are produced with their own independent severity. This means that some tics are more dominant than others. The GTS has a negative impact on the social life of a Tourette patient and can result in disrupted social functioning[15]. Not only the tics themselves can be a burden, also going to therapeutic sessions takes up a lot of time. During these sessions the patient is evaluated using a semi-structured interview whether the patient has a certain kind of tic and what the severity of that tic is. These sessions also include exercises to suppress tics for as long as possible.

In a study by Verdellen et al.[24] the authors concluded that behavioral therapy does have a significant effect on the number of tics before and after the therapy. During the therapy sessions, exposure and response prevention (ER) are applied. Typically the urge (or premonitory sensation) of a tic is followed by an involuntary movement after which the sensation decreases. Therefore, the involuntary movement is associated with the sensation and the goal of ER is to disconnect this association. With response prevention a GTS patient has to block the involuntary movements and thus experiences exposure of the sensation which ultimately results in habituation. As a result to this habituation the urge to produce these involuntary movements decreases and thus the number of tics decreases as well. An example is given of a subject which had 220 eye tics per ten minutes, whereas the follow-up (after nine weeks) showed only normal eye blink behavior. The treatment during the therapy sessions have to make the subject more aware of its tics, since the subject indicates that it does not see the tics coming. The next stage of the treatment is producing counter-movements which in this case means the subject has to widen its eyes. The subject practices with these counter-movements during the sessions and at home and the number of tics per ten minutes decrease to eight tics.

The severity of a tic, as R. Kurlan et al. define it, is a combination of multiple factors such as the amount, the frequency and intensity of the tics and the part of the body where the tics occur[12]. The environment also has an impact on these factors[11]. In order to rate the severity of a patient's disorder, there are multiple scales available that on their own tell how severe the disorder is. The three most commonly used scales are the Yale Global Tic Severity Scale (YGTSS), the Tourette Syndrome Global Scale (TSGS) and the Shapiro Tourette Syndrome Severity Scale (STSS). In a timespan of at least a week a patient is being interviewed and based on this information each scale gives a severity measure that shows the current status of the patient[9].

As mentioned earlier, tics can be distributed over the entire body. This thesis explores tics occurring solely in the facial region. The facial region can be further categorized into the mouth, eyes and nose region. Evidence from a study by H. Rickards[18] shows that tics usually start in the facial region and from facial tics, the eye tics are the most frequent. Eye tics can be divided into eye movement and eyelid or eyebrow movement. The latter can be further divided into frowning, raising eyebrows, blinking and winking[16].

Besides severity scales such as the YGTSS, another measure that is used to determine how severe the disorder is, is counting tics in video recordings. Chappell et al. have determined that tic counts over a duration longer than 10 minutes are reliable[9]. Furthermore, the authors have examined what the minimum length of a recording is in order to reliably determine the severity of the disorder in general. They have shown that offline counting of tics in video recordings is indeed feasible. The video recording has only a narrow time frame of 30 minutes or less, while the YGTSS rating is based on interviews over a timespan of at least a week. Chappell et al. concluded that a video recording "of at least 10 minutes duration" is needed.

1.1 Previous research

Research has been done about automatically detecting tics of GTS patients. M. Bernabei et al. used triaxial accelerometers to detect tics[3]. As tics are generally mapped onto sudden movements, these accelerometers can detect a tic if such a sudden movement is present. However, this method can only detect tics for certain body parts which are suitable for accelerometers, such as the torso, arms and legs. For detecting facial tics these accelerometers are not suitable and a different approach is needed.

1.2 Goals

This project is a result of an internship at the HSK Group Den Bosch with drs. J. van de Griendt as contact person. The purpose of the internship is combining two fields (Psychology and Artificial Intelligence) into one project. During the early stages together with J. van de Griendt a specification of a system which detects facial tics is formulated. Eventually, the system is implemented based on the specification and design guidelines are given for further use of the system.

The aim of this project is to explore the possibilities of a system that automatically detects facial tics in recordings of GTS patients. Since the therapy requires great effort from both the therapist and the patient, the proposed system is to alleviate these parties from being dependent on long interviews or manually scoring videos. A possible solution to this issue would be to feed the video recording into a system which could automatically detect tic movements at any given interval based on expert information.

The research question for this project consists of two parts:

1. 'What are the domain-specific features of facial tics used by experts'
2. 'Can these features be used for automatically detecting facial tics? If so, how can these be used?'

1.3 Outline

The main part of the thesis consists of explaining the design and implementation process of the tic detection system. The thesis is divided into six parts and these parts are summarized here:

1. Chapter 2 describes the methods for this project and the features that are used in the tic detection system are listed. Additionally, the most prominent features as provided by experts are described.
2. In Chapter 3 the system specification is described and a detailed description of the pipeline of the system is given. The domain-specific features of facial tics together with the feature extractor are specified.
3. Chapter 4 gives a short description of the application at user level which runs the tics detection system. Furthermore, trade-offs that were made during the implementation of the system are discussed.
4. The results as well as an interpretation and explanation for the results of this exploratory study are described in Chapter 5. Here, the second part of the research question is answered.
5. In Chapter 6 a detailed discussion of the tics detection system and a reflection on the performance of the system is given.
6. In the appendix a listing of the code used for generating the data set for the system is provided and the dependencies of toolkit libraries is listed.

2 Methods

In this chapter the methods for acquiring the data set is defined and the features that are used in the system are explained and justified.

2.1 Data acquisition

2.1.1 Old data

At the HSK Expertise Tics Den Bosch recordings were available of GTS patients from therapeutic sessions. During a couple of meetings these recordings are gathered in the presence of an expert to be analyzed and used for the proposed system. These recordings were analyzed to see which facial tics are present in the patients. The ideal position for the recordings would be such that the patients are facing the camera. When using computer vision techniques to process facial images, often subjects have to sit straight in front of the camera, as literature shows in many studies[10],[19]. However this is not the case in these recordings, because the patients sit in front of the camera with an angle and looked at the interviewer positioned next to the camera. Sometimes this occluded one part of the face and in other situations the face of a patient is tilted.

During a test stage, the results from the dataset acquired from HSK Expertise Tics were not promising when inspecting the performance of the system. In several studies concerning computer vision and human faces, the head is fixed in front of the camera and then computer vision techniques are applied[17]. These techniques are subject to a reliable and controlled input. This was unfortunately not the case with the dataset. The noise — in terms of the rotation and tilt of the head — was too big during the test stage and a more reliable dataset was constructed to check if the bad performance was due to a dataset which was not reliable enough. This presumption appeared to be correct when testing the custom dataset during the test stage. The results of various configurations using the dataset acquired from HSK Expertise Tics are shown in Table 1 using the first feature (blink detection with template matching) of Section 2.3. These results show that the performance is unacceptable with this dataset. Therefore, the rest of the project is based on this custom dataset.

#	Threshold		Performance
	Left	Right	
1	0.90	0.90	0.52
2	0.85	0.85	0.53
3	0.80	0.80	0.59
4	0.70	0.70	0.62
5	0.50	0.50	0.62

Table 1: The performance measures of the blink detection with template matching feature of the HSK Expertise Tics dataset.

2.1.2 New data

Since this is a study to explore what domain-specific facial tics features are used by experts and to focus on an implementation of the system, video material has been acquired from three subjects. Since the goal of this thesis is not to generalize over a large population, this should already give an indication whether a feature can be accepted or rejected as a good domain-specific feature. One subject did not wear glasses, whereas the other two subjects did. The built-in iSight camera of a MacBook Pro with a resolution of 640 x 480 pixels was used to record the video material. The lighting was the same intensity during all recordings. Each subject was set in front of the webcam for 10 minutes and performed a simple task. During this task the subject had to blink with either the left or right eye or both eyes. Furthermore, the subjects were allowed to move other parts of the face, e.g. the mouth corners or the whole face, as well. The interval was not defined, but each subject was asked to produce tics at random intervals.

There were three constraints on the movements the subjects were allowed to make.

- The first constraint was that the head had to stay within the range of the camera to make sure that the face would not get out of bounds.

- The second constraint was that the subject had to face the camera without tilting or rotating their heads.
- The last constraint was the duration of each tic. Based on the footage of the HSK Expertise Tics Den Bosch, on average the duration of a tic is one to two seconds.

The subjects were asked to take these constraints into consideration while making their tic movements.

The video material was annotated by hand using the annotation software ELAN[7] from the Max Planck Institute for Psycholinguistics (`http://tla.mpi.nl/tools/tla-tools/elan/`) to indicate at which intervals a facial tic occurred. These intervals are the positive samples, while every other interval in the video material – containing no facial tics – is a negative sample. The positive and negative samples form the dataset on which the tic detection system is trained and tested.

After the annotation process, all intervals indicating tics were automatically clipped from the original video. To create the negative samples, a similar method was used, only this time random clipping was applied on the intervals where no tics occurred. The scripts for the clipping process are listed in Appendix A. The length of these intervals was determined based on the average length of positive samples. Furthermore, the number of negative samples depended on the number of positive samples in that particular video material. From the three subjects the number of positive samples extracted is respectively 75, 90 and 48. The number of negative samples is the same as the number of positive samples. Thus, the dataset consists of 213 positive and 213 negative samples.

The dataset is divided into a training set and a test set using 10-fold cross-validation where the accuracy of the system is expressed as the average accuracy of all iterations. This way the system is tested on untrained data and the accuracy is averaged over different compositions of training and test sets. To remove the random element of creating a permutation for the K-fold cross-validation, the 10-fold cross-validation is repeated for 200 times.

2.2 Requirements

Before a system can be designed, requirements have to be documented. These requirements are used as a guideline during the implementation of the tic detection system and can justify certain design choices.

Performance One of the most important requirements of the system is the performance. The system must not only be able to classify video material as a tic or a non-tic, but also do this as accurately as possible. Aside from focusing at the correct classifications of the data, it is also important to take the incorrectly classified data into account. In practical use the number of mis-classifications has to be minimal in order to reduce the error rate. There are GTS patients where the number of tics are sparse compared to the number of non-tics. In those cases the number of mis-classifications is high when the false positives is high.

Robustness The system must be able to cope with a variety of data input. The methods for the features described in Section 2.3 must therefore be tested on robustness.

CPU/Memory usage When the tic detection system is used in practice it should not consume too many resources of the computer it is running on. This way the system can be embedded into an application and is runnable on any modern computer.

Real-time The preprocessing and classification process need to be near real-time in order to have a usable application which feeds a webcam stream into the system. Although creating an application to use a webcam stream is not part of this project, it is important to keep this requirement in mind for future research.

2.3 Feature selection

Blinking The literature about the GTS mentions various types of facial tics ranging from the eye region to the mouth region. Subsequently, each region can be subdivided into more specific tics. The most occurring tics in the eye region are listed in a study by Martino et al. [16]. Of the 212 GTS patients they have analyzed, 94.8% reported eye tics. Of these patients that reported eye tics, 91.5% had eyelid and/or eyebrow tics and consists among others of frowning, blinking and winking. Because the occurrence of these eye tics are that common, a good feature to select is looking at the eye movements and in particular at blinking and winking. Two prominent methods for detecting blinks are explained in Section 3.2.2 and Section 3.2.3.

Facial landmark distances Next to features related to blink tics, there are additional features that provide information on different facial tics. Many examples are given in a study by Rickards of which one is moving the mouth to the side[18]. This is one of the mouth movements that can occur during a tic. In order to detect such movement, facial landmark distance features can be used. These facial landmarks are points on the face such as the eye corners and mouth corners. When a subject moves its mouth during a tic, the distance between these points change. This feature is another type of indicator whether a facial tic has occurred. Section 3.2.4 describes the method used for extracting this feature.

Motion When studying the recordings of facial tics of a GTS patient, tics are paired with movement. More specifically, the motion present in the eye and mouth region. Subsequently, an intuitive feature would be to examine motion. Using a detection method for motion — further described in Section 3.2.5 — this feature could be a third type of feature to benchmark.

Wrinkles Aside from blinking, facial landmarks and motion, the increase of wrinkles is another indicator of a tic occurrence. As Azrin et al. mention in their study the "wrinkles around the eyes were evident each time the eyes were tightly closed"[1]. The method for detecting wrinkles is described in Section 3.2.6.

(a) (b)

Figure 1: A neutral face is shown in Figure 1a, whereas an example of the various features present in typical facial tics is shown in Figure 1b.

Each feature described above — eye blink, distance between facial landmarks, motion (when it is compared to a previous frame) and wrinkles — are present in Figure 1. These features are implemented by a tic detection system as explained in Chapter 3 and are benchmarked to see whether they are suitable and robust indicators of tic occurrences. The results of the benchmarks are further discussed in Chapter 5.

3 System specification

In order to test whether the features can be used in an automatic tic detection system, such a system had to be designed for this project. In this chapter the requirements for a system are discussed as well as the specifications. It consists of state-of-the-art components which are combined into one system. During the project, the focus on the type of tics started mainly on eye tics, but has slightly shifted towards other facial features as well. This shift occurred to test whether incorporating other facial features could contribute to the overall performance of the system. In the following sections, the pipeline of the system as depicted in Figure 2 will be discussed and each processing step will be specified.

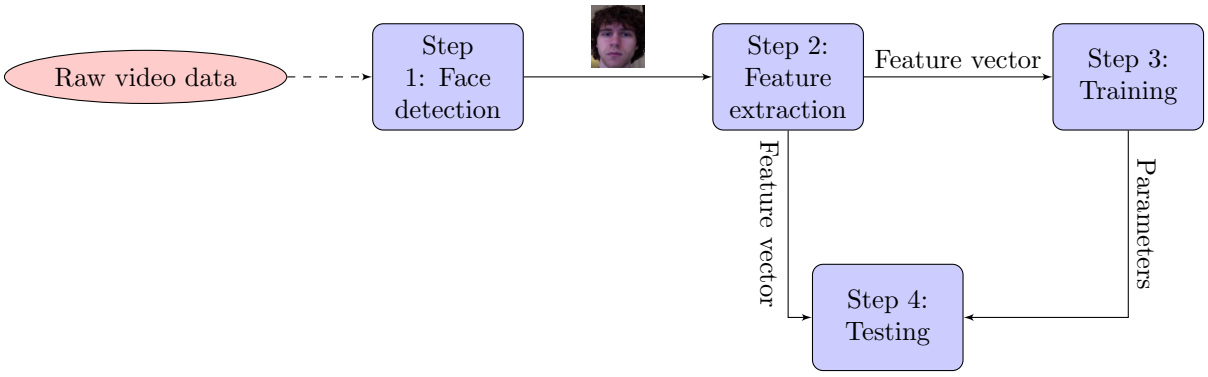


Figure 2: The pipeline of the automatic facial tic detection system.

3.1 Step 1: Face detection

Step one of the pipeline is removing any unnecessary information from the data to make it more compact. A region of interest (ROI) of a frame is determined to minimize the amount of data. The ROI contains only the face of the GTS patient, since other information is irrelevant for detecting facial tics. In order to extract the face from every frame, an object detector implemented by Lienhart[14] is used which makes use of Haar-like features. These are horizontal, vertical and diagonal lines which together describe the object which needs to be detected and are depicted in Figure 3. A Haar-like feature examines the adjacent regions at a specific location, pixel intensities are summed up in each region and the difference between these sums are calculated. The difference in each section is then classified by a cascade classifier. A cascade classifier consists of multiple simpler classifiers that are applied in a sequence until a ROI with differences is rejected at an earlier stage or accepted by all stages.

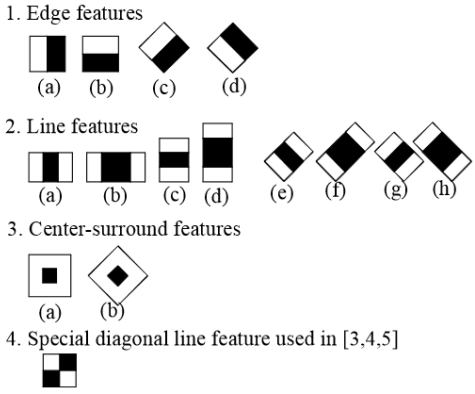


Figure 3: Examples of Haar-like and center-surround features. Figure adopted from paper by Lienhart et al.[14]

The cascade classifier for object detection slides a search window across the image and applies the Haar-like features to the search window. Furthermore, the classifier operates independent of the size of the search window. This means that a classifier can detect an object regardless of its size. The implementation of the classifier starts with a minimum size of the search window to detect the object and increases it if necessary. The minimum size of the search window is set to 100×100 pixels. Besides the minimum size the classifier also needs to have a model which can detect a face. The type of model that is used is the frontal face cascade model. The model consists of multiple stages and decision trees with thresholds in order to accept or reject a candidate face in a search window. This pre-existing model is trained on a few hundred sample views of a particular object; in this case a face facing the camera. The last step of localizing the face is building a rectangular shape around the face to retrieve the ROI for later use.

3.2 Step 2: Feature extraction

In this section the process of extracting features from the data is explained and each individual type of feature is highlighted.

Feature extraction starts with condensing the ROIs retrieved from step one into various features by applying the feature extractor to each frame. The feature extractor produces multiple types of features, stored in a vector which contains feature values for each frame. Each feature of Section 2.3 is further described in the following section. During the implementation of the tics detection system, the feature extractor and all of the features are built entirely from scratch.

3.2.1 Sliding window

For most features that describe a tic the factor time has to be taken into account, because the tic develops over time. Therefore a sliding window is needed that stores feature data for a number of frames, which is illustrated in Figure 4. The sliding window is of fixed length and keeps record of past values. Thus, given a window of size s at frame t the values v of the frames $t - s$ until t are stored and the feature vector is filled with $f(v_{t-s}, \ldots, v_t)$.

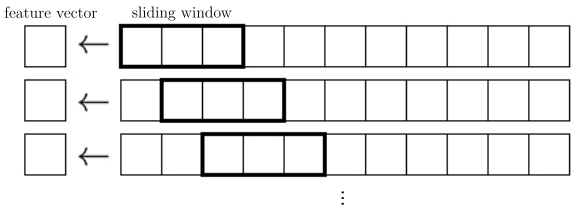


Figure 4: Illustration of the mechanics of a sliding window.

Note that at the start of a video fragment, at $t = 0$, there are no previous frames. Therefore, feature extractor starts processing the sliding window after it has been filled with the first s values.

3.2.2 Blink detection: Template matching

The literature mentions multiple ways of detecting a blink of which some are more robust than others. The two most prominent methods are compared with each other. In this section the first method is explained and the second method is explained in the next section.

The first method to detect a blink is to use the template matching technique. Template matching is a standard approach used in pattern recognition and feature detection[13],[5], [10]. In this technique an image of an open eye is used as a template. The template slides over all pixels of the frame and at every pixel it is calculated how well the template matches the current position. This method is formalized in Equation 1. The template summations in the numerator and denominator are done over all channels and separate mean values are used for each channel. However, the result R is still a gray scale image which makes it convenient to analyze.

$$R(x,y) = \frac{\sum_{x',y'}(T(x',y') \cdot I(x+x',y+y'))}{\sqrt{\sum_{x',y'} T(x',y')^2 \cdot \sum_{x',y'} I(x+x',y+y')^2}}$$

Equation 1: I denotes the frame in which a match must be found using the template image. T stands for the template image. This is the part of the frame which needs to be matched. The summations runs over $x' = 0 \dots w-1, y' = 0 \dots h-1$, where w is the template width and h is the template height. In the resulting matrix R the value is calculated using the normalized correlation coefficient metric at coordinate x, y.

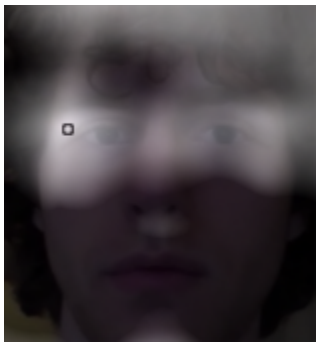


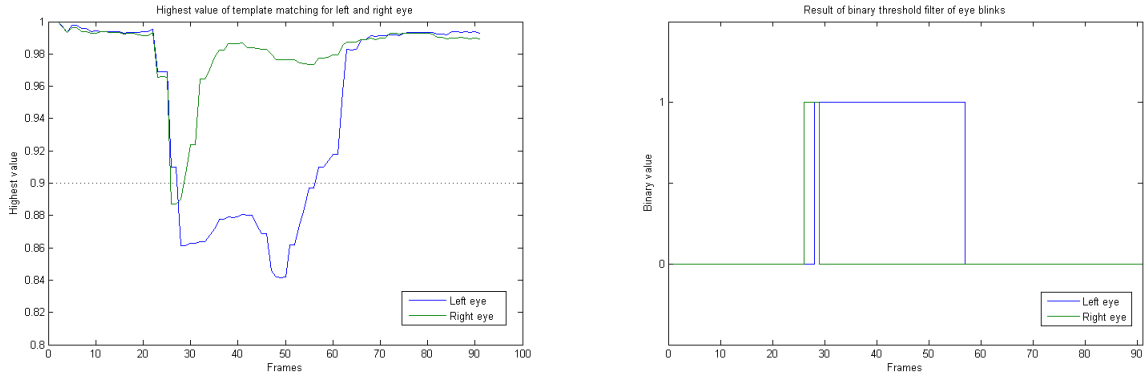
Figure 5: A visual representation of matching a left eye template onto an input image. The visualization of R is a heat map of the best match. The best match is the highest value in R and is visualized with a circle that indicates the brightest spot.

In a technical report of Chau et al. a similar approach was used to detect blinks with a USB webcam[10]. An example of matching a template of the right eye is shown in Figure 5. In order to work with the template matching method, templates of the left and right eye need to be created. For convenience, these templates have to be created as soon as a sample is being processed and therefore is fully autonomous. This means that there is no preprocessing required to generate templates of the subject and load it into the system. Therefore, the eyes need to be detected and extracted at the beginning of each sample. This results in a smaller image T describing the open eye, which can be used to detect an open eye at later intervals in the sample.

In the report of Chau et al. an eye localizer and an eye tracker were used. Motion detection and various post-processing techniques were used to extract the eyes from the image during a blink. For this project, a different approach for detecting the eyes is used. This approach assumes that in each sample the subjects head is facing the camera. The eye regions are calculated using constants based on the dimensions of the face, as shown in a study by Timm et al[21]. The four constants that are required to estimate the eye region are $percentTop$, $percentSide$, $percentHeight$ and $percentWidth$ with respectively the assigned values 25%, 13%, 30% and 35%. Since these regions are expressed in terms of percentages the regions scale to the size of the ROI.

The method used here for detecting an eye blink from a sample uses a metric to determine whether an eye blink occurred. It calculates an average of the highest values R over time t frames using a sliding window, as described in Section 3.2.1. The last step of this method is applying a threshold to the metric. As Figure 6a depicts, the highest value of R drops below the threshold for a period of time. This time period is utilized as a heuristic for detecting blink tics. With a binary threshold filter the drops below the threshold can be extracted more easily from the data. At each time frame the number of drops within a sliding window is counted and an average (with respect to the window size) is stored as the feature value.

Determining which threshold results in the best performance is still a manual process. Based on the input samples, the threshold is determined by trial-and-error. A series of thresholds are benchmarked and each threshold is combined with varying sliding window sizes to benchmark which setting optimizes the performance.



(a) (b)

Figure 6: Figure 6a depicts the highest values during one of the samples. The solid lines indicate the highest value of the template matching process and the dotted line indicates a possible threshold. Figure 6b shows the result of a binary threshold filter using the threshold depicted as the dotted line in Figure 6a.

3.2.3 Blink detection: Pupil gradient

The second method for detecting blinks does not use a template of an open eye to match in an image as described in Section 3.2.2, but instead detects the pupil in the eye region. If the pupil can be detected it is assumed the eye is not closed and when the pupil cannot be detected there is a blink. In order to successfully detect pupils a method described by Timm et al. is used to accurately detect the center of the eye using gradients[21]. The method of using gradients is robust and can process faces where the eyes might partly be obstructed by objects such as glasses or hair or obscured by shadows or low contrast. The approach by the authors yield an accuracy of 82.5% for pupil localization and the iris is localized with a probability of 93.4%.

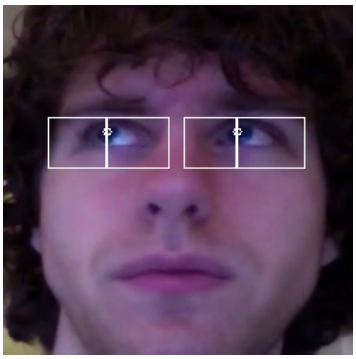


Figure 7: Demonstration of the detection of the pupils using the method described by Timm et al.[21]

Figure 7 shows the output of the algorithm described above. The output of the algorithm consists of three parts of information about the eye. The first two parts are the left and right eye regions for each eye. The last part is the location of each pupil. The algorithm determines the eye region of each eye which are subsequently subdivided into the left region and the right region. This is done with the same constants as described in Section 3.2.2.The next step is to check whether there is a gradient that could match an iris. If there is a match, the iris center location is returned and is depicted as a circle. However, when there is no correct iris localization possible, there is no clear indication how to detect that the algorithm has not localized the correct location of the iris. Since it always tries to find a matching gradient, it will also try to find such gradient outside the left and right region. This way the center point moves outside the left and right region, but still remains inside the eye region. The feature extractor

tries to detect this behavior and classifies a location of the pupil as a mismatch when this center point moves outside its boundary.

3.2.4 Facial landmark distances

Examples of facial landmark distance features are the distance between the mouth corners and the distance between the left eye corner of the left eye and the left corner of the mouth (as well as for the opposite side of the face), as depicted in Figure 8 with black lines. Seven so-called facial landmarks are needed to calculated these features. The facial landmarks are determined by the facial landmark detector by Uřičář et al[22]. This detector uses an estimation approach where the points are estimated based on a Local Binary Patterns (LBP) pyramid and parameters which have been trained with Support Vector Machines (SVMs)[22]. The LBP pyramid consists of LBP features which are computed in every pixel within a certain predefined boundary and the input image is also scaled to varying sizes. The detector is compared with a detector of Everingham et al., an Active Appearance Models (AAM) and independent SVMs. According to Uřičář et al. from all methods they have compared with, this method gave the best result and was the most robust. In this case robustness is required, because with a steady image, fluctuations in the position of each facial point will result in a unreliable feature.

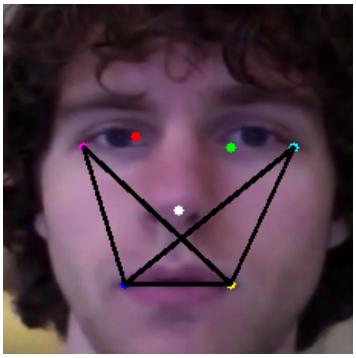


Figure 8: The facial features based on the seven facial landmarks using a detector by Uřičář et al.[22]

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$$

Equation 2: The Euclidean distance metric on a two-dimensional plane where $p = (p_1, p_2)$ and $q = (q_1, q_2)$.

The distance metric used in this feature is the Euclidean distance between point p and q using the formula in Equation 2. The value in the feature vector for this feature is the average distance over t time frames in a sliding window.

3.2.5 Motion

Extracting a value for facial motion is done in two steps. First, two consecutive frames are compared with each other and a (gray scale) difference image is calculated. However, this difference image contains noise inherited from the original frames. A Gaussian filter is applied to reduce noise in the difference image. Subsequently, a binary threshold filter is applied to highlight the parts of the image with the highest motion values. Depending on the threshold value, more or less pixels are evaluated as having moved. In a visual representation these areas are depicted in white in Figure 9. The second step is extracting regions of interest in the binary image. Only a few areas are of importance in detecting the tics, such as the eyes and the mouth. The percentage of white pixels in these ROIs is used as the feature value. The method described above is expressed in pseudo-code in Listing 1.

```
def motion(imageOne, imageTwo, thresh):
        blurredImageOne = gaussianFilter(imageOne, Kernel(3, -1))
        blurredImageTwo = gaussianFilter(imageTwo, Kernel(3, -1))
```

```
differenceImage = diff(blurredImageOne, blurredImageTwo)
thresholdedImage = threshold(blurredImage, thresh, THRESH_BINARY)
return (countNonZero(thresholdedImage) /
        (thresholdedImage.rows * thresholdedImage.cols))
```

Listing 1: The procedure for calculating the motion feature value in pseudo-code.

(a) (b)

Figure 9: Figure 9a is a visual representation of the difference between two pictures before the threshold filter. Figure 9b depicts the same difference picture after applying the binary threshold filter. Both figures are from the same positive sample.

3.2.6 Wrinkles

As outlined in [20] the last method could extract a number which denotes how many wrinkles there are in a ROI by detecting the edges. This can be done by using the Canny edge detector[8]. The Canny edge detector is chosen, because it is computationally efficient and tries to satisfy two important criteria. The first criteria is having a low error rate, meaning that this algorithm minimizes the false positives. Second and finally, the localization of edge pixels is good, meaning that the distance between the detected edge pixel and the real edge pixel is minimal.

The algorithm consists of four main steps. The first step is preprocessing the image using a Gaussian kernel to filter out noise. Next, the algorithm computes the intensity gradient of the image by convolving the image with the Sobel filter. This results into a gray-scale image where black and white pixels have a high gradient and gray pixels have a small gradient. The black and white pixels are the possible edge pixels. After computing this gradient, all pixels which are not considered to be part of an edge are removed. The last step of the algorithm is accepting a pixel as an edge or rejecting the pixel using two thresholds. The first threshold is the upper threshold, which accepts a pixel gradient as an edge if its value is higher than this threshold. The second threshold is the lower threshold, which rejects a pixel if its value is lower than this threshold. When a pixel value is between the two thresholds, the pixel is accepted when it is adjacent to a pixel of which value is above the upper threshold.

At every frame the Canny edge detector is applied. Because Canny recommends a $2:1$ or $3:1$ ratio for the *upper* : *lower* thresholds in his study, using only the ratio and the lower threshold, the upper threshold can be calculated. For this feature a ratio of 3 is used. When the edge detector is applied to an image, it generates a mask with the detected lines. After applying this mask to the original image, this produces a black and white image with the detected edges as shown in Figure 10. An implementation in pseudo-code is given in Listing 2.

```
def wrinkle(image, thresh):
        ratio = 3
        blurredImage = blur(image, Kernel(3, -1))
        edges = canny(blurredImage, thresh, thresh * ratio)
        return (countNonZero(edges) / (edges.rows * edges.cols))
```

Listing 2: Pseudo-code for calculating the wrinkle feature value.

(a) (b)

Figure 10: Figure 10a shows the edges detected during a negative (non-tic) sample. Figure 10b depicts the detected edges during a positive (tic) sample.

The last step in order to output a metric is expressing these lines in a single value. The white pixels are divided by the total number of pixels of the ROI, which is the eye region. The resulting value is a possible metric for the wrinkle feature. Again, a sliding window is used to incorporate the time factor.

3.3 Step 3: Training

During the training phase of the project, the detection system is trained on random permutations of the data set as described in Section 2.1. All samples in the training set are processed by the feature extractor which results into a feature vector. The system can be trained on specific features. Each feature discussed earlier can either be turned on or off. The classifier for this project is an Support Vector Machine (SVM)[23]. An SVM works well with discriminating two classes, which are used in this system. An SVM discriminates samples by separating the data points with a line. This line describes the boundary of the categories. With new samples the SVM looks at which side this new sample is located and can then label it with a category. An SVM operates based on supervised learning and therefore needs labeled training data. So, each sample in the dataset that will be used also is labeled indicating either being a positive/tic sample (1) or being a negative/non tic sample (-1).

SVM settings needs to be configured in order to give sensible results. The first setting is the type of a SVM formulation. From the possible formulations (C-SVC, ν-SVC, Once-class SVM, ϵ-SVC and ν-SVR) C-Support Vector Classification seems the best option, because it can classify n ($n \geq 2$) classes with a penalty multiplier C for outliers.

The next setting is the kernel type of the SVM. With the set of possible types (linear kernel, polynomial kernel, radial basis function and sigmoid kernel), the radial basis function (RBF) is recommended by the OpenCV documentation in most cases. The linear and polynomial kernel perform as good as the RBF kernel on 1D or 2D feature vectors, but overall the RBF kernel is more suitable for higher dimensional data. The RBF kernel requires a parameter γ, which is a scalar for the similiarity measure in the RBF equation. γ can either be set manually or calculated automatically while training (see OpenCV documentation SVM::train_auto). With the latter option, OpenCV performs a 10-fold cross-validation to estimate the optimal parameter values. After running this automatic parameter selection it resulted into an average $\gamma = 0.0001$.

The final setting is the termination criteria for the SVM. There are three types of termination criteria. The first criteria is stopping after n iterations and the second criteria is stopping when the accuracy becomes lower than ϵ. The last criteria is a combination of the first two, where the algorithm stops after n iterations or when the accuracy becomes lower than ϵ, whichever comes first. For this project the default setting was used, which is to stop after 1000 iterations.

3.4 Step 4: Testing

Once the training phase is finished, the system uses the trained configuration and tries to predict the label of the yet unseen test samples. Similar to the training set, this test set is being processed by the feature extractor with as result getting a feature vector.

During the test phase the true positives, true negatives, false positives and false negatives are counted to later calculate the overall performance. Once the cross-validation is completed the process of cross-validation is repeated for 200 rounds to cancel out the random permutation. Per round the average performance is being calculated and the overall performance of the system is the average of these round averages. How the performance is calculated per round and overall, is shown in Equation 3.

$$Performance = \frac{TP + TN}{P + F}, \text{ with } P = TP + TN \text{ and } F = FP + FN$$

Equation 3: This formula shows how the performance is calculated.

4 'Tics' application

In this section trade-offs for implementing the application will be discussed and the application which runs the tics detection system is described at user level.

4.1 Implementation

A video sample consist of a sequence of pictures. Each picture or frame contains pixels. Computer vision techniques are required to reason about these pixels with a computer application. For this project the computer vision library called OpenCV was used. It contains numerous functionalities, but the most important functionalities are concerned with image/video processing, object detection and machine learning using a Support Vector Machine (SVM). The image and video processing module of OpenCV handles opening, reading and parsing images and videos. Furthermore, images are represented in a mathematical form by using matrices. This makes it convenient to apply algorithms to images and access individual pixels. Popular algorithms for object detection are already implemented in OpenCV, of which one is a classifier which detects objects based on a model. For this project using the classifier reduces the development time of a system which requires such algorithms. So the focus can be shifted from implementing already existing algorithms to implementing feature extraction.

The tics detection system itself is programmed in C++, because most dependencies that were used for developing the system were also implemented using C++. Furthermore, C++ is a low-level programming language which makes it possible to do memory management. The system is implemented using Visual Studio and is built for a Windows operating system, but the source code can also be compiled for other operating systems if needed. The full list of dependencies of toolboxes and libraries is available in Appendix B.

4.2 User interface

The user interface (UI) does not have a graphical shell, since this prototype is part of an exploratory study to see which features work best for detecting tics and does not have to be user friendly. A command-line application has been made which accepts input arguments to configure the system and interact with it.

```
Usage:
  -h [ --help ]           Produces help message
  -d [ --dir ] arg        Data input directory
  -l [ --load ] arg       Load features dump file
  -s [ --save ] arg       Save features to dump file (default: out-<time>.txt)
  -F [ --feedback ]       Enable feedback
  -k [ --kfold ] arg      Set number of k-folds (default: 10)
  -f [ --features ] arg   Set features to detect
  -L [ --list-features ]  List all features possible to detect
  -r [ --rounds ] arg     Set number of iterations folds are executed (default:
                          200)
  -w [ --win-size ] arg   Set sliding window size (default: 0.8)
  --bl-thresh arg         Set blink left threshold (default: 0.85)
  --br-thresh arg         Set blink right threshold (default: 0.85)
  --wrinkle-thresh arg    Set wrinkle threshold (default: 40)
  --motion-thresh arg     Set motion threshold (default: 30)
```

Figure 11: The help command shows all possible command-line parameters to interact with the application. Next to each parameter a short description is given.

Most of the parameters have a default value to simplify configuring the detection system. The most important parameters are *--dir* (*-d*), *--features* (*-f*) and the feature specific parameters such as *--win-size* (*-w*) and the blink threshold parameters.

When setting the 'data input directory', a particular directory structure is expected. If this is not the case, an error message is returned reminding the user what kind of directory structure is needed. Currently a parent directory with a 'pos' and 'neg' subdirectory is accepted. These two subdirectories may contain more subdirectories, but the application automatically traverses through all depths of the parent directory. The files matching a video file are being loaded and fed into the feature extractor.

Sometimes the step with the feature extractor needs to be skipped and the already existing feature vectors need to be trained on immediately. This can be the case when it is desired to re-run the training and test phase. In this case the *--load* (*-l*) is useful. With this parameter a file containing a serialized object which holds all feature vectors of a particular run can be loaded into the system. Unless the dataset or the methods described in Chapter 3 has been changed, this serialized object is exactly the same as one would re-run the feature extractor.

4.3 Annotation

As described in Chapter 2 the data has been manually annotated using the ELAN application. With this application it is possible to inspect a video recording frame by frame and annotate each frame accordingly. The interface of ELAN is shown in Figure 12. The video playback is displayed in the top left corner and the time-line where the annotations are made is shown at the bottom of the window.

ELAN has the option to export these annotations to a file in a readable format. Such file contains all the information of an annotated segment consisting of the start and end time in milliseconds as well as the duration. Based on these files and using the code listed in Appendix A the data is clipped using QuickTime. Each line from the file is read and fed into QuickTime which requires a start and end time as well as an output filename.

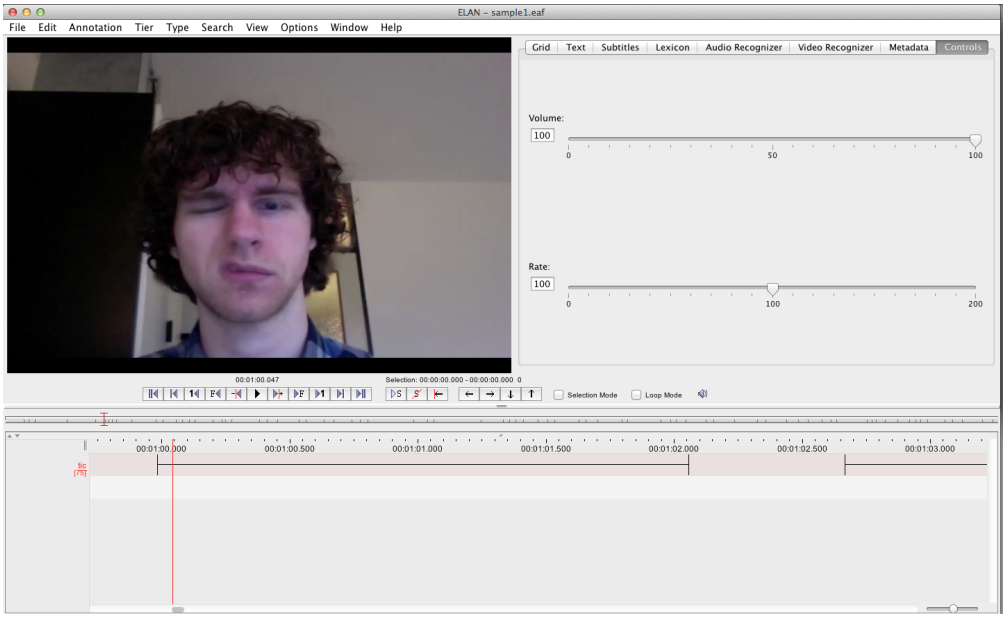


Figure 12: The interface of the annotation application ELAN. In the time-line two annotation segments are visible which denote where a facial tic starts and ends.

4.4 Feedback

Besides configuring the system with the command-line, the system also presents feedback to the user. When the parameter for feedback *--feedback* (*-F*) is set, the system outputs a visual representation of each feature of the feature vector. All outputs are listed in Figure 13.

4.5 Output

The application renders three types of output files. The first type is a serialized format of the feature vectors from the feature extractor. This format is not easily readable by a human, but the application can easily load the contents in the system and continue with classifying. The second type is a format readable for the user of the feature vectors. It contains all values of each feature and the number of the sample and the label of the sample. This output file makes debugging and performing statistics easier. The final file contains the true positives (TP), true negatives (TN), false positives (FP) and false negatives (FN) information. With the final file the performance is being calculated based on these numbers.

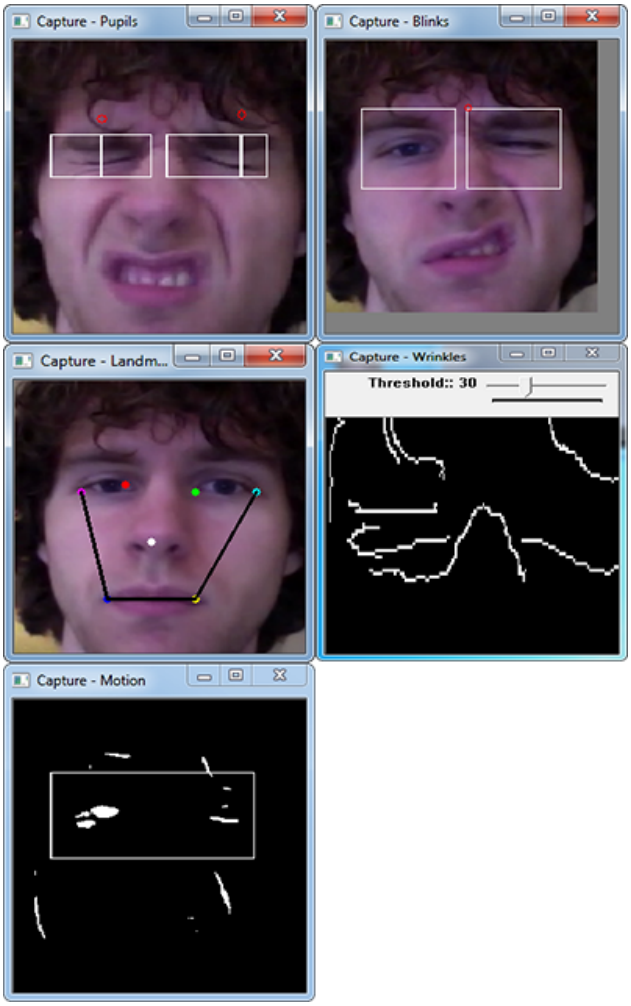


Figure 13: From top to bottom and from left to right, the five feedback windows are for the blink detection using pupil gradients, the blink detection using template matching, the facial landmarks and their distance lines, the wrinkle detection with a slider to manually change the threshold and the facial motion.

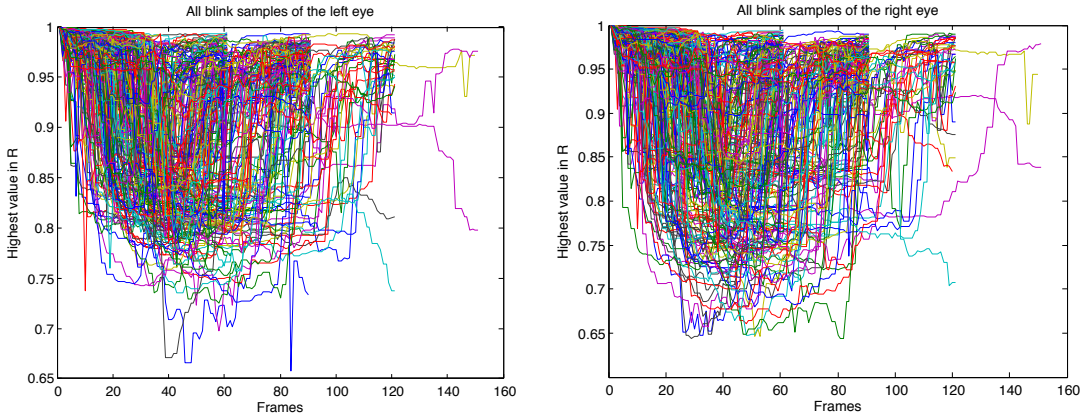
5 Evaluation

In this section the results of each feature described in Chapter 3 are shown and discussed. Aside from statistical analysis of the results, there is also a manual inspection of the feedback of the system to support the results. All performances are a result of running the 10-fold cross-validation 200 times to reduce the influence of the random permutation of samples for the cross-validation.

As mentioned in the requirements in Chapter 3, the performance has to be high enough in order to be acceptable. For a system where a miss-classification is not desired the ideal performance is 100% where the true positive rate (TPR) is 1 and the false positive rate (FPR) is 0. However, this is not feasible with the current system and a more realistic goal of around 90% for the performance is chosen.

5.1 Blink detection: Template matching

using the raw highest values in R, as described in Section 3.2.2 a plot can be made of all the positive and negative samples. Figure 14 depicts all blink samples for the left and the right eye. Because these plots show a broad band of lines of all samples, a next step would be to calculate an average over these raw values to see a more cleaner plot. This plot is shown in Figure 15 together with the standard deviation at each point.



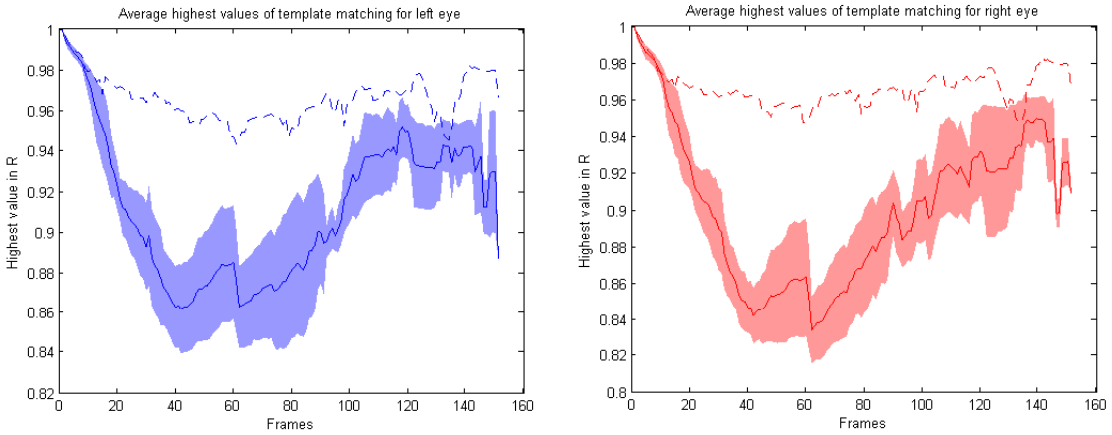
(a) (b)

Figure 14: All blink samples of the left eye are plotted in Figure 14a. Figure 14b depicts all right eye bliink samples.

Here one can see that the distinction between the positive and negative samples is clear. The two average plots show that overall the values of a negative sample (being a non-tic) stays within the 0.94 – 1.0 range while the area where the tics occur dip below the 0.9 threshold on average. Furthermore, there is a difference between the left and right eye values, which means by tweaking the threshold for both eyes individually this could increase the performance. The standard deviation is high in some cases, which does indicate that this method is not as robust is it was intended to be.

As discussed in Section 3.2.2, there are three main parameters which influence the feature values and thus the performance of the system. The first one is the sliding window size. The parameter value for the window size is expressed in terms of percentages of the frames per second. The second and third parameters are the thresholds for respectively the left and right eye blink values. As described earlier, these two parameters do not have to have the same values. Therefore, a series of benchmarks were run to see which combination of thresholds yield the optimal performance for this feature.

Figure 16 shows the performance values of each setting for the blink detection method. The three parameters (sliding window size and the left and right blink thresholds) are varied and each combination is benchmarked. The performance of any threshold combination increases linearly when the sliding window size is increased. The performance of every threshold setting rapidly drops with a window size smaller than 1.3. This can be explained by the fact that the feature value is based on a smaller number of frames which are flagged as a blink, as described in Section 3.2.2.



(a) (b)

Figure 15: The solid blue and red lines indicate the average plots of the positive samples for the left and right eyes. The dotted lines depict the average plots of the negative samples for the left and right eyes. The blue and red areas around the solid lines indicate the standard deviation.

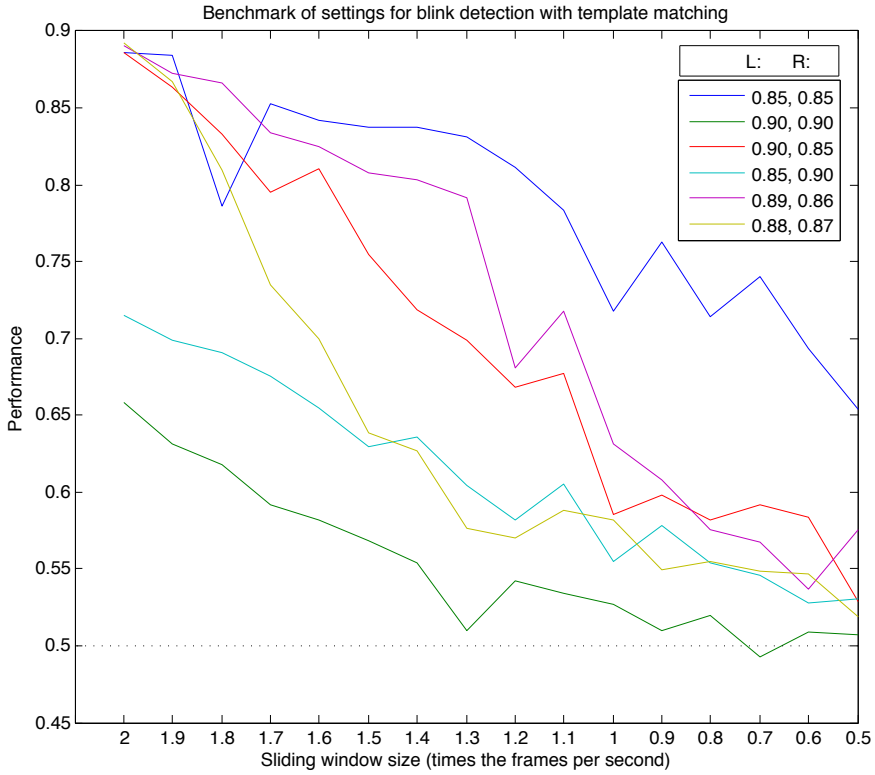


Figure 16: The performance of each setting for the blink detection method. The dotted line indicates the random chance level.

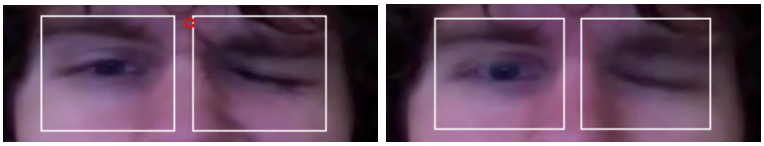
Even though Figure 15 suggests the two thresholds for the template matching method should not have the same value in order to achieve the best performance, the benchmark results show otherwise. When the left and right eye threshold is set to 0.85, the blink detection method performs better than any other configuration in the benchmark. When setting the threshold too high, as the (*0.9, 0.9*) line shows, the performance drops quickly and with a smaller window size the method drops towards the random

chance line. This can be explained by the fact that with a threshold set too high the FPR increases.

TP	TN	FP	FN	Performance
611.61	414.19	218.91	59.39	0.79

Table 2: The average statistics over all rounds with the following settings: Both thresholds set to 0.85 and the sliding window size is set to 1.8.

When observing Figure 16, there is a drop visible in the (*0.85, 0.85*) line with a window size of 1.8 times the FPS. The average TP, TN, FP and FN are depicted in Table 2. With these settings the SVM seems to miss-classify a non-tic sample as a tic sample. However, this is unexpected behavior and the reason is for now unknown. Moreover, when the window size is increased or decreased, the performance increases again.



(a) (b)

Figure 17: A correctly detected blink is shown in Figure 17a with $R = 0.841$, whereas a missed blink is shown in Figure 17b with $R = 0.907$ using Equation 1 in Section 3.2.2. A red circle indicates there is a blink in that region. The threshold used for these results is 0.85 for both eyes.

After manually inspecting the feedback there are instances of tic samples where the value of the highest template match does not drop below the threshold, as depicted in Figure 17b.

Furthermore, there is a difference between-subject as shown in Table 3 which indicates that there is need for a calibration process for each subject in order to make this feature more robust across subjects. Even though the light levels are the same throughout the samples, the skin tone is different between subjects affecting the performance. To account for this difference, the threshold must be variable between subjects.

Subject	Performance	# samples
1	0.89	150
2	0.96	180
3	0.60	96
1 & 2	0.93	330
2 & 3	0.91	276
1 & 3	0.84	246
1 & 2 & 3	0.89	426

Table 3: The performance of blink detection using template matching between-subject.

With only this feature selected and the optimal settings, the performance of the system is acceptable as it reaches the goal of 90%.

5.2 Blink detection: Pupils gradient

When processing the blink detection with the pupils gradient, the only parameter that can vary is the window size. Figure 18 shows the performance per window size. The performance is worse or equal to chance and these results are confirmed by manually inspecting the systems feedback.

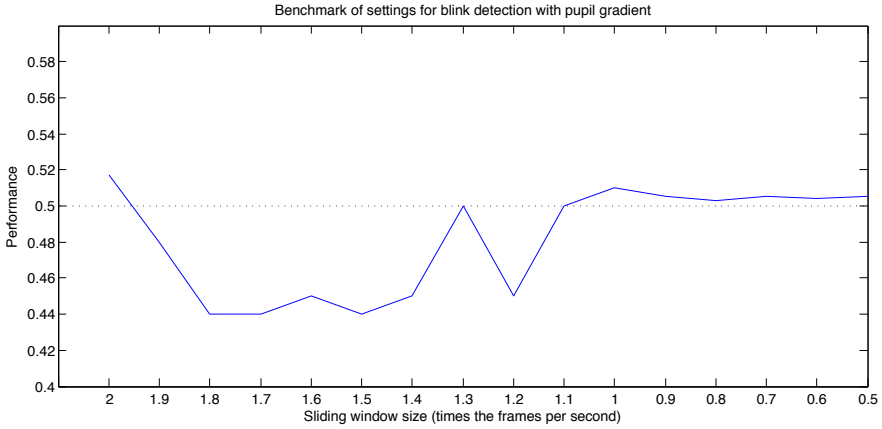
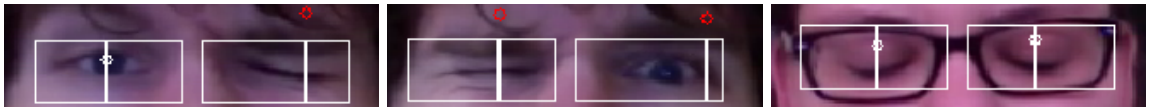


Figure 18: The performance plot for the blink detection feature using pupil gradients across various sliding window sizes. The dotted line indicates chance level.



(a) (b) (c)

Figure 19: Three examples of the pupil gradient blink detection method. Figure 19a shows a correct behavior whereas Figure 19b and Figure 19c show an incorrect behavior of the feature extraction. A white circle indicates an open eye (no blink) and a red circle indicates a closed eye (blink).

A correct detection of a blink is shown in Figure 19a. The left eye pupil is detected correctly and the right pupil has not been found within the region of the right eye.

Two examples about why this blink detection method performs so poorly are shown in Figure 19b and Figure 19c. These two figures show that there are two cases in which the method influences the performance negatively. Figure 19b shows a false negative, where even when the pupil is visible the method does not detect the correct gradient. Instead, a gradient outside the region of the right eye is chosen and the actual pupil is rejected. However, for the left eye it behaves correctly in this case. A possible explanation for this result is that the eye is slightly closed and therefore the gradient differs from gradient the algorithm is searching for. This means that the gradient it has found outside the region is a better match than the pupil it should find. This could be caused by the hair in front of the forehead.

Figure 19c depicts a false positive, where even when the pupils are not visible the method detects a gradient which corresponds with a pupil. For this result a well formed argument cannot be formulated. It could be caused by the resolution of the image in combination with shadow, but since the image gets preprocessed using a Gaussian blur that seems unlikely. Why the method detects a pupil even though the eyes are closed is open for discussion and is for future research.

Concluding, even though the authors of this method state that this method is robust and can detect pupils in images where there is low contrast and partially occluded eyes, there are interfering factor in the data because of which the method does not perform as good as intended.

5.3 Facial landmark distances

For the facial distances the only parameter which can vary is the sliding window size. The window size is varied from 1 to 2 times the FPS. Furthermore, the three distances depicted in Figure 8 are combined to see whether this improves the performance. Figure 20 gives an overview of all the distances discussed below. Based on Figure 21 a couple of observations can be made:

- The first facial distance feature is the distance between the two mouth corners. The performance of this feature is depicted as the line labeled as *M* in Figure 21. Using this feature alone gives a bad performance and maxes at 0.557 with a sliding window size of 2 times the FPS.
- The distance between the left mouth corner and the left eye corner and the distance between the right mouth corner and the right eye corner are the second and third facial distance features. These are depicted as *MEL* and *MER*. Independently these features perform as bad as or worse than chance level.
- When combining the previous two features into the feature vector, the performance increases and performs better in most cases than only the *M* feature. This combination is depicted as the *MEL & MER* in Figure 21.
- Combining the three distances mentioned earlier into the feature vector, the performance increases even more. The line denoted as *M & MEL & MER* shows the performance over this combination and the varying sliding window sizes.
- Aside from the distances of the left or right side of the face, it is also worth testing whether a diagonal distance cooperates to the performance. The performance of these two features is shown as the lines labeled as respectively *MREL* and *MLER*. The performance of these two features are comparable with the performance of *MEL* and *MER* and therefore choosing diagonal distances does not give more information about a tic to train on.
- Combining these two diagonal distances is another feature that has been tested. The performance of this feature is depicted as the line labeled as *MREL & MLER*—. This combination performs worse than the *MEL & MER* combination and can mainly be explained due to the fact that a tic in this dataset deforms the face on one side and not diagonally.
- Combining *MREL* and *MLER* with *M* in the same way as *M & MEL & MER*, the performance of this new combination is slightly better than *M & MEL & MER* as shown with the line *M & MREL & MLER*.
- The last combination that has been tested is combining all edges in Figure 20 and are depicted as the line labeled as *All* in the performance plot. When all distances are combined the performance is increased even more compared to the *M & MEL & MER* line.

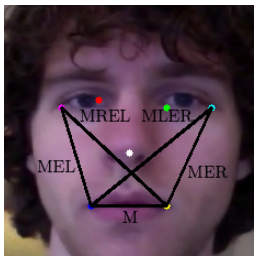


Figure 20: The facial features based on the seven facial landmarks using a detector by Uřičář et al.[22] including the labeled lines indicating the distance between the points used by the features.

After manually inspecting the feedback of the system, the results for 'M', 'MEL' and 'MER' are confirmed by the feedback. There are instances where a tic occurs, but the distance does not change significantly. This is displayed in Figure 22. The distance between the mouth corners during a positive ($d = 0.299$) and negative sample ($d = 0.305$) are close together and are therefore hard to distinguish in those instances.

Figure 22b does show that the left mouth corner (blue) is slightly higher than the right mouth corner (yellow). This means that the distance between the left mouth corner and the left eye corner compared

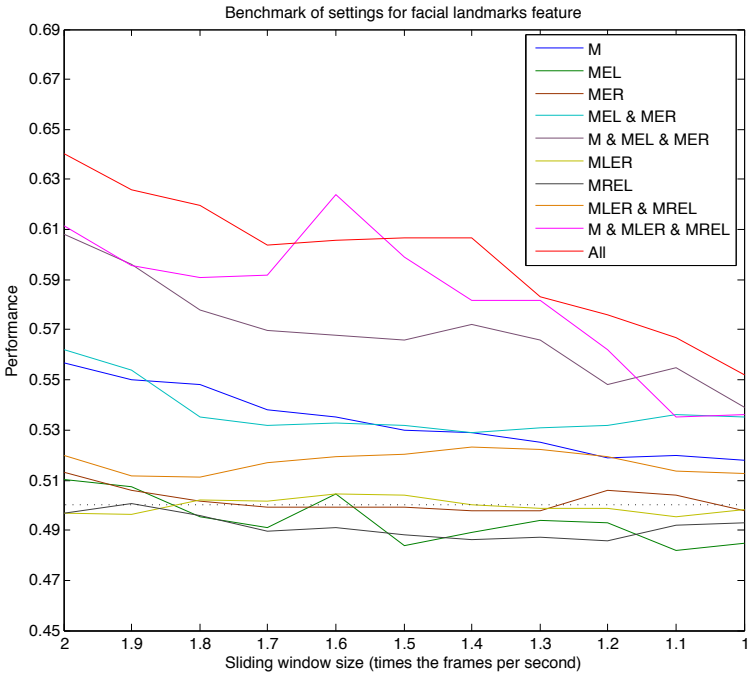
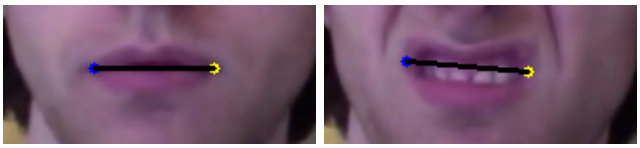


Figure 21: The benchmark results of each facial landmark feature and their combinations. The dotted line indicates the the chance level.

to the right mouth corner and the right eye corner is smaller. When both distances between the mouth and eye corner are used for training and testing, the system has more information to classify the samples compared to the distance between the mouth corners. This observation is reflected in the plot in Figure 21, because the 'MEL & MER' line is slightly better than the 'M' line.

The dataset also contains samples where the face does not deform. This implies that the eyes are shut, but the facial landmark distances remain constant. These instances have the characteristics of a negative sample — the facial landmark distances do not change — but still get labeled as a positive sample. This can be another explanation why the combined performance is low.



(a) (b)

Figure 22: Figure 22a depicts the distance line between the two mouth corners during a negative sample. The distance line during a positive sample is shown in Figure 22b. The distance value for the negative and positive sample is respectively 0.299 and 0.305 using the Eucledian distance between the two mouth corner landmarks (blue and yellow).

Overall, there is a linear relationship between the sliding window size and the performance of the features whether they are combined or are independent. The facial landmark distances are not reliable and robust enough on their own to detect facial tics in recordings.

5.4 Motion

The motion feature depends on two parameters. The first parameter is the sliding window size which varies from 0.5 to 2 times the FPS. The second parameter is the binary threshold value which filters the low intensity pixel values from an image as described in Section 3.2.5.

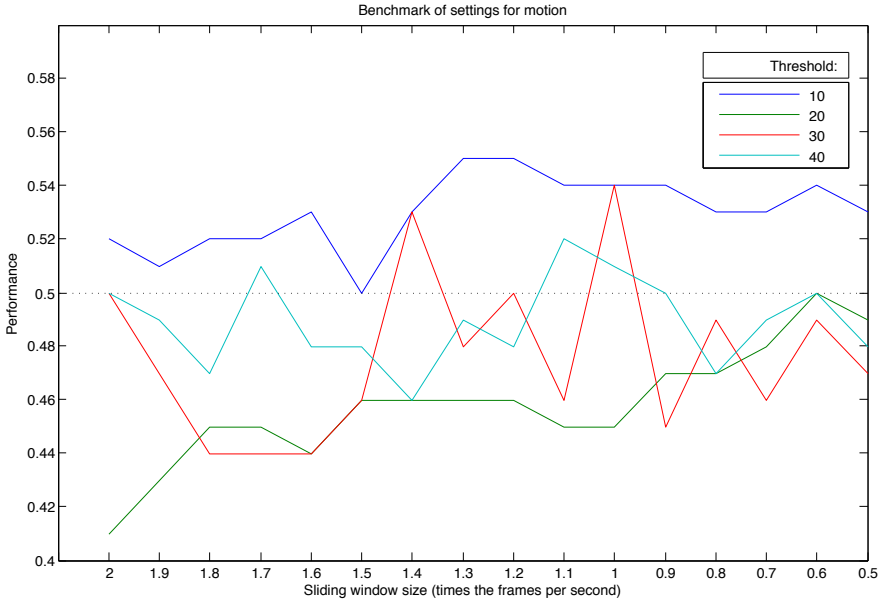


Figure 23: The benchmark for the motion feature and its settings. The dotted line indicates chance level.

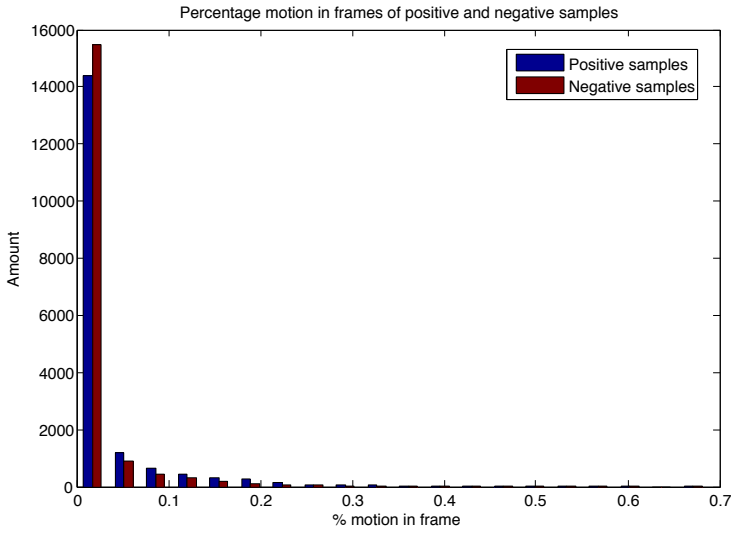


Figure 24: The histogram shows the feature values of both the positive and negative samples using a threshold of 20.

Figure 23 shows the performance for the various thresholds. The motion feature performs badly when the window size is large and slowly increases when the window size decreases. Setting the threshold to 10 for this feature — which is used for the Canny edge detector — stays above chance level whereas the rest of the settings perform worse than chance level.

The histogram shown in Figure 24 depicts the feature values of the positive and negative samples. The problem here is the large overlap between the values for the positive and negative samples. With the current method of extracting motion there is no clear distinction between the two classes possible.

Each tic is paired with movement or motion, but there is a crucial difference between the motion and a facial tic. The motion is only visible in the tic onset. Once the eyes are closed the motion is gone and the value reads zero. The main reason why the positive and the negative samples have the same signature for motion is because a normal blink and a tic blink do not differ much in the motion part. They do differ on the duration of having the eyes closed and that phase is not paired with motion. Therefore, setting a threshold for this feature is redundant.

Thus, detecting facial tics of GTS patients using the motion feature is not reliable enough.

5.5 Wrinkles

For this feature there are two parameters that can be configured. The first parameter is the sliding window size which varies from 1 to 2 times the FPS. The second and last parameter is the lower threshold for the Canny edge detector and the upper threshold is fixed to $3 \cdot$ lower threshold. For the lower threshold four settings are benchmarked, namely 15, 20, 25 and 30. Setting the threshold lower than 15 results in an excessive amount of edges at regions where there are no edges and for setting the threshold higher than 30 there are too little edges detected.

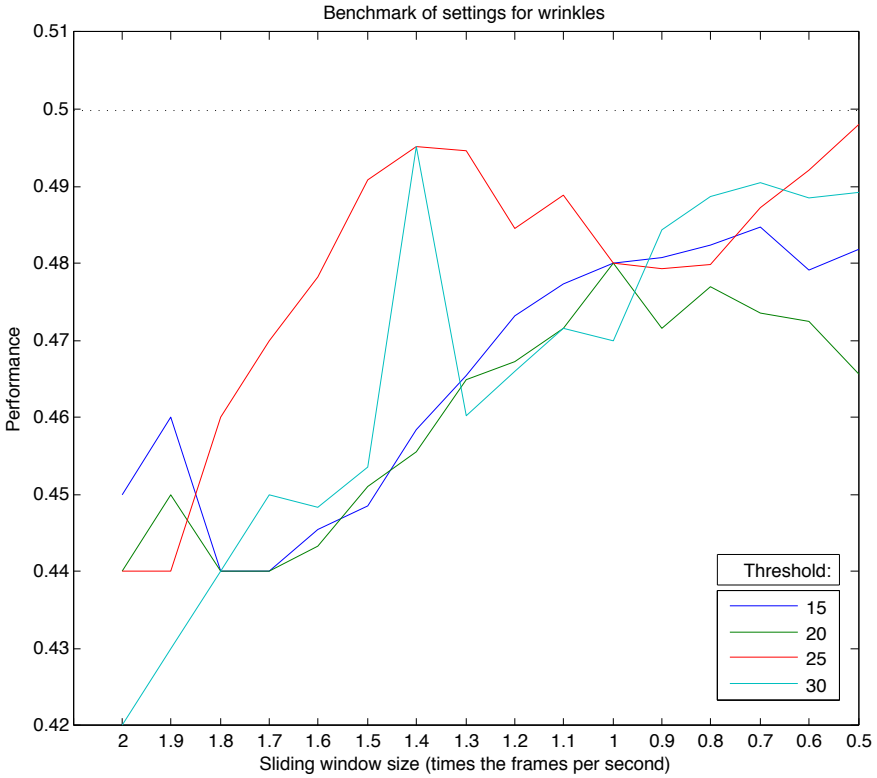
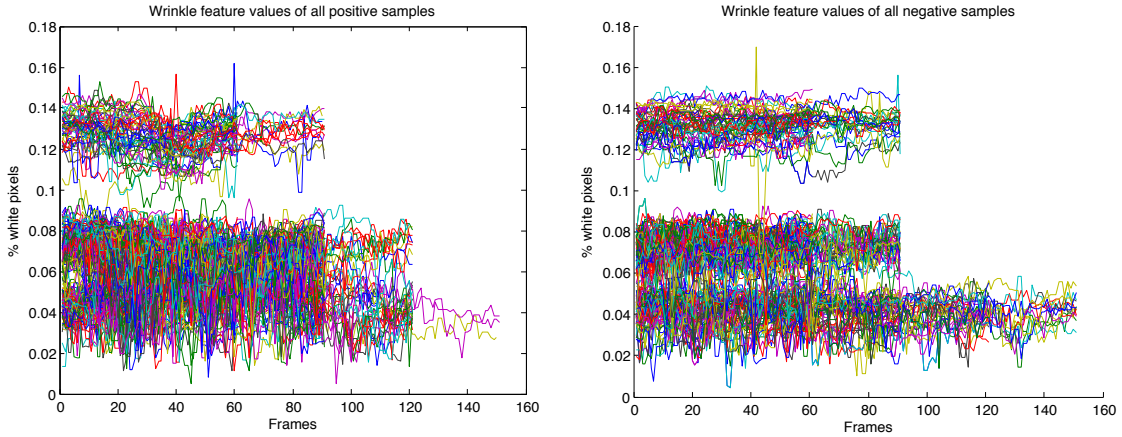


Figure 25: The plot of the benchmark for the wrinkle features with its settings. The dotted line indicates chance level.

The performance for the four settings is plotted in Figure 25 and shows that there is no setting where the performance reaches above the chance level. The raw values of the wrinkle feature produced by the feature extractor are plotted in Figure 26. An explanation why the performance is bad with this feature enabled is due to the large overlap of the values of the positive samples and the negative samples. With this method of extracting wrinkles, there is too little information on which the system can train and distinguish the positive samples from the negative samples. The histogram in Figure 27 shows that both

positive and negative samples have a large overlap in all occurring values. Therefore, settings (threshold or window size) with a good performance cannot be found.

Concluding, this feature on its own is not reliable enough for detecting facial tics of GTS patients.



(a) (b)

Figure 26: Figure 26a and Figure 26b depict respectively all positive and negative samples using a threshold of 30.

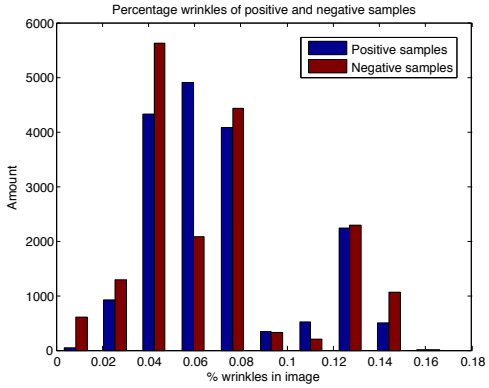


Figure 27: This histogram depicts the same values as Figure 26a and Figure 26b.

5.6 Combining features

Features on their own are likely to not perform that well. In this section combinations of features are benchmarked, analyzed for their performance and compared with single features. The settings resulting in the best performance for each feature as concluded in the sections above are chosen in the benchmarks in this section. Whether combining features with these settings also result in the best combined performance is left open for future research. In this section the performance of each combination is also plotted in a receiver operating characteristic (ROC) space to visualize the false positive rate (FPR) and true positive rate (TPR) ratio.

The first set of benchmarks are the configurations with two features combined as shown in Table 4.

	Template matching	Pupil gradient	Motion	Wrinkle	Distances
Template matching		0.90 (1)	0.90 (2)	0.90 (4)	0.90 (7)
Pupil gradient			0.51 (3)	0.48 (5)	0.53 (8)
Motion				0.50 (6)	0.63 (9)
Wrinkle					0.56 (10)
Distances					

Table 4: The performance of each combination is shown in this table. The features are commutative and therefore the bottom part of the table is grayed out. The number between parentheses correspond to the numbers in Figure 28.

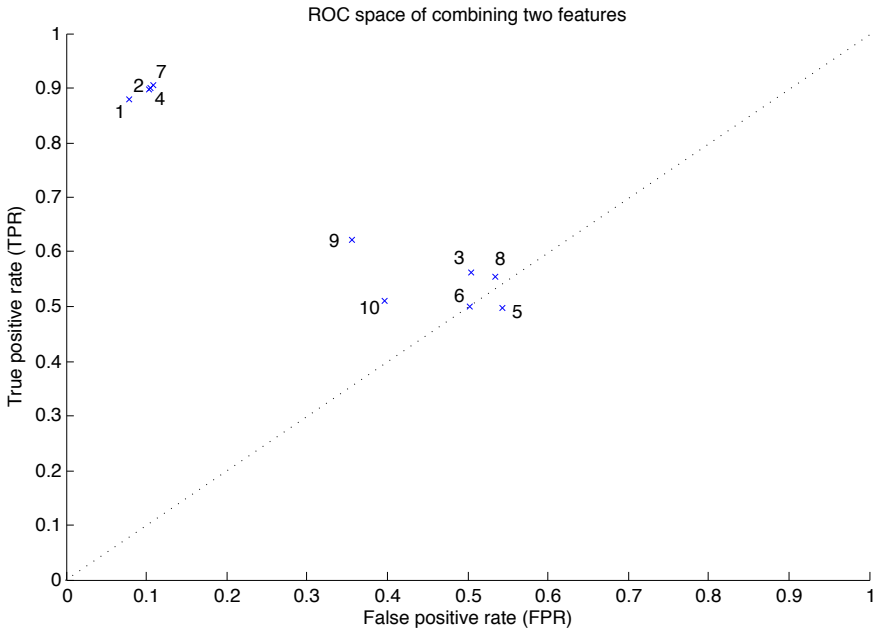


Figure 28: The FPR are plotted against the TPR of the ten configurations. The numbers correspond to the numbers in parentheses in Table 4. The dotted line indicates chance level.

Figure 28 depicts the ROC space by plotting the FPR against the TPR of the ten configurations. The combinations with blink detection using template matching in it (combinations 1, 2, 4 and 7) perform the best and have the lowest false positive rate. Even though the badly performing features are added to the feature vector, the system still performs acceptable. Furthermore, the combinations containing only badly performing features (combinations 3, 5 and 6) are a poorly performing combination and the ROC space also indicates that the FPR is high in those cases.

The second set of benchmarks are the combinations with three features combined. These are shown in Table 5.

Similar to the previous ROC space, the ROC space in Figure 29 plots the FPR against the TPR of the ten configurations of Table 5. The same conclusions can be drawn with three features as with two

#	Features	Performance
1	Template, Pupil, Motion	0.90
2	Template, Pupil, Wrinkle	0.90
3	Template, Pupil, Distance	0.90
4	Template, Motion, Wrinkle	0.90
5	Template, Motion, Distance	0.90
6	Template, Wrinkle, Distance	0.90
7	Pupil, Motion, Wrinkle	0.50
8	Pupil, Motion, Distance	0.55
9	Pupil, Wrinkle, Distance	0.52
10	Motion, Wrinkle, Distance	0.63

Table 5: The performance of each combination consisting of three features is shown in this table.

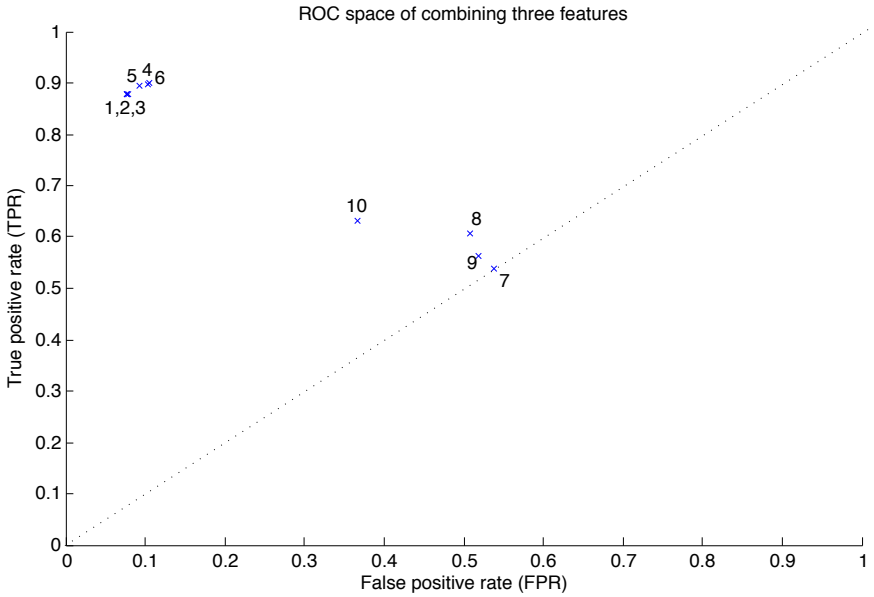


Figure 29: The FPR are plotted against the TPR of the ten configurations. The numbers correspond to the numbers in Table 5. The dotted line indicates chance level.

features; the configurations with blink detection using template matching in it (combinations 1, 2, 3, 4, 5 and 6) perform the best and have the lowest false positive rate. Furthermore, the motion feature and the distance feature in one combination increase the performance and show a lower false positive rate compared to the other configurations.

The third set of benchmarks consists of combinations with four features combined. The performance of these combinations are displayed in Table 6.

#	Features	Performance
1	Template, Pupil, Motion, Wrinkle	0.90
2	Template, Pupil, Wrinkle, Distance	0.90
3	Template, Pupil, Motion, Distance	0.90
4	Template, Motion, Wrinkle, Distance	0.90
5	Pupil, Motion, Wrinkle, Distance	0.54

Table 6: This table shows the performance of each combination consisting of four features.

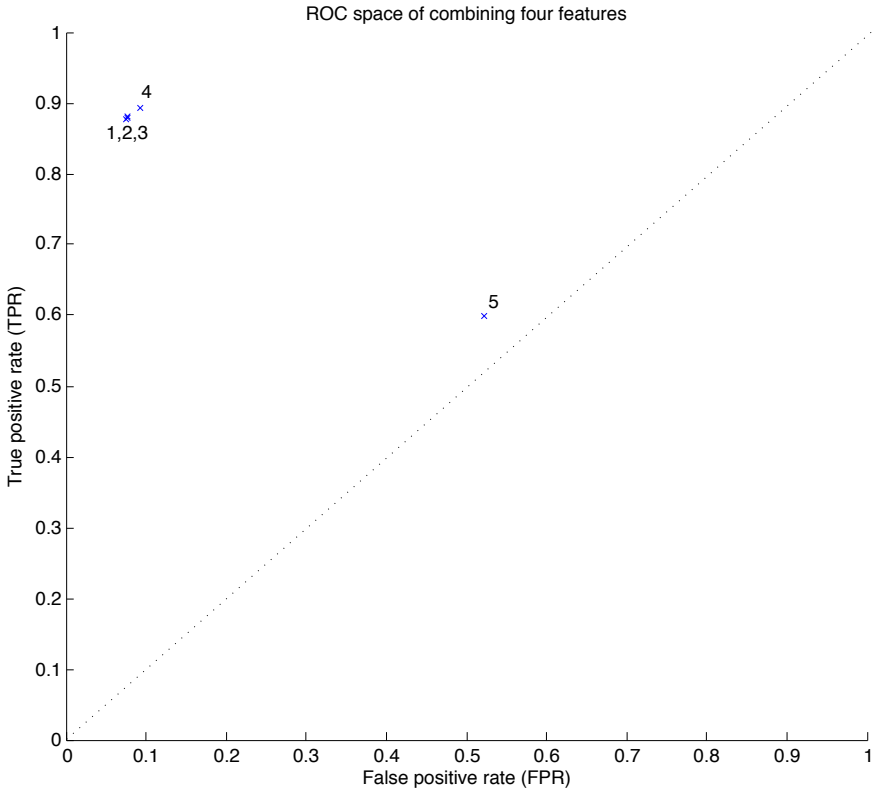


Figure 30: The FPR are plotted against the TPR of the five configurations and the numbers correspond to the numbers in Table 6. The dotted line indicates chance level.

Comparable to the ROC space in Figure 28 and Figure 29 are the FPR/TPR ratios of the five configurations in Table 6. These ratios are depicted in Figure 30. The conclusions drawn earlier are applicable here as well, because the performance and the ratio has not changed after adding an additional feature.

The last benchmark contains only one configuration, namely all features combined. The performance of this configuration is 0.9009 and has a FPR of 0.0752 and a TPR of 0.8792. Thus, with all features enabled it performs tha same as configurations with only three features and in the ROC space it is situated in the same place as the numbers 1, 2 and 3 in Figure 29. Concluding, only specific features influence the performance and adding more features to the feature vector does not increase the performance.

6 Discussion

In this section the strengths, future improvements for the tics detection system and also design choices during the development of the system are discussed. Finally , design guidelines are listed to adapt the system to be used in an application for psychotherapeutic feedback.

6.1 Strengths

One of the strengths of this tics detection system is the modular structure where each feature can be replaced by another. This way when the results of a feature indicate the feature is performing poor, the feature can either be modified or removed. Furthermore, each feature can individually be evaluated as described in Chapter 4.2.

Another strength of this system is that it can be used in real-time. All methods are built to be fast in processing time. Therefore, this system can be combined with a webcam to process video material in real-time as will be further discussed in Section 6.4.

6.2 Future improvements

During the development of the tics detection system trade-offs have been made and there are parts of the system which need to be improved to further boost the performance of the system. In this section these further improvements will be discussed.

6.2.1 Face detection

The first step of the pipeline is face detection as described in Chapter 3. During the development of the tics detection system two methods of detecting faces were explored. The first method was using a cascade classifier to detect faces in an image. Given the constraints for the data acquisition as described in Section 2.1, the method worked as intended. However, when these constraints were lifted and the head can move freely, detecting the face became harder.

Therefore, the second method was evaluated, which involved tracking the face once it had been detected. The face tracker uses the CAMshift (Continuously Adaptive Meanshift) algorithm to track the face[4]. An advantage for using CAMshift is that the face can vary in size during the recording and the rectangle that encloses the face resizes to the suitable size. With the face tracker the face detection increased in speed, but was not compatible with the feature extractor. There was a trade-off whether to post-process the ROI so that the feature extractor could use the output of the face tracker or to use the first method in combination with the constraints for the data acquisition. Thus, improving the face detection so that the tic detection system can work with input the first method could not detect is a possible future improvement.

6.2.2 Blink detection: Template matching

The blink detection feature using template matching currently uses a static threshold for each eye. It is possible to adjust the thresholds with the application shell around the tics detection system as shown in Chapter 4.2. However, these thresholds are applied to all data samples and this means that across subjects the same threshold is used. A more robust method would be by training or adjusting the thresholds per subject and therefore increasing the performance of the tic detection system. A calibration process per subject could be a possible way of adjusting the thresholds. However, the calibration process is currently not implemented in the system and is left as a possible future improvement.

6.2.3 Alternative approaches for features

During this study a couple of features have been benchmarked of which some implementations had a relatively low performance. These performances do not necessarily have to mean that these features have to be discarded. The implementation could be tweaked or a different approach can be chosen to increase the performance. What this different approach may be, is left open for future research, but for a couple of features a suggestion is described here.

Motion For the motion feature there are multiple methods for detecting motion. In this study motion detection using a difference between two images is used, but the literature also shows methods such as optical flow to detect motion[6][2].

Wrinkle In this study the wrinkle feature uses the Canny edge detector in order to detect edge pixels and the feature value is the number of edge pixels in a certain ROI. An alternative approach to this feature is using the change of the number of edge pixels of two subsequent images as the feature value. This approach is closely related to the motion feature used in this study.

Furthermore, aside from the reliablility of the system, the approaches of the features themselves should be tested if they are reliable as well. However, this is left open for future research.

Unrelated to the implementation of the methods for each feature, an optimization that related to the the processing time is using multi-threading. Currently the system is using only one thread where each feature is extracted in serial. A small optimization to the feature extractor is running each feature on its own thread and decreasing the processing time even more.

6.2.4 Distinguishing tics

Currently, the system detects whether there has been a tic or not. This does give insight into how many tics the GTS patient has, but does not show which tics have occurred during the samples. A possible future improvement is extending the current system so it supports multi-class classification.

Additionally, a confidence value of how confident the system is at classifying the tics can be added as well. Confidence values can be used to improve the feedback towards the user, for instance samples classified as a tic with a low confidence value could be ignored or having a weight to each tic.

6.3 Design choices

The data acquisition was described in Section 2.1 and more specifically the manner the data set was built. The data set consists of an equal number of positive and negative samples, whereas the tics do not occur as frequent as non-tics at GTS patients. This factor has not been taken into account and whether this influences the classifier is unknown. Furthermore, the duration of a tic is based on studying recordings of GTS patients as the literature does not show results of how long a tic lasts in general. Therefore, the system needs to be tested using a dataset consisting of GTS patients to see if the assumptions used in designing the system are applicable to to be tested dataset.

6.4 Design guidelines

The tics detection system is a prototype to explore whether distinctive features used by experts can be used in a system using computer vision and machine learning technique. However, this system can be adapted to be used in an application for therapeutic feedback. The system is still subject to change and the future improvements listed in Section 6.2 are recommended to explore in order to increase the robustness and performance of the system.

6.4.1 Applications

With this prototype an application can be made to harness the capabilities of this system and use it for therapeutic purposes. Some of these possible future applications are described here. The data that has been used during training and testing of the system is constrained, as described in Section 2.1. It is recommended to also use these constraints in the application which uses this system.

An application could feed a webcam stream into the tics detection system and based on the response of the SVM the application could return feedback to the user. Aside from the future improvements to boost the performance, there is little change needed in order to use a webcam as an input. Instead of using a video sample as an input for the feature extractor, a slice of the webcam stream — of the past x frames — could be used as an input. Based on the results described in Section 5.1, a suitable number of frames can be used.

- A first type of application could be used to coach the user with exercises where the user has to suppress tics. The feedback of the application indicates whether a tic has occurred and the goal of the user is to minimize the number of tics the system has detected.

- Another type of application would be for diagnostic purposes where the user sits in front of a webcam and the application counts the number of detected tics and checks whether the user has improved or not.
- A last type of application would be a tic filtering tool for counting, where the footage that does not contain a tic is skipped. Applying the system this way reduces the amount of data for experts to manually count tics.

When implementing a system which can be used to coach GTS patients using the feedback, it is important to look at the false positives and the false negatives the system produces. The FP and FN have to be minimal for all features in order to reduce frustration of the user of the system. This frustration arises when the feedback indicates a tic has occurred while it was not the case; a false positive has occurred. On the other hand, if the system ignores tics while they do occur, then it was a false negative. In that case the number of tics is incorrect and the result might not reflect the reality. A trade-off has to be made which result is preferred. For a coaching purpose it might be recommended to tweak the features in such a way that the false positives are reduced so the user is not punished for a tic which has not occurred. To reflect the preference of either a low FP or FN, the SVM could be alternated using weights to add a bias.

7 Conclusion

This research explored what prominent domain-specific features of facial tics used by experts there are. Four different features have been benchmarked; the blink, facial landmark distances, motion and wrinkles feature. For each feature a method for extracting a metric has been described based on predefined requirements.

Furthermore, this project explored if and how they can be used for automating facial tics detection. The results show that there are indeed domain-specific features of facial tics, but that not all methods of extracting a feature from a recording are suitable and/or robust. In particular the blink feature using the template matching method worked well. The facial landmark features does not perform that well on its own, but can be used in combination with other features to improve the performance. The other two features — motion and wrinkles — did not work that well and more research is needed to improve the method of extracting a metric for these features. The methods of the badly performing features can be improved to boost the performance of these features and improvements have been suggested as future research.

After evaluating the individual features and their performance, all combinations consisting of two, three, four and five features were tested and their performance were analyzed. The combinations containing the features which performed well on their own also performed well even though badly performing features were included as well. The performances of the various combinations were the same regardless of the number of features, e.g. the combination of five features performed just as well as a combination of three features sharing the same features.

References

[1] Nathan H Azrin and Alan L Peterson. Reduction of an eye tic by controlled blinking. *Behavior Therapy*, 20(3):467–473, 1989.

[2] John L Barron, David J Fleet, and Steven S Beauchemin. Performance of optical flow techniques. *International journal of computer vision*, 12(1):43–77, 1994.

[3] Michel Bernabei, Giuseppe Andreoni, Martin O Mendez Garcia, Luca Piccini, Federico Aletti, Marco Sassi, Domenico Servello, Mauro Porta, and Ezio Preatoni. Automatic detection of tic activity in the tourette syndrome. In *Engineering in Medicine and Biology Society (EMBC), 2010 Annual International Conference of the IEEE*, pages 422–425. IEEE, 2010.

[4] Gary R Bradski. Real time face and object tracking as a component of a perceptual user interface. In *Applications of Computer Vision, 1998. WACV'98. Proceedings., Fourth IEEE Workshop on*, pages 214–219. IEEE, 1998.

[5] Kai Briechle and Uwe D Hanebeck. Template matching using fast normalized cross correlation. In *Aerospace/Defense Sensing, Simulation, and Controls*, pages 95–102. International Society for Optics and Photonics, 2001.

[6] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In *Computer Vision-ECCV 2004*, pages 25–36. Springer, 2004.

[7] Hennie Brugman and Albert Russel. Annotating multi-media/multi-modal resources with elan. In *LREC*, 2004.

[8] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, (6):679–698, 1986.

[9] Phillip B Chappell, Maureen T McSwiggan-Hardin, Lawrence Scahill, Michelle Rubenstein, David E Walker, Donald J Cohen, and James F Leckman. Videotape tic counts in the assessment of tourette's syndrome: stability, reliability, and validity. *Journal of the American Academy of Child & Adolescent Psychiatry*, 33(3):386–393, 1994.

[10] Michael Chau and Margrit Betke. Real time eye tracking and blink detection with usb cameras. Technical report, Boston University Computer Science Department, 2005.

[11] Diane F Harcherik, James F Leckman, Jill Detlor, and Donald J Cohen. A new instrument for clinical studies of tourette's syndrome. *Journal of the American Academy of Child Psychiatry*, 23(2):153–160, 1984.

[12] Roger Kurlan and Michael P McDermott. Rating tic severity. *Neurological Disease and Therapy*, 15:199–199, 1993.

[13] JPl Lewis. Fast template matching. In *Vision Interface*, volume 95, pages 15–19, 1995.

[14] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Pattern Recognition*, pages 297–304. Springer, 2003.

[15] Brook A Marcks, Kristoffer S Berlin, Douglas W Woods, and W Hobart Davies. Impact of tourette syndrome: a preliminary investigation of the effects of disclosure on peer perceptions and social functioning. *Psychiatry: Interpersonal and Biological Processes*, 70(1):59–67, 2007.

[16] Davide Martino, Andrea E Cavanna, Mary M Robertson, and Michael Orth. Prevalence and phenomenology of eye tics in gilles de la tourette syndrome. *Journal of neurology*, 259(10):2137–2140, 2012.

[17] Carlos H Morimoto and Marcio RM Mimica. Eye gaze tracking techniques for interactive applications. *Computer Vision and Image Understanding*, 98(1):4–24, 2005.

[18] Hugh Rickards. Tics and fits. the current status of gilles de la tourette syndrome and its relationship with epilepsy. *Seizure*, 4(4):259–266, 1995.

[19] Caifeng Shan, Shaogang Gong, and Peter W McOwan. Robust facial expression recognition using local binary patterns. In *Image Processing, 2005. ICIP 2005. IEEE International Conference on*, volume 2, pages II–370. IEEE, 2005.

[20] Y-I Tian, Takeo Kanade, and Jeffrey F Cohn. Recognizing action units for facial expression analysis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(2):97–115, 2001.

[21] Fabian Timm and Erhardt Barth. Accurate eye centre localisation by means of gradients. In *VISAPP*, pages 125–130, 2011.

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

[23] Vladimir N Vapnik. Statistical learning theory. 1998.

[24] Cara Verdellen, Jolande van de Griendt, Linda van Heeswijk, and Marc Verbraak. Gedragstherapie: eerste keuzebehandeling voor ticstoornissen. *Kind adolescent praktijk*, 8(2):58–67, 2009.

Appendices

A Clipping

The clipping scripts that were needed during the data acquisition phase are listed below.

A.1 Clipping the positive samples

```
import sys
import math
import os

# Usage: pos.py inFile annotationFile beginLine
if len(sys.argv) >= 3:
    inFile = sys.argv[1]
    annotationFile = sys.argv[2]
    startAtLine = 1

    if len(sys.argv) == 4:
        startAtLine = int(sys.argv[3])

    f = open(annotationFile, 'r')
    currentLine = 1
    for line in f:
        if (currentLine == startAtLine):
            start, end, _ = line.split('\t');
            start = int(math.floor(float(start)))
            end = int(math.ceil(float(end)))
            if start >= end:
                end = end + 1
            command = "osascript trim.scpt " + inFile + " " +
                str(startAtLine).zfill(3) + " " + str(start) + " " +
                str(end)
            print command
            os.system(command)
            startAtLine += 1
        currentLine += 1
else:
    print "Wrong number of arguments"
```

A.2 Clipping the negative samples

```
import sys
import math
import os
import random
import itertools

random.seed(1234)

# Usage: neg.py inFile annotationFile numSamples beginSample
if len(sys.argv) >= 4:
    inFile = sys.argv[1]
    annotationFile = sys.argv[2]
    maxNumSamples = int(sys.argv[3])
```

```
    startAtSample = 1

    if len(sys.argv) == 5:
        startAtSample = int(sys.argv[4])

    f = open(annotationFile, 'r')

    minSampleSize = 2
    maxSampleSize = 3
    previousEnd = -1
    gaps = []

    for line in f:
        start, end, _ = line.split('\t');
        start = int(math.floor(float(start)))
        end = int(math.ceil(float(end)))
        if start >= end:
            end = end + 1

        if previousEnd < 0:
            previousEnd = end
        else:
            if previousEnd < start:
                if (start - previousEnd) == 1:
                    # good candidate
                    gaps.append((previousEnd, start))
                elif (start - previousEnd) == 2:
                    # left two OR right two
                    gaps.append((previousEnd, previousEnd + 1))
                    gaps.append((previousEnd + 1, start))
                else:
                    for pair in itertools.combinations(range(previousEnd, start), 2):
                        if (pair[1] - pair[0]) <= maxSampleSize and
                        (pair[1] - pair[0]) >= minSampleSize:
                            gaps.append(pair)
            previousEnd = end
    print startAtSample
    if maxNumSamples <= len(gaps):
        selection = random.sample(gaps, maxNumSamples)
        currentLine = 1
        for sample in selection:
            if (currentLine == startAtSample):
                command = "osascript trim.scpt " + inFile + " " +
                        str(startAtSample).zfill(3) + " " + str(sample[0]) + " " +
                        str(sample[1])
                print command
                os.system(command)
                startAtSample += 1
            currentLine += 1
    else:
        print "The max number of samples (" + str(maxNumSamples) + ")
                exceeds the selection size (" + str(len(gaps)) + ")"
else:
    print "Wrong number of arguments"
```

A.3 Video trimming

```
on run args
        set inFile to POSIX file (item 1 of args)
        set outFile to (item 2 of args) as string
        set startTime to (item 3 of args) as integer
        set endTime to (item 4 of args) as integer

        tell application "QuickTime Player"
                open inFile
                activate
                delay 1
                if not (exists document 1) then error number -128

                set the current time of document 1 to startTime
                trim document 1 from startTime to endTime
        end tell

        tell application "System Events"
                keystroke "s" using {command down}
                delay 1
                keystroke outFile
                delay 1
                key code 36
                delay 1
                keystroke "w" using {command down}
                delay 1
        end tell
end run
```

B Software dependencies

The full list of dependencies for compiling the source code is listed below.

Name	Description	URL
Boost	Collection of libraries for functionality that is not (yet) available in the standard version of C++.	`http://www.boost.org/`
OpenCV	A computer vision library with a strong focus towards real-time applications.	`http://opencv.org/`
Flandmark	An open-source implementation of a facial landmark detector.	`http://cmp.felk.cvut.cz/~uricamic/flandmark/`
EyeLike	A webcam based gaze tracking system.	`https://github.com/trishume/eyeLike`