

Video Segmentation and Indexing Report

Roxana Danila, Benjamin Grabham, Charlie Paucard,
Jasper Pult, Agnieszka Szefer

January 2013

Abstract

Efficient handling of video data highly depends on the availability of effective indexes. Currently, most of this process is being done manually. In this way, it becomes a laborious and time consuming task, being unfeasible for large video databases. The project brief posed the challenge to develop a system allowing users to segment videos and index them accordingly, based on face recognition. During our research, we could not find any applications providing an efficient indexing technique, but a considerable amount of work was already put into the area of face recognition.

Our solution, SIS (Segmentation and Indexing Software) means to fill this gap in an automated manner. It allows the user to input a video and choose segmentation parameters. The software segments the video into scenes depending on the input parameters and provides means for the user to browse generated segments as well as the faces detected in them. Some of the algorithms used are PCA, K-Means and Means-Shift.

Contents

1	Introduction	4
1.1	Motivation	4
1.2	Project goals	4
1.3	Result	4
2	Design and implementation	5
2.1	Architecture	5
2.2	Language choice	5
2.3	Front end	6
2.3.1	Desktop versus Web	6
2.3.2	Development of the interface design	6
2.3.3	Graphical user interface	9
2.3.4	Technical challenges	9
2.3.4.1	Progress bar	9
2.3.4.2	Play button	10
2.4	Back end	10
2.4.1	Face detection	10
2.4.2	Face recognition	13
2.4.2.1	Histograms	13
2.4.2.2	Principal Components Analysis (PCA)	13
2.4.2.3	Other algorithms	15
2.4.3	Face Clustering	15
2.4.3.1	Tarjan algorithm on neighbours' graph	15
2.4.3.2	K-means algorithm	16
2.4.3.3	Means-Shift algorithm	18
2.4.3.4	Hybrid algorithm	19
2.4.4	Segmentation	19
2.4.4.1	Homemade algorithm	20
2.4.4.2	Other algorithms	20
2.4.5	Indexing	21
2.4.6	Technical challenges	21
2.4.6.1	Understanding the mathematical concepts	21
2.4.6.2	Finding a good clustering algorithm	22
2.4.6.3	Accuracy versus efficiency	22
3	Evaluation	22
3.1	Testing	22
3.1.1	Test-driven-development	22
3.1.2	Unit testing	23
3.1.3	Usability testing	23
3.2	Validation of the product	24
3.2.1	Efficiency	24
3.2.2	Accuracy	25
3.2.3	Tradeoffs	25
3.3	Main achievements	26
3.3.1	Effort	26
3.4	Reflections	27
3.4.1	Plans vs. Achievements	27

4	Conclusions and future extensions	28
4.1	Things learned	28
4.2	Future extensions	29
4.2.1	Segment browser	29
4.2.2	Web interface	29
4.2.3	Indicator of remaining time	29
4.2.4	Segmentation options	29
4.2.5	Face recognition algorithm	29
4.2.6	Clustering algorithm	29
4.2.7	Indexing	30
5	Project management	30
5.1	Collaboration	30
5.1.1	Pair programming	31
5.1.2	Workflow	32
5.1.3	Group organisation	32
5.2	Reflections	33

1 Introduction

1.1 Motivation

With the rise of YouTube and other video platforms, sharing videos has become as common as sharing pictures. The increasing amount of videos on the web has given rise to another problem. How to search through and order an incoherent mass of videos? It cannot be done in the same way as searching text - the information is not written out, it is graphically displayed or spoken. The usual solution for this is to leave tagging to the user, resulting in weak or irrelevant tags for the better part of the videos available on the internet.

Neatly putting videos into boxes via tagging is only a solution to part of the problem. Just like a text document, when a video is produced, whether by an average home camera user or by a big movie studio, it can often be split into multiple scenes. These are the chapters in a DVD, or the links to a specific time in the video often found on YouTube. Again, these splits are done manually, leading to ad-hoc solutions on video sharing sites and loss of time in the film industry.

1.2 Project goals

In an effort to make a step towards solving these issues, the product has been designed around those two key features: segmentation and indexing. More precisely, it should be an aid - if not a fully automated solution - to tagging and delimiting scenes. In order to create a product as simple as possible, these two requirements were fulfilled within a single piece of software.

The idea presented itself as working with the actors of the video. Indexing could be done by tagging the video with the names of the actors appearing in it. Similarly, a segment split can be made every time a group of actors are replaced on the screen by another. In fact, by limiting the detection of actors to face detection, it is possible to uniquely identify each actor, and tagging simply becomes putting a name to a face.

By providing automated mechanisms to tag and index videos, the product would then ideally allow anyone, from the home user to the movie producer, to generate splits and tags in the video. This would otherwise be a time consuming process. In more moderate terms, the program would aim to help the user in defining those splits and tags.

1.3 Result

The outcome of our project is a desktop application that can be run on most major platforms. Users can load a video file and select what kind of split they want to perform: every x seconds, on black frames, or on faces. The first option lets users specify a time interval for segments. Black frames are a common way in films to jump from one scene to another, by fading out one picture to black before showing the new scene. This segmentation option therefore simply follows the black frames used by directors and producers to split the video into scenes, which returns quite useful results.

The main focus of the project was laid on the option to split a video based on faces appearing in it. It first recognises and clusters the faces in the video. Scenes are then determined by analysing how the number and composition of faces in each frame change over time. For example if first person A is seen speaking to person B, followed by person B speaking to person C, these scenes will be split into two sections due to the change of faces on screen.

After performing the segmentation, a list of the produced segments is presented, which can be viewed within the application. In addition, there is an overview of all faces detected in the video, with the option to browse and view segments indexed by individual faces appearing in them.

2 Design and implementation

2.1 Architecture

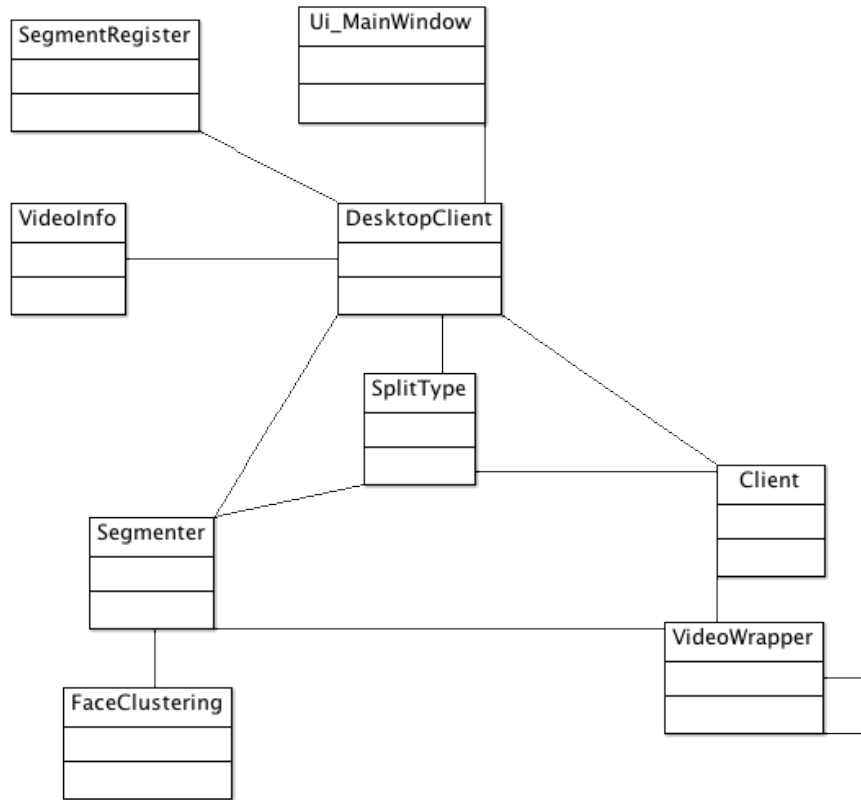


Figure 1: First interface version

View: UI_MainWindow contains the GUI.

Controller: Desktop Client controls interactions between model and view.

Model: SegmentRegister and VideoInfo hold information about the current state of the segmented video

Helper Classes: All remaining classes aid the Controller by providing functions like loading the video into a wrapper to work with it, segmenting it or clustering faces.

2.2 Language choice

The main implementation language used is Python. Due the focus being on video processing, the code greatly relies upon the OpenCV library. Consequently, a choice between the two programming languages that are most extensively supported had to be made: Python or C++. The main reason for choosing Python was the fact that some of the team members were very eager to learn this programming language, not having any experience with it before, while others wanted to gain

more advanced knowledge and expertise. Furthermore, Python is meant to be a rapid prototyping language with a very clear and readable syntax. Hence, implementing various approaches to problems could be faster in Python than in C++. Moreover, knowing that OpenCV libraries are written in C/C++ anyway, the performance gain of using C++ would not outweigh the benefits of using a rapid prototyping language like Python.

2.3 Front end

2.3.1 Desktop versus Web

The first decision that had to be made with regards to the front end of the application was whether it would be a web or a desktop application. The first two development iterations were spent on a web application. The main benefit of this was the fact that it could be easily executed in any browser and make use of existing browser features, for example to play videos. This implied users did not need to install anything on their local machines to execute the application, having access to it at any time, anywhere.

However, initial tests of the video segmentation through the web interface revealed that the process of iterating over the video and writing out segments to files took a considerable amount of time. Consequently a more extensive evaluation of the typical use case of the application followed. A person deciding to segment a video would most probably be doing this in a desktop environment, and would also be prepared to wait for at least a few minutes for the segmentation to complete.

On the other hand, users of a web application are not usually prepared to wait for too long and lose patience easily, because a browser interface is assumed to be quick. After this analysis the issue was also raised with the client, who confirmed the team's belief that a desktop application would be more suitable. The final product therefore takes the form of a desktop application, and is described in more detail in the following sections.

2.3.2 Development of the interface design

The first interface design consisted of a single view with options to load in a video, select the desired segmentation method and afterwards play the generated video segments:

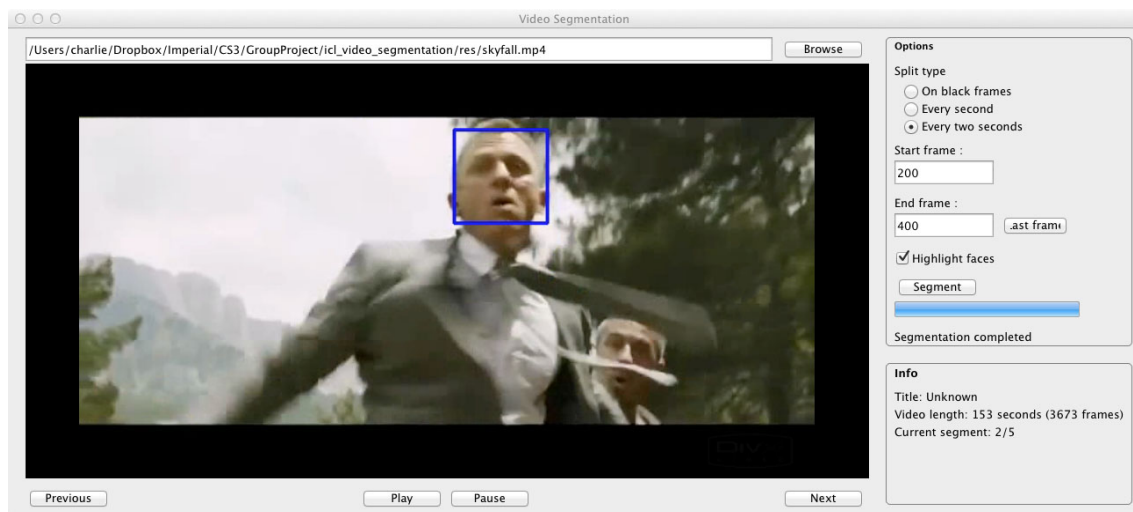


Figure 2: First interface version

After testing this interface with the client as well as some other user candidates, some substantial areas for improvements to the interface were discovered:

- A *Play Next* button to play the next element
- A *Cancel* button to abort the segmentation
- A split screen, giving an overview of the segments on one side and a video preview on the other
- *Save* and *Load* buttons to be able to save generated segments and reload them later without having to segment the same video again

In addition to these points it was decided to separate the loading of the video and selection of segmentation options from the viewing of the generated segments. The single window was replaced by two separate views, the second one following the first one upon successful segmentation. The initial version of the split screen featured a list of segments on the left of the window, with the option to select a segment by clicking on it, and the video preview on the right. Videos could now be navigated either by selecting them from the list, or by using the previous, next and play next buttons. The resulting interface is shown below:

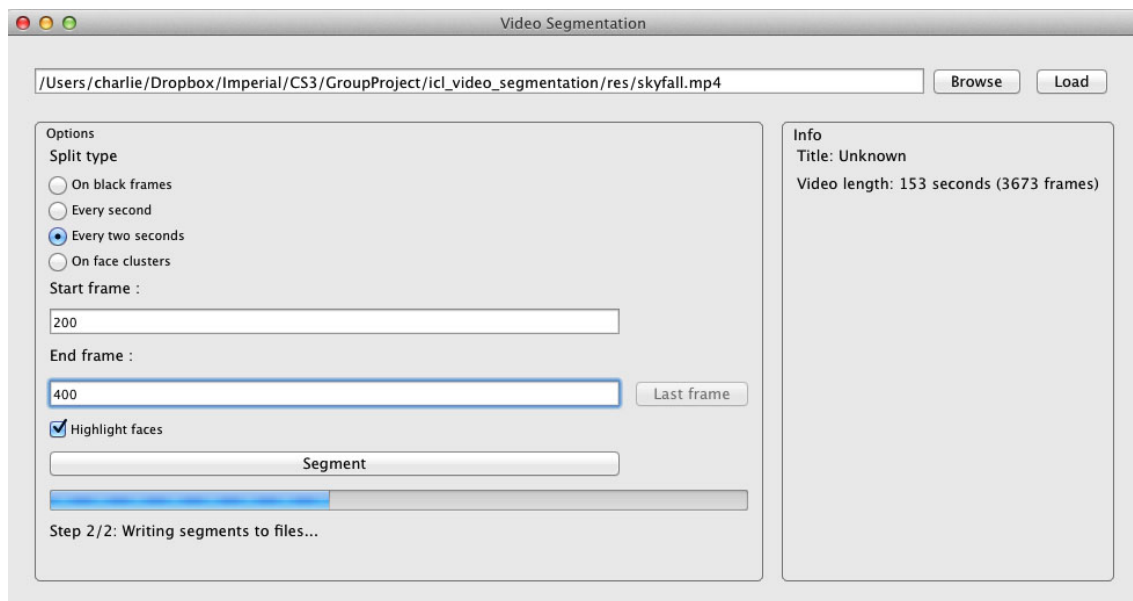


Figure 3: Second interface version, first view

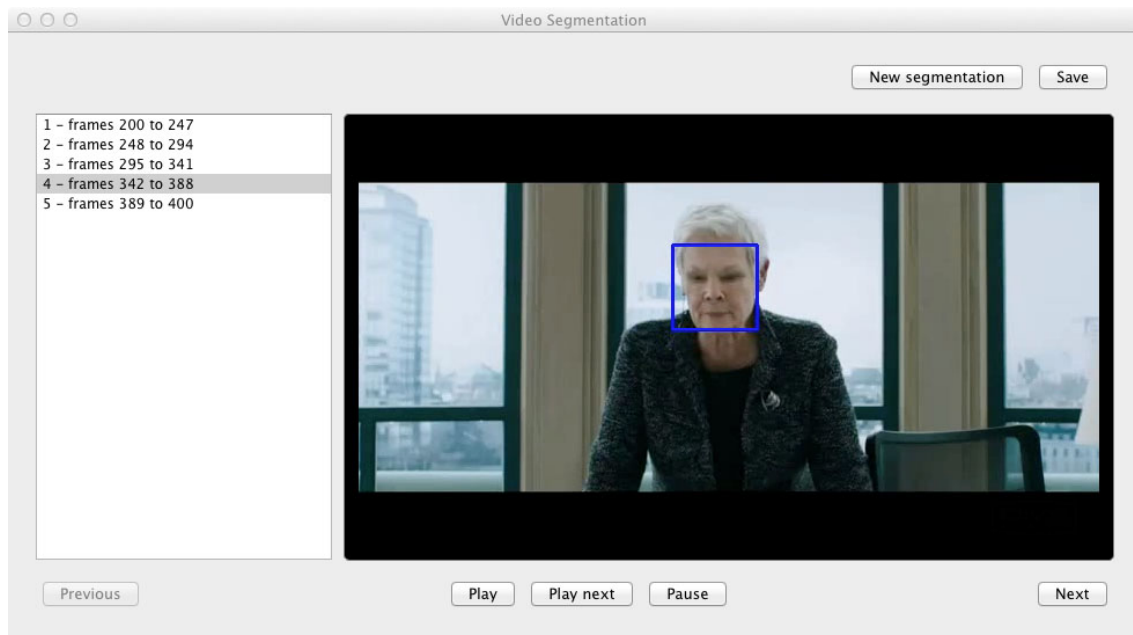


Figure 4: Second interface version, second view

The overall structure of this interface was satisfactory, so it was perfected in the following iterations. Below is the final version, including more fine grained segmentation options for the user to specify, as well as an overview of indexed faces in the second view.

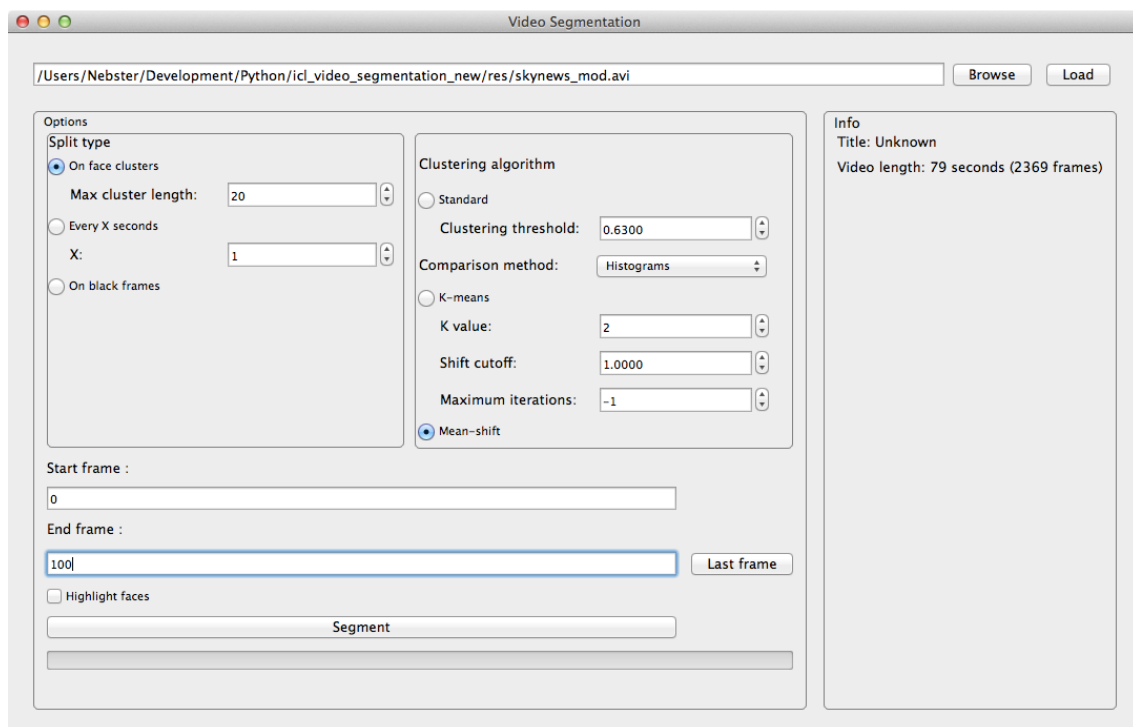


Figure 5: Final interface version, first view

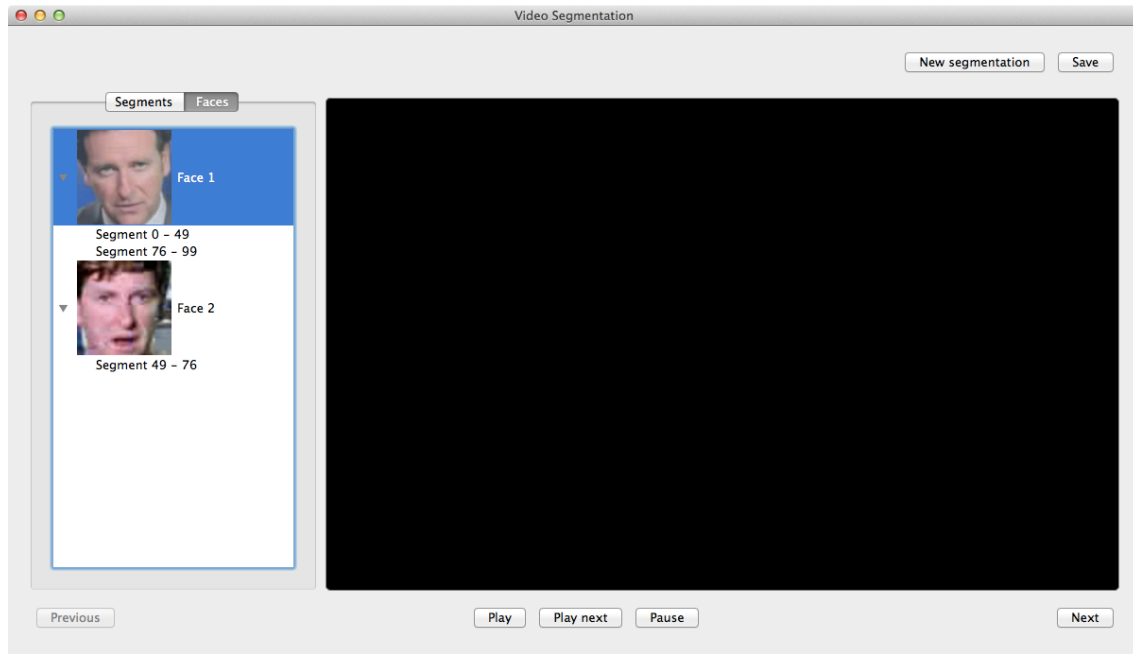


Figure 6: Final interface version, second view

2.3.3 Graphical user interface

To implement a graphical user interface with underlying functionality, the Model View Controller design pattern was employed.

The QT framework was used for the interface design, more precisely its Python binding PyQT. Reasons for choosing this framework were that some of the team members had used it before and knew it to be a well-rounded, reliable framework. Furthermore its overall look and feel compared to other frameworks were appealing. This native look is also kept consistent across different platforms, since QT supports all major platforms which is another bonus. The QT Designer furthermore proved a great what-you-see-is-what-you-get editor to quickly assemble an interface that could easily be accessed from supporting backend classes.

2.3.4 Technical challenges

2.3.4.1 Progress bar The first version of the GUI did not provide a progress bar, which meant that long waiting times for the video segmentation seemed even longer because there was no sense of progress and remaining time. Even worse, one could get the impression that in fact nothing was happening at all and the backend had crashed. The progress bar was therefore made the first priority in the second iteration including the desktop application.

The initial idea was to make use of multithreading. The segmentation process itself could be split off to a new thread to run in the background, sending notifications to the main thread to update the progress bar. A considerable effort was put into this approach, and a version which ran the segmentation on a separate thread as outlined above was implemented.

At first a new thread was started to deal with updates to the progress bar. The segmentation process itself was running on the main thread, sending update messages to the second thread. However, the QT framework did not handle other threads accessing its elements, and the entire application crashed every time the updater-thread tried to access the progress bar.

The conclusion was that the progress would have to be updated from the main thread, and therefore the functionality of the threads was swapped, the new thread now dealing with the

segmentation process, while the main thread was receiving update notifications for the progress bar. This solved the first problem, the application was not crashing anymore.

However, this approach generated a new problem, namely that the main thread would have to wait for the segmentation thread to finish before being able to process the generated segments. This meant that it was busy waiting, and the update notifications for the progress bar were processed all at once only after the segmentation thread had finished. Effectively the multithreading was therefore rendered useless, since there was no continuous update of the progress bar, and the same result could have been achieved by executing the segmentation sequentially in the main thread.

While thinking about a solution to overcome the busy wait issue, the team realised that in fact the aim could be achieved in a single thread, by giving the Segmenter object access to the progress bar and letting it update the bar on every segment iteration. The design was consequently changed in favour of this simpler solution, which unfortunately made the work put into the implementation of multithreading redundant.

2.3.4.2 Play button When testing the desktop GUI with the client, it was found that the Play button seemed to only work the first time it was used for any segment. To play the same segment again one had to go back and forward first.

The first idea was that this might be a timing issue, but waiting for a millisecond after loading in the video before playing it did not fix the error. The final solution was to reload the video on every click of the *Play* button.

2.4 Back end

The core of the application is the segmentation and indexing of videos based on the faces appearing in them. The main steps that were followed in achieving this are: video acquisition, face detection, recognition and clustering, and finally segmentation and indexing of the video based on the people identified.

At each stage several algorithms of different complexities were implemented. Undertaking research in the very first stage of the project helped to identify which algorithms could potentially be used to solve the problem before even designing the application. However, it was initially decided to implement simple algorithms and test their performance under various scenarios rather than focus on more advanced ones, which would have required a much stronger mathematical background. This approach served two purposes. Firstly, it was possible to test the entire infrastructure working together as early as possible in order to minimise the risk of not being able to deliver a fully functional product by the deadline. Secondly, it gave a good measure of progress and improvement as one could clearly see an increase in performance for the more complex algorithms implemented in later stages.

2.4.1 Face detection

In general, face detection can be regarded as a particular case of object-class detection, its main purpose being to find the locations and sizes of all objects belonging to a given class. For the purpose of this project the focus only lay on detection of frontal human faces because this is how most of the available libraries are implemented. There is extensive work being undertaken on newer algorithms that attempt to solve the more general and difficult problem of multi-view face detection, taking into account factors such as face appearance, lighting and pose. However, due to the scope of the project and limited timeframe, it was decided to use existing libraries such as OpenCV to compute the detection of faces.

OpenCV (Open Source Computer Vision) is a cross-platform open source library of programming functions for computer vision. Among other features, it handles all the memory automatically, allocating and deallocating it as needed. Furthermore because it deals a lot with image pixels that are often encoded in a compact form, 8 or 16 bit per channel, which will have a limited value range, it often uses saturation arithmetics in order to ensure the values are in the valid range (0, 255). Values out of bounds may be produced by operations such as colour space conversions, brightness/contrast adjustments, sharpening, complex interpolation and may affect further image analysis.

The main data type used is VideoCapture, which represents videos captured from files, together with the available functionalities for face detection. OpenCV's face detector uses a method published in 2001 by Paul Viola and Michael Jones, which combines four key concepts: simple rectangular Haar features, an Integral Image for rapid feature detection, the AdaBoost machine learning method and a cascaded classifier to combine many features efficiently.

In practice, a simple rectangular Haar-like feature is a pair of adjacent rectangles - one light and one dark - as shown in the picture below. The presence of such a feature is identified by subtracting the average dark-region pixel value from the average light-region pixel value and comparing the result with a certain threshold. The feature exists if the computed difference is above the given threshold.

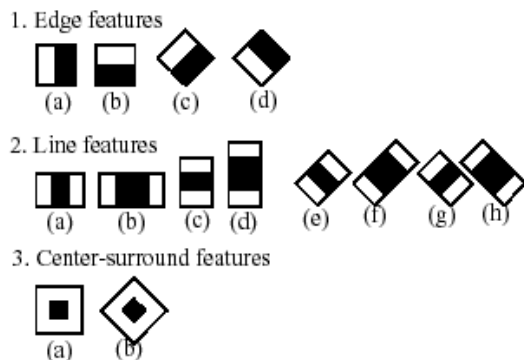


Figure 7: Examples of Haar features used in OpenCV.

An integral image makes it possible to calculate summations over image subregions rapidly. Viola and Jones use this technique to determine the presence or absence of a very large number of Haar features at every image location and at several scales efficiently. Given an image with width w and height h , the integral image of it will be $w+1$ pixels wide and $h+1$ pixels high, having the first row and column all zeros, and each of the other pixels having as value the sum of the pixels above it and to its left, starting at the top left corner. The image below illustrates this technique. After integrating, the pixel at coordinates (x, y) has the value of the sum of all pixels in the shaded region. In particular, in rectangle D, the sum of pixel values is: $(x_4, y_4) - (x_2, y_2) - (x_3, y_3) + (x_1, y_1)$.

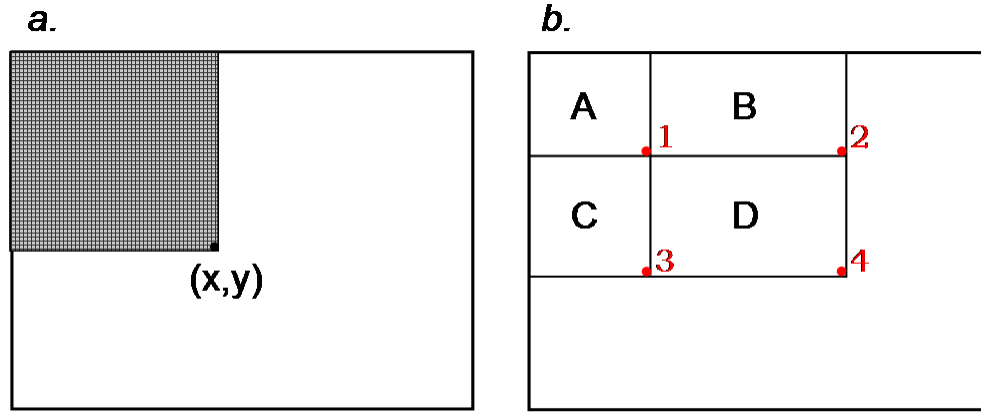


Figure 8: The Integral Image technique.

The AdaBoost machine learning algorithm helps selecting the specific Haar features to be used and sets the threshold levels. It combines many “weak” classifiers to create a “strong” one, where “weak” is defined as getting the right answer a little more often than random guessing. Putting together many of these weak classifiers, we obtain a result strong enough to achieve the correct solution through the combined force. The technique given by Viola and Jones uses a series of AdaBoost classifiers, combining them together as a filter chain, which is depicted in the image below. It is proven to be particularly efficient for classifying image regions.

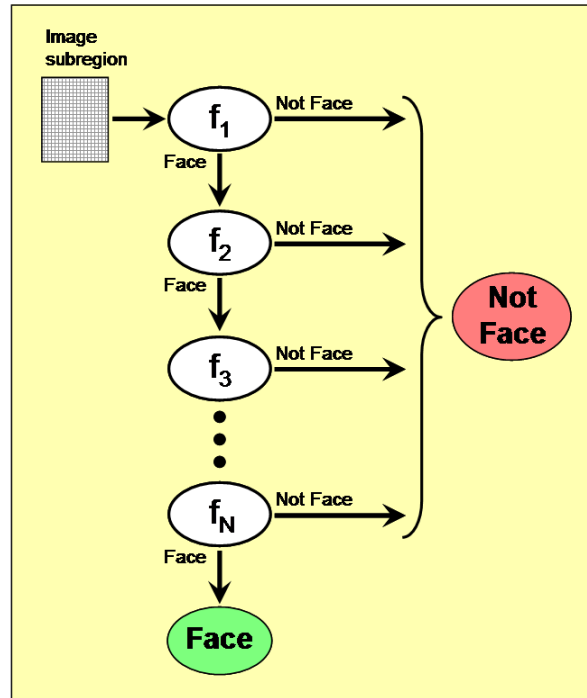


Figure 9: Filters Chain - the image subregions that manage to traverse the entire chain are classified as faces.

2.4.2 Face recognition

Strictly speaking, the task of face recognition is to distinguish the so called input signals, the image data in this case, into several classes, namely persons. The image data can be highly noisy (e.g. differing lighting conditions and pose). However, it is not completely random and despite the differences, one can still notice patterns that occur, such as eyes, nose, mouth, as well as the distances between them. When applying face recognition for video indexing, there is a limitation in terms of its generic applicability that has to be accounted for, as reported in [5]. Face recognition is proved to work in constrained environments, being more accurate when showing a frontal face close to the camera.

After a face is detected, it is extracted from the frame, normalised and stored in memory. A description of several algorithms that were investigated and implemented in order to recognise the extracted faces follows.

2.4.2.1 Histograms Histograms of different face images can be used for recognition purposes when taking into account the closest distance between them. For the project intensity histograms of gray scale face images were used, which are in fact vectors having each element representing the number of pixels at each different intensity value found in the image. An alternative to this could have been using colour histograms that would have mapped the image colours into RGB space. However, they would not have been able to deal as effectively with illumination variations, such that the same person under different lighting conditions would not be recognised.

Overall, the histograms are invariant to rotation and translation because they do not consider the spatial relationship between the pixels. Additionally, they are robust against occlusion and changes in camera viewpoint. For us, histograms proved to be effective for small initial data sets, but did not behave as expected for large ones due to the higher probability of having very different images with very similar intensity distribution and hence, similar histograms.

2.4.2.2 Principal Components Analysis (PCA) For face recognition the Eigenface method was also implemented. This is considered to be the first truly successful example in the field. It is based on the principal component analysis of face images, a way of identifying patterns in data by highlighting similarities and differences. It constitutes a powerful tool in data analysis since patterns can be hard to find, especially when handling high dimensional data that are impossible to represent graphically. In addition, it helps to compress the data by reducing its number of dimensions without much loss of information. These properties make PCA a suitable strategy for face recognition, capable of identifying variability in human faces. We will first discuss the mathematics behind the method, followed by a description of its implementation.

Mathematically speaking, PCA is a statistical procedure based on orthogonal linear transformations that projects data to a new coordinate system. In Euclidean space, the first principal component is the line that passes through the empirical mean of the data set and accounts for the greatest possible variance in the data set. The second principal component is calculated in the same way, independently from the first component, and gives the next highest variance. Continuing this procedure, the values in the last dimensions tend to highly correlate and dropping them implies a minimum loss of information. In other words, the Eigenfaces PCA approach transforms faces into a small set of essential characteristics, which are the main components of the initial set of images. Recognition is done by projecting a new image in the eigenface subspace and comparing its position in eigenface space with the position of known individuals.

In face recognition, the image manipulation is therefore less demanding using PCA due to its abilities for data reduction and interpretation. For example, it can be shown that a 100x100 pixel image containing a face can be very accurately represented using only 40 eigenfaces. If one considers the high-dimensional vector space of possible faces of human beings and calculate

the covariance matrix of its probability distribution, the eigenfaces can be defined as the set of eigenvectors derived from this matrix. A set of eigenfaces is generated as follows.

Data is retrieved by detecting faces in the video using the OpenCV library API. All detected faces are cut, resampled to the same pixel resolution and normalised. Each face image is represented as a two-dimensional NumPy array¹, which is flattened and stored as a row in a matrix. The PCA algorithm will use this matrix composed of N images to achieve dimension reduction as face images are highly redundant (i.e. image intensities of adjacent pixels are correlated and every individual has one mouth, one nose etc). The origin of the coordinate system must be moved to the mean of the data, so the average image is subtracted, computed as the column average from each element in each corresponding column. Below you can see samples of mean face images.



Figure 10: Mean faces

From the mean centred data vector we calculate the covariance matrix using the formula:

$$\Sigma = U^T U / (N - 1)$$

where U is the centred data representation and N is the number of data points. Intuitively, the covariance matrix generalizes the notion of variance - a measure of how far a set of numbers is spread out - to multiple dimensions. The eigenvectors of this covariance matrix are called Eigenfaces. They are the directions in which the images differ from the mean image. Each eigenvector has the same number of components as the original images, and thus can itself be seen as an image.

Lastly, the components that are needed in order to form a feature vector are chosen. A DxD covariance matrix will result in D eigenvectors, each representing a direction in the image space. The eigenvectors with largest associated Eigenvalue are kept. These Eigenfaces can now be used to represent both existing and new faces: We can project a new (mean-subtracted) image on the Eigenfaces and thereby record how that new face differs from the mean face. The Eigenvalues associated with each Eigenface represent how much the other images vary from the mean image in that direction. Information is lost by projecting the image on a subset of the eigenvectors, but the loss is minimised by keeping those Eigenfaces with the largest Eigenvalues. For instance, if we are working with a 90 x 120 image, then we will obtain 10,800 eigenvectors. In practical applications, most faces can typically be identified using a projection on between 100 and 150 Eigenfaces, so that most of the total of Eigenvectors can be discarded.

Once the PCA algorithm was implemented and tested, an analysis of its advantages and limitations was conducted. It was agreed that its simplicity and speed make it a good algorithm to use for the scope of this project. In addition, it is a completely non-parametric technique, such that any data set can be given as input, requiring no parameters or additional input and the output provided is unique and independent of the user.

On the other hand, its main limitations include the fact that it requires full frontal display of faces and it is not sensitive to lighting conditions or position of faces. In terms of further

¹homogeneous multidimensional array in NumPy library for Python

limitations, noteworthy points are that it is not scale invariant and that dimension reduction can only be achieved if the original variables were correlated. Otherwise PCA does nothing, except for ordering them according to their variance.

2.4.2.3 Other algorithms During initial research, the team also studied and analysed other algorithms for face recognition, including “Fisherfaces” and “Hidden Markov Models” (HMM).

It was found that for classification, one should aim to find the projection direction in which images belonging to different classes are separated maximally. One of the techniques used to achieve this goal is called Linear Discriminant Analysis (LDA), which finds the subspace representation of a set of face images. The resulting basis vectors are known as “Fisherface”. Studies [6, 7] have shown that for small datasets, the “Fisherface” approach performs better than the “Eigenface”. However, as the number of faces increases, both techniques produce almost the same percentage of truly recognised faces. Choosing between these two algorithms is a question of robustness versus simplicity. The “Fisherface” algorithm is hard to implement because of its computational complexity. And given that for a large dataset they have similar accuracy benchmarks, the “Eigenface” algorithm is a better choice in most of the cases, including the scope of this project.

Another interesting concept is video-based face recognition, which has significant advantages over the image-based recognition. It provides person specific dynamic characteristics, which may lead to easier recognition and the capability to learn and update the subject over time. In addition, it allows the retrieval of improved representations, such as 3D face model or super-resolution images. [8] shows how Hidden Markov Models (HMM) - a statistical model used to characterise the statistical properties of a signal - can be used to undertake video-based face recognition. Although the results obtained in [8] indicate a better performance than the majority of image-based recognition results, it would have been impossible to understand the mathematics behind the HMM and implement a similar approach for video-based recognition in the given timeframe.

2.4.3 Face Clustering

Face clustering is mainly a task of explorative data mining, concerning grouping a set of face images into clusters such that the ones in the same cluster are more similar to each other than to those in other clusters. Clustering is an iterative process of knowledge discovery or multi-objective optimisation that involves trial and failure in a sense that it will often require adjustment of parameters (e.g. distance function to use, density threshold or the number of expected clusters) until the result achieves the desired properties. There are various available techniques that differ significantly in their notion of what constitutes a cluster and how to efficiently find them. Which one is better to use depends entirely on the individual data set and intended use of the results.

2.4.3.1 Tarjan algorithm on neighbours’ graph The first clustering method implemented by the team was based on the distance between the histograms of the face images. This distance is calculated using the OpenCV API, obtaining the Bhattacharyya distance between images. In statistics, Bhattacharyya distance reflects the similarity between two discrete or continuous probability distributions, or, in other words, the amount of overlap between two samples. Once the distances are obtained, we build a weighted graph illustrating the similarity between the face images in the video. Edges below a certain threshold are then removed from the graph to form a disconnected graph. The Tarjan algorithm was used to find the strongly connected components of the graph, in this case representing different face clusters, using a depth-first approach.

This simple algorithm was found by us to work well on small datasets, both in terms of speed and accuracy. The execution time of Tarjan algorithm is linear in the number of edges, having $O(|V| + |E|)$ time complexity. However, we realised that for larger datasets the accuracy would have decreased dramatically as very different images can have a similar amount of overlap, hence they would belong to the same strongly connected component. In addition, the computational time would be proportional to the number of edges in the graph, the algorithm becoming significantly slower for large number of input images.

More research revealed two other algorithms that fit the problem and work well in relation to the two face recognition methods implemented, namely histograms comparison and PCA. These two algorithms, that will be described below, are similar in the sense that both extract information from data using mean vector operations.

2.4.3.2 K-means algorithm K-means clustering algorithm is an unsupervised learning procedure that aims to partition n observations into a number of clusters known in advance - k , chosen such that the points are mutually farthest apart and each observation belongs to the cluster with the nearest mean. The main idea gravitates around an iterative scheme for finding a locally minimal solution. At every iteration, each point belonging to a given data set is examined and associated to the nearest centroid. The position of the centroid is recalculated every time a new component is added to the cluster. When no more data points are pending, it means the first step is complete and an early grouping is done. A re-calculation of k new centroids, binding all the data points to the new nearest mean is then required. The centroids keep changing their position in this loop until no more moves are possible. As a final stage, the algorithm tries to minimise the objective function given by:

$$\arg \min_S \sum_{i=1}^k \sum_{X_j \in S_i} \|X_j - \mu_i\|^2$$

where (x_1, x_2, \dots, x_k) is the data set, and μ_i is the mean of points in the cluster S_i . The time complexity of the algorithm is $O(n * k * \text{no_iterations})$.

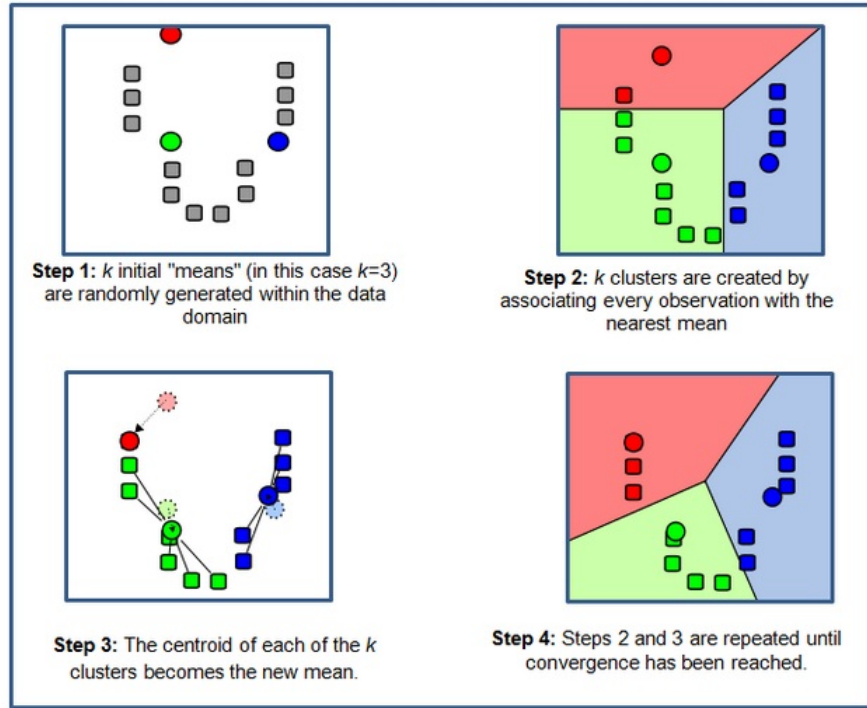


Figure 11: Illustration of K-means algorithm.

The main reason for the decision to implement this algorithm is the fact that it is proven to work well with PCA, which had already been used as a technique for face recognition. In [9] experiments show that the continuous solutions of the discrete K-means clustering membership indicators are in fact the principal components derived using PCA. In addition, K-means can be used on large datasets due to its general good average speed as every iteration is linear in the size of its input. It would, therefore, suit the scope and scale of our process.

However, it also has its limitations. In general, it is known to have problems when clusters are empty, non-globular or of different sizes and densities. Furthermore, it has a high sensitivity to initial conditions, which can influence dramatically the quality of the output. It was noticed that different initial partitions can result in totally different final clusters, with different levels of optimality. Also, predicting accurately the number of clusters in advance is impossible in the case of our project, which makes this algorithm on its own impossible to use. As a temporary solution for both issues, running the program with various initial configurations was considered and the results compared. Below is an illustration of these two major problems.

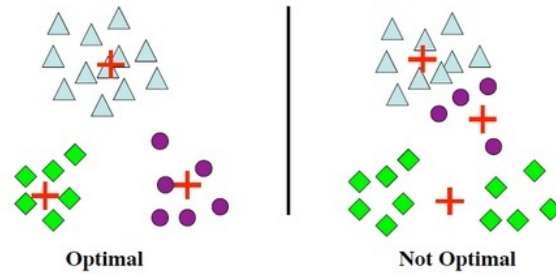


Figure 12: Initial partition - bad initialisation results in poor results.

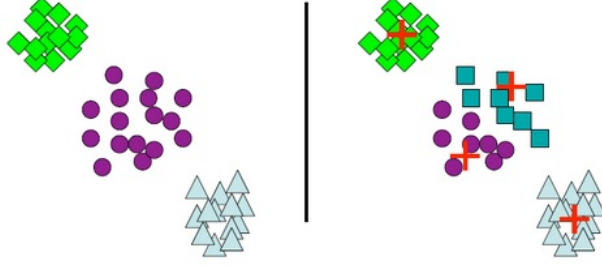


Figure 13: Wrong prediction of number of clusters.

2.4.3.3 Means-Shift algorithm Further research indicated a new algorithm to overcome the issue raised by being unable to predict the number of clusters in advance. This is the Means-shift algorithm, which is a non-parametric technique for feature space analysis, with applications in computing vision and image processing in particular. It is also an iterative procedure which works on the principle of shifting each data point towards the average of the data points in its neighbourhood. It can be considered as a generalisation of K-means clustering algorithm and is generally considered to work well with histograms.

Going into further detail, Means-shift algorithm aims to locate the maxima of the density function given discrete data sampled from this function. It starts with an estimate x and repeatedly calculates the value of the mean shift, given by the following formula in [10]:

$$m_h(X) = \frac{\sum_{i=1}^n X_i g\left(\left\|\frac{X-X_i}{h}\right\|^2\right)}{\sum_{i=1}^n g\left(\left\|\frac{X-X_i}{h}\right\|^2\right)} - X$$

where x_i are the data points and $g(x)$ is a kernel function. The procedure continues with successive computations of the means shift vector $m_h(x^t)$ and translations of the vector x^{t+1} to $x^t + m_h(x_t)$ and is guaranteed to converge to a global maxima.

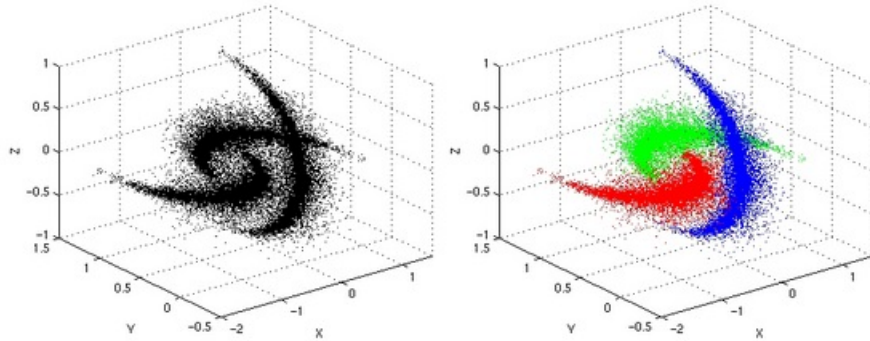


Figure 14: Real example of 14 826 points in the LUV color space

After further analysing this algorithm, the following observations were noted. Its time complexity is $O(k \times N^2)$, where N is the number of data points and k the average number of iteration steps required for each data point. We can conclude the algorithm is calculation intensive and expect it to run slower than K-means clustering. Another drawback the comparison revealed is

the fact that it sometimes fails to classify data points situated between natural clusters, so called data outliers, leading to creation of some small uncertain clusters in addition to the wanted major ones.

2.4.3.4 Hybrid algorithm After researching, implementing and analysing the two algorithms described above, we realised none of them on their own would fully fit the scope of our project and return an accurate result in a reasonable time complexity. Inspired by a blog entry on “Visualising High Dimensional Data” [12], we implemented a hybrid algorithm of K-means and Means-shift, trying to eliminate the drawbacks of the two techniques and maximise their advantages.

In the first stage, we apply K-means algorithm on the given data set, without requiring a precise number k of clusters, but generate instead a number that is significantly larger than the total of natural clusters. On the centroids found by this first procedure, we apply the Means-shift algorithm which will merge appropriate cluster centers to match natural clusters. This is being done quickly because the input set at this stage is much smaller than the original data set. Moreover, because generally the K-means clusters are located in high density sites, Means-shift will not produce the small uncertain clusters that we noticed when implementing the technique on its own.

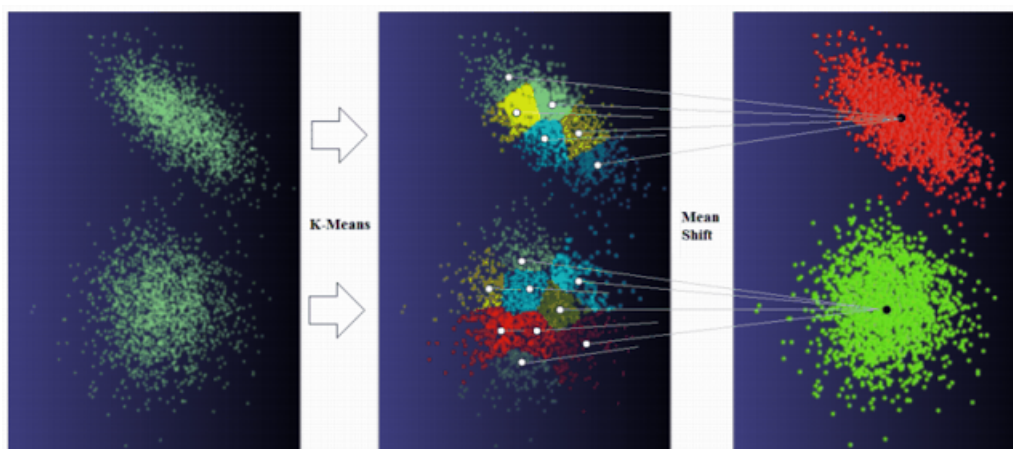


Figure 15: Hybrid algorithm

However, testing revealed that the predicted number of clusters, k , must not only be significantly larger than the number of actual number of clusters, but also smaller than the total number of faces in the video. This value is not being known in advance in our case, which means that the performance of the hybrid algorithm will not exceed the one of K-Means algorithm.

2.4.4 Segmentation

Video segmentation refers to “grouping of pixels into spatio-temporal regions that exhibit coherence in both appearance and motion” [13]. It is considered to be a difficult and complex problem. Under special circumstances, such as black frames or computer generated videos, the segmentation could be achieved easily, using frame histograms comparison for example. However, what we aimed to achieve was a segmentation based on faces appearing in the video, which brings semantic in the picture and can present major challenges in terms of temporal coherence, automatic processing and scalability.

2.4.4.1 Homemade algorithm The algorithm for segmenting the video based on the faces identified in the previous steps was designed from scratch. Based on a given threshold k , the procedure iterates through k consecutive frames at a time and keeps track of the faces encountered. For each face, it checks whether that particular person is already in the current segment. If it is, the face is added to the list of the current segment, otherwise to a waiting list. In the case when no face is found in the current segment for k frames, the waiting list is made into a new segment. The algorithm ends when all the faces are allocated to a segment.

Testing revealed that this algorithm runs fairly quickly, with a linear time complexity of $O(k \times N)$, where k is the maximum number of faces appearing in a single frame and N the total number of frames in the video. One of the identified main drawbacks is related to the length of the segments. For example, if a new face appears for a number of frames larger than the threshold, the algorithm will create a new segment.

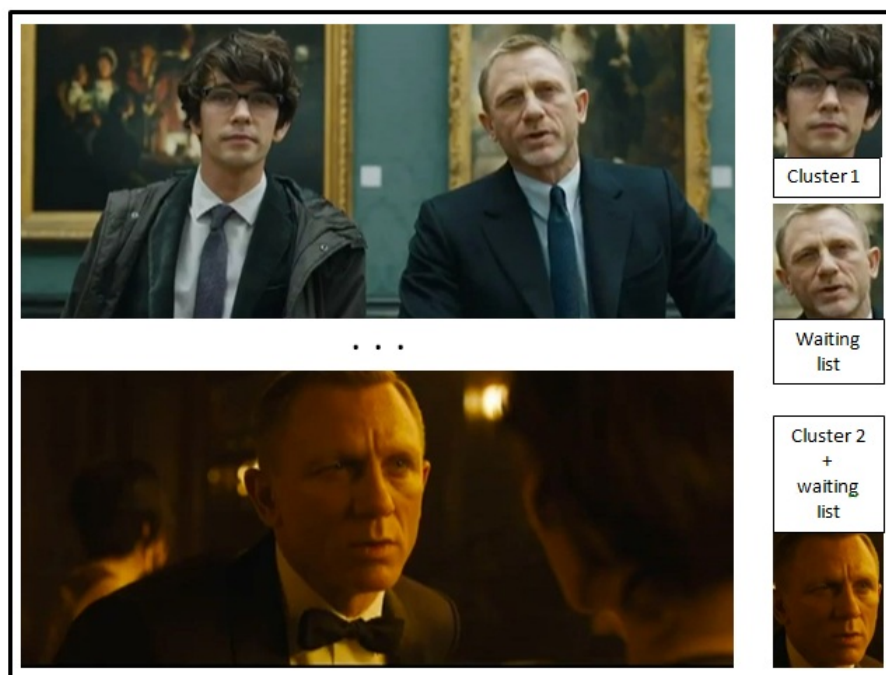


Figure 16: Illustration of the segmentation algorithm

2.4.4.2 Other algorithms Although the team decided to implement its own segmentation algorithm in order to experiment with ideas, research investigating the existing work in the field was also conducted.

One of the existing approaches to video segmentation disregards temporal coherence, being inherently scalable, but “may cause jitter across frames” [14]. One example of such an algorithm was proposed by Freedman and Kisilev [15], who used a Means-shift approach on a cluster of 10 frames in order to generate results without accounting for temporal information.

As far as the spatio-temporal video segmentation procedures are concerned, one can distinguish between those relying only on past frames and those making use of information from future frames as well. The approach used for the project only accounts for past frames and still provides satisfactory results. Similarly, an algorithm based on Means-shift proposed by Paris et al. [15] achieved real-time performance without examining future frames in the video.

Taking advantage of both past and future frames in segmenting a video introduced a new class of techniques, known to be efficient, which makes them suitable for extensions to spatio-temporal

segmentation. Researchers from Georgia Institute of Technology in collaboration with Google Research Laboratories in Mountain View [16] implemented a hierarchical framework based on this principle, using a graph algorithm. Repeating an iterative process over multiple levels, it creates a tree of spatio-temporal segmentations. The algorithm is proved to be efficient in terms of execution time and memory and scale well for long videos. However, it is sensitive to smooth illumination changes, making it unsuitable to use for the scope of our project. In the videos worked with in this project, it happens quite often for the same person to appear in several scenes with different lighting conditions even in consecutive frames, which would lead to unnecessary fragmentation.

A great amount of work related to video segmentation is based on scene recognition, with very little material being available on how to divide a video into chapters based on object classes, which posed a challenge.

2.4.5 Indexing

Video indexing can either be done at low-level or semantically. The low-level features can be extracted from so called “key-frames” such as color, text, shape, being usually performed frame-wise. In terms of semantic index, the purpose is to tag meaningful objects, like faces. This is exactly what we are aiming to achieve in our project. It highly depends on the appropriate segmentation done beforehand and is considered in general an open problem, mainly due to the multiple issues still arising at the segmentation level.

In this sense, the user may navigate through all the scenes corresponding to a particular person identified in the video.

2.4.6 Technical challenges

2.4.6.1 Understanding the mathematical concepts The project required a strong mathematical background and undertaking extensive research in Computing Vision in general and face detection and recognition in particular. None of the team members had previous experience in this field, hence initially a considerable amount of time was spent learning about different existing algorithms. Resources used were research papers available online as well as material for the course in “Intelligent Data and Probabilistic Inference” taught by Prof. Duncan Gillies to fourth year students. Particularly useful lectures were numbers 15 and 16, on “Feature Reduction, Principal Component Analysis” and “Linear Discriminant Analysis”, respectively.

However, the main challenge faced here was truly understanding how they work from a mathematical point of view and become familiar with all the new concepts involved in order to be able to implement them ourselves. For example, in the case of PCA, the OpenCV API would have provided the required methods to extract the principal components from an image face and project them in the subspace. However, the lack of documentation in OpenCV 2 made this a difficult task as the output of the projection didn’t resemble a face and it was impossible to determine whether the input data was configured correctly. When calling the same API in C++, this being the main language the library has support for, the intended results were obtained. It was therefore decided to implement the entire algorithm manually. This decision served two purposes: Firstly, it solved the problem (trying to debug a call to an external library method poorly documented would have been a waste of time) and secondly, it gave a better understanding of the algorithm and forced the team to read more on its mathematical side.

All further algorithms required for completing the project were implemented by the team. This meant that unnecessary bugs were avoided. Moreover it provided everyone with more experience on the technical and mathematical sides, laying the basis of a strong background in video processing, face detection, recognition and classifications.

2.4.6.2 Finding a good clustering algorithm “Good” is indeed a broad term, but in this case we mean for it to work well with our face recognition algorithms, be fast, accurate and not over complicated such that we manage to understand and implement it in the given timeframe.

Research showed both K-Means and Means-Shift algorithms are solutions to the clustering problem. Although both had good performance and were known to work well with face detection procedures, none of them seemed to provide satisfactory test results in the specific case of this project. While the former one required an accurate prediction of the number of clusters in advance, the second one was computationally intensive, failing to classify faces falling between natural clusters. After several attempts of finding another algorithm that would be able to solve this problem, a hybrid between the two, which maximises their benefits while diminishing their limitations, seemed to be perfectly suited as a solution. As previously explained, in our particular case, generating a prediction for the number of cluster bigger than the actual number of faces in the video, didn’t add any improvement compared to K-means algorithm, so we decided to use Means-shift clustering.

2.4.6.3 Accuracy versus efficiency The product is intended to be able to handle video files of various sizes. When choosing what algorithm to use for solving problems such as face detection, recognition or clustering there is always a trade off between accuracy and efficiency. As this was realised in the early stages of the project, the decision to deliver a desktop application made it clear that accuracy was a priority. Even under these circumstances, some operations might take an unacceptably long amount of time to complete even for a desktop application. This is why we had to constantly focus on improving our algorithms in order to maintain a fair balance between accuracy and efficiency. Examples of these improvements can be found at every stage as we implemented alternative solutions for the same problem when possible and compared their performance.

3 Evaluation

3.1 Testing

Testing the software played a significant yet challenging part in our project. The team aimed to develop a reliable product of high quality and testing was essential to achieve this goal.

One of the challenging aspects of testing results from the fact that the input to the application are movie files of any kind. In the beginning, the team focused on testing using only several movie trailers. This seemed to give accurate results, however when a bigger variety of movies was introduced the results were not as accurate as hoped, for example due to thresholds in functions adjusted only for specific test cases. Fortunately, this problem became clear in the early phase of the project and later on the team ensured that the test cases cover a wide range of examples making the product more reliable.

3.1.1 Test-driven-development

In the first phases of the project the team experimented with test-driven-development method as part of following the agile development techniques. The method is based on continuous repeating of three steps: first write a test to check functionality to be added (this test should fail in the beginning), then implement this particular functionality (the test checking it should pass) and finally refactor the code.

Following this method proved to be beneficial in the beginning of the project when simpler algorithms for face detection or recognition were used. It helped in discovering bugs early in the

development cycle (e.g. detecting faces of people taking into account various skin complexions) as well as in understanding how the algorithms we use work. However, later on, when more challenging and accurate algorithms were introduced, that required more advanced mathematical background and profound understanding in order to write any code, test-driven-development approach became more difficult to apply. This is why the team decided to follow a more intuitive approach, mainly introducing tests after writing the code.

3.1.2 Unit testing

The main testing technique applied in the project was unit testing. The team relied mostly upon *unittest* framework for Python that provides a wide range of tools for constructing, running and automating the tests. Unit testing of functions from the very beginning of the project ensured high quality of the code and helped to determine the accuracy of algorithms, what worked well and what had to be improved. As mentioned, a significant effort was put into writing unit tests in order to check the reliability and correctness of the most difficult algorithms, such as PCA or mean shift. Writing unit tests while attempting to use multiple threads to handle the progress bar was also an interesting and challenging problem.

While developing unit tests the focus was put on making them very readable. As the code tested was often difficult to understand at first sight the purpose of the tests was not only to check the correctness of the code, but also to make it easier to understand for the code reader. Following several suggestions from *Growing Object-Oriented Software, Guided by Tests* [17], tests were made more readable and consequently improved the productivity. Firstly, features rather than methods were tested. For instance, given a test function like *distinguishDifferentPeopleInTheSameScene()*, we can't tell which methods it checks from the name, but it gives an overview of the features implemented in a class. This approach also has the advantage of providing free documentation.

Another approach suggested in the book was to write the tests in a standard form. This means first *setup* the context of the test, then *execute* the target code, triggering the tested behaviour, *verify* if the expected behaviour occurs and then *teardown* in order to clean up any leftover state that might corrupt other tests.

3.1.3 Usability testing

“Usability means that the people who use the product can do so quickly and easily to accomplish their own tasks.” [1]

As the final product is a desktop application, it was agreed that it would be critical to test the usability of the user interface.

The usability testing technique that fitted the project best was the “what and why” brief, suggested by Hannah Donovan, one of the guest speakers in the Software Engineering Practice course. Instead of giving a demo to clients, they are given the opportunity to play with the product themselves and all suggestions and comments they make are noted on the way, carefully observing their actions and any moments of hesitation.

Throughout the project several usability testing sessions were conducted. The team focused on testing with the main client (the supervisor of the project). His feedback was helpful in improving the user interface of the product, especially the way of navigating through the program. For instance, during a test the client attempted to quickly go through video segments, one after another. To do so, he had to click *Next* button located in the right edge of the window and then click *Play* button positioned in the middle of the window. This way of navigating through segments seemed to be impractical and hence a simpler solution was introduced: a *Play Next* button.

Following suggestions of the correct usability testing described in *A practical guide to usability testing* [1], more tests were conducted on students from the department. These are people who could potentially be future clients, as they often use videos. Furthermore, it was suggested that real data should be used during tests. In the first usability tests example videos to process were provided to test participants. In the later iterations the team realised that this approach is not useful, as the video samples were already unit tested and hence using them would not help in detecting possible problems that had not been thought about before. Thanks to the student's feedback the interface was gradually improved and made more intuitive for the average user.

3.2 Validation of the product

At the time of writing, an unforeseen bug causes the program to use an extremely large amount of memory. As an example, running the PCA and means shift algorithms on 1500 frames of a 720p video (1280 x 720) uses approximately 12 GB of memory.

This is of course excessive, and results in an unusable program when trying to process more than a minute of video. While this is in the process of being fixed, the statistics and graphs given below are based on the faulty program. This accounts for some of the growth in the “time efficiency” curves below - because of memory swapping to the disk.

3.2.1 Efficiency

The numbers required to create the following graphs have been obtained on a Linux machine with:

- Intel Core i7 CPU clocked at 3.40GHz (four physical cores)
- 8 GB of RAM and 4 GB of SWAP space.

Each video has been run through the program with the PCA comparator and the Means-Shift clustering algorithm. A measure of time has been taken every 50 frames.

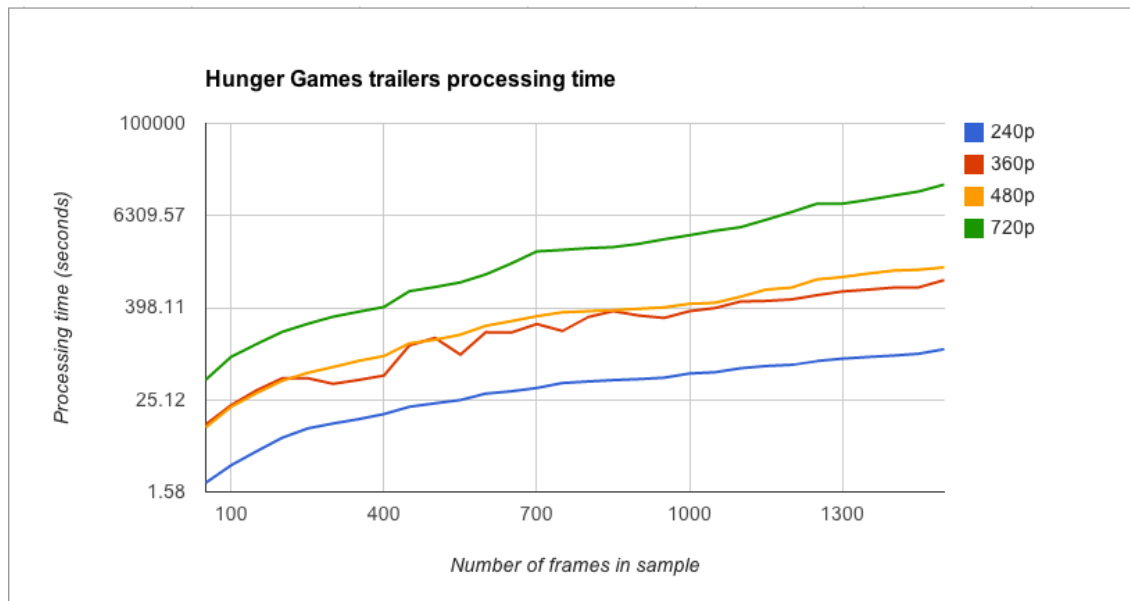


Figure 17: Hunger Games trailers processing time.

Run over a one minute long sample, our product takes roughly 20 minutes to detect, recognise and cluster the faces, segment the video and write it all to disc. To better show the effect of

video size on the time taken to complete a run of the program, four identical videos (the “Hunger Games” trailers) in different resolutions have been chosen for the statistics.

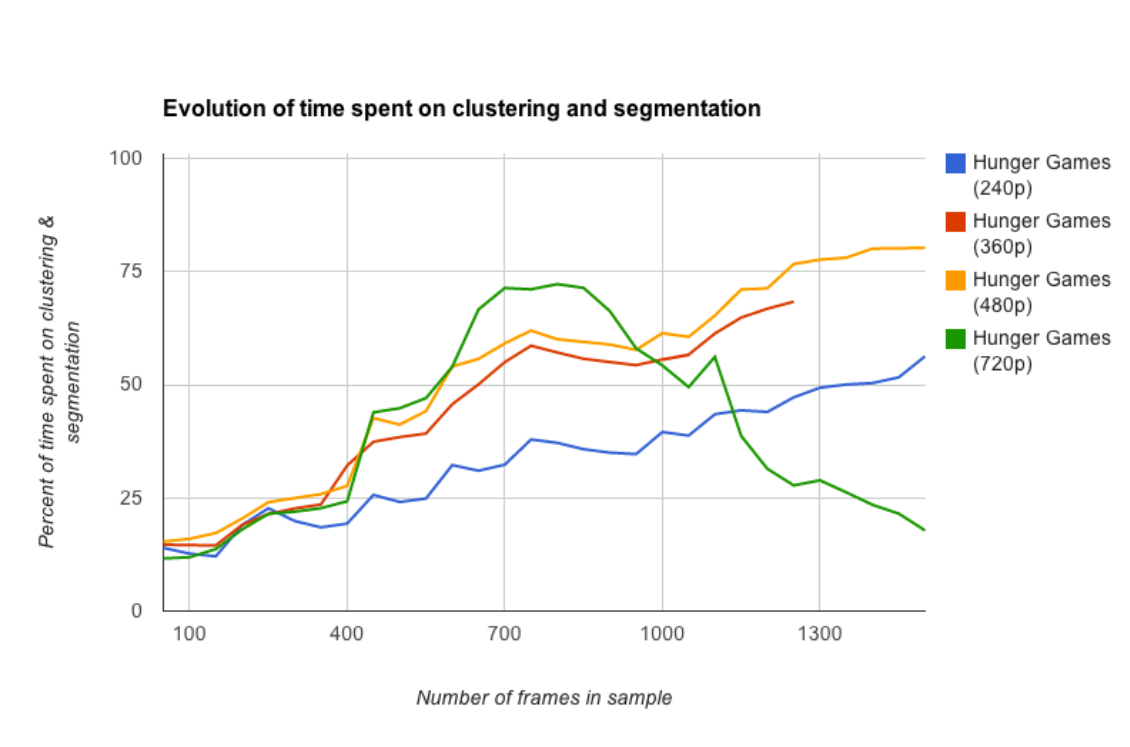


Figure 18: Evolution of time spent on clustering and segmentation.

Experimental results show that the time spent on the clustering and segmentation algorithms grows faster than the time needed to recognise faces. This accounts (with swapping to disc) for the above linear growth of the curves in Figure 14. Note that the 720p video curve drop is explained by disc swapping occurring during the face recognition phase, thereby shifting the time spent to the face recognition algorithm.

3.2.2 Accuracy

Because of the small size of the sampled video (limitation imposed by the bug described above), measures of accuracy are not representative of longer video sizes.

Accuracy has been deemed correct and reliable given careful choice of parameters. It is difficult to adequately represent the accuracy of an intuitive notion such as “video scenes” in writing. It therefore suffices to say that the splits returned by the program were as expected given the details of segmentation algorithm used in this product.

3.2.3 Tradeoffs

Once more, correction of the bug described in the introduction is necessary before a discussion of the tradeoffs is relevant. For the sake of argument, it is assumed that the program does not use an abnormal amount of memory.

The main tradeoff involved was speed of development versus efficiency. The choice of Python was a great help for the development phase: automatic handling of memory, quality bindings for OpenCV and the ease of use of a scripting language all contributed to fast development. It is our

opinion, however, that a product intended to be more than a proof of concept should be written in a faster language.

3.3 Main achievements

3.3.1 Effort

The following table and charts below illustrate the contribution of each member of the team to the project taking into account number of lines of code and time spent on each of the components, computed from the logs that have been updated at the beginning of each iteration.

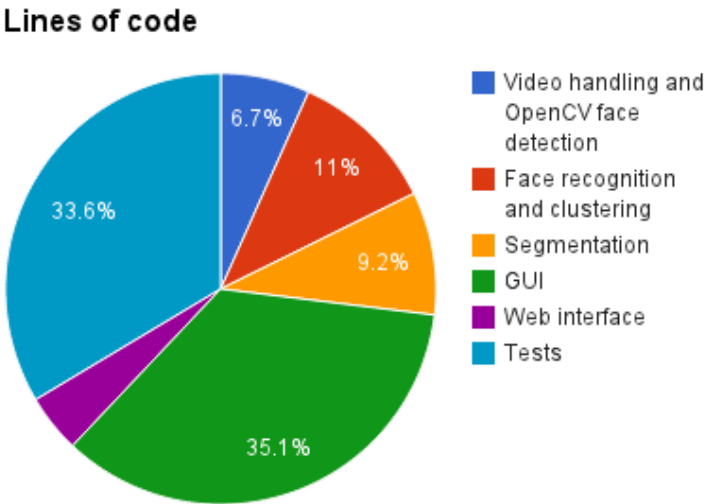


Figure 19: Lines of code.

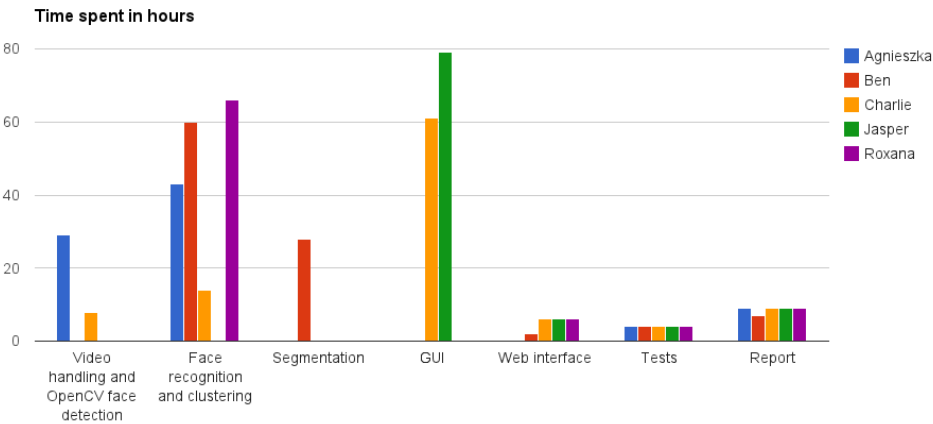


Figure 20: Time spent in hours.

As recommended in the “Group Project Overview”, the aim was to spend around 10 hours a week on the group project, including development and diaries. However, due to coursework deadlines, MEng tests and Christmas break, the time spent weekly varied greatly. For example, no work was expected to be done during the exams week or the holiday, so a balance had to be maintained to make sure the deadlines were met. At the beginning of each iteration, a detailed analysis of time spent was undertaken as a retrospective, highlighting which elements took more time to implement and why. This served in finding new strategies to increase productivity. The biggest amount of effort was put into implementing the algorithms for segmentation, face recognition and clustering, as can be seen in the diagram above. The component with the largest number of lines of code is the graphical user interface. The team put much effort into ensuring that the interface is intuitive and appealing, provides all the fundamental features, as well as various additional options for the user suggested during usability testing.

3.4 Reflections

3.4.1 Plans vs. Achievements

In the first iteration of the project the team decided what features the final product would need to have and what extra features could be added with enough time left. The focus was put on implementing these main goals first. After becoming more familiar with the libraries and algorithms used additional ideas for future extensions emerged.

The following table compares the planned features with the ones that were actually implemented.

Features	Type	Implemented	Observations
Back end			
Face detection using OpenCV	key feature	Yes	
Histograms comparison for face detection	key feature	Yes	
Tarjan algorithm for finding strongly	key feature	Yes	
Segmentation based on black frames	key feature	Yes	
Segmentation based on faces	key feature	Yes	
Segmentation based on time	key feature	Yes	
Indexing of scenes based on faces	key feature	Yes	
Means-shift algorithm for clustering	extension	Yes	Added later to
Hybrid algorithm for clustering	extension	No	Feature added later
K-means algorithm for clustering	extension	Yes	
PCA	extension	Yes	
Hidden-Markov Model algorithm for face	future extension	No	Would have required
Combination of Eigenfaces and Fisherfaces	future extension	No	Would be an
Trained database for face recognition	future extension	No	This has been
Connection to IMDB database	future extension	No	Would have required
Front end			
Play button	key feature	Yes	
Desired segmentation method selection	key feature	Yes	
Specifying start/end frame	key feature	Yes	
Web interface	key feature	Yes	Implemented in the
Segments view: list view	key feature	Yes	
Progress Bar	extension	Yes	
'Play next' button	extension	Yes	
Save and Load buttons	extension	Yes	It is possible to save
Cancel button	extension	Yes	
Separating loading and segmenting a video	extension	Yes	
Displaying info about video	extension	Yes	
More types of segments views.	future extension	Partially	E.g. a "coverflow" of
Combining segmentation options	future extension	No	It is possible to
Estimated time remaining	future extension	No	Only progress bar

Figure 21: Expectations versus achievements.

The key features needed for the product to work correctly were implemented, as well as some of the possible extensions. The team plans to continue working on the project to implement extra features that required a greater amount of time (e.g. having a trained data set). One aim is to improve the running time of the segmentation based on faces and to implement an indexing strategy that would add more meaning to the tags for the user. Future extensions will be facilitated by the structure of our current code, which was designed to be easily extensible and maintainable.

4 Conclusions and future extensions

4.1 Things learned

The team took away many technical things from this project, the first being experience in programming in Python. Particularly the effects of code only being interpreted at runtime and the duck typing on one hand, but the simple script syntax and resulting ease of coding on the other hand are aspects worth considering when looking for an implementation language.

Secondly knowledge in face recognition, video handling and front end design was gained by team members. The team has become familiar with several useful libraries and algorithms to deal with things from handling video files frame by frame to detecting, comparing and clustering faces and finally designing a graphical user interface for a desktop application.

4.2 Future extensions

Extensions to the product are possible both in the front and the back end.

4.2.1 Segment browser

To make the front end more attractive, different views of the segments could be offered apart from a simple list. One option could be a tiles view with either screenshots or moving previews of each segment in miniature form. Another idea is to have a “coverflow” of previews or screenshots.

4.2.2 Web interface

As mentioned above, during the course of the project the initial web interface was changed to a desktop application, since it was found that this would be a more suitable interface for the type of service being offered. However to reach a broader audience a web extension would still be possible. To make the segmentation process in the background quicker, MapReduce could be used, and already segmented files could be cached on the server. User accounts could also be introduced, to let users save their segmented videos on the server and access them again later.

4.2.3 Indicator of remaining time

An indicator of the remaining time needed for segmentation would be useful next to the progress bar, to give the user an even better feeling of the time needed to segment the video. This became apparent in one of the conducted usability tests.

4.2.4 Segmentation options

An interesting extension to the segmentation options would be to combine them, so a video could for example be segmented on black frames as well as every 3 minutes, the next cut being made according to whichever segmentation rule is applicable next.

In addition, more careful thought should be given to the current segmentation algorithm. Although its time complexity is good, we would like to improve on the accuracy and overcome its limitations by adopting some advanced mathematical approaches in tackling this matter.

4.2.5 Face recognition algorithm

To improve the face recognition algorithm, *Hidden Markov Models* (HMM) could be implemented, as described in 2.4.2.3.

4.2.6 Clustering algorithm

The clustering algorithm could be improved by using a trained data set of faces and learning algorithms based on neural networks. Clusters could also be connected to photos of actors, to be able to assign the actor’s names to them. Once the name is linked to a cluster it can even be used further to accompany indexed segments with links to information on the actors and characters appearing in the scene.

Furthermore, a combination of eigenfaces and fisherfaces could make the face detection prior to clustering more robust while retaining the simplicity of eigenfaces.

4.2.7 Indexing

Having a trained database would allow for dramatic improvements in terms of automatic indexing of the people appearing in videos, attaching more meaning to the indexes.

5 Project management

5.1 Collaboration

In order to describe and evaluate the interactions between team members, every team member was categorised using the Belbin Team roles (c.f. image below). Note that the characteristics assigned to each member are not absolute. Because of the relatively small scale of the project, each team member was at one point or another implementing, solving difficult problems or leading a development effort.










	Contribution	Allowable weaknesses
 Plant	Creative, imaginative, unorthodox. Solves difficult problems	Ignores incidentals. Too preoccupied to communicate effectively
 Resource investigator	Extrovert, enthusiastic, communicative. Explores opportunities. Develops contacts	Over-optimistic. Loses interest once initial enthusiasm has passed
 Co-ordinator	Mature, confident, a good chairperson. Clarifies goals, promotes decision-making, delegates well	Can be seen as manipulative. Offloads personal work.
 Shaper	Challenging, dynamic, thrives on pressure. The drive and courage to overcome obstacles.	Prone to provocation. Offends people's feelings.
 Monitor Evaluator	Sober, strategic and discerning. Sees all options. Judges accurately.	Lacks drive and the ability to inspire others.
 Team worker	Co-operative, mild, perceptive and diplomatic. Listens, builds, averts friction.	Indecisive in crunch situations.
 Implementer	Disciplined, reliable, conservative and efficient. Turns ideas into practical actions.	Somewhat inflexible. Slow to respond to new possibilities.
 Completer Finisher	Painstaking, conscientious, anxious. Searches out errors and omissions. Delivers on time.	Inclined to worry unduly. Reluctant to delegate.
 Specialist	Single-minded, self-starting, dedicated. Provides knowledge and skills in rare supply.	Contributes on only a narrow front. Dwells on technicalities.

Figure 22: Belbin Team Roles.

- Roxana Danila: resource investigator.
- Ben Grabham: specialist.
- Charlie Paucard: implementer.
- Jasper Pult: coordinator.
- Agnieszka Szefer: team worker.

A major aspect of team collaboration were the group meetings. They were used to reflect on the previous iteration and plan the week's work. Team organisation, further development of the project and issues related to the codebase were common topics discussed to ensure good team cohesion and a solid architecture for the project. Once the work was planned and each team member was assigned a task and a partner, the week's features were implemented. Any further precision or issues arising during the week were dealt with in quick stand up meetings. Pair programming was where the bulk of the implementation took place, as detailed in the next section.

Because of the importance and length of the weekly meetings, it was useful to have someone keep the team on track and avoid getting caught up in specifics. Jasper proved to be an effective coordinator, and so led the discussions at the weekly meetings. Meetings are also a source of conflict, and need to remain polite and argued to be productive. Agnieszka was a good team catalyst and helped the discussion move forward smoothly. By framing the discussions and encouraging team work, pleasant and productive group meetings were achieved.

The technical nature of the project required some amount of research in the area before starting the implementation. In particular, the clustering process proved difficult to grasp. It is used to group faces together, which then is used for the segmentation and indexing jobs. However, with no database of faces, variable video quality and differences in lighting and background, it is a complicated process. Roxana was charged with comparing all the suitable algorithms, resulting in our choice of PCA. Having a single person focus on the research of algorithms was an effective way to quickly come to a decision. Roxana would inform the rest of the group of her findings and the reasons for her choice at the next group meeting to make the final choice.

Implementation of the chosen algorithms was a group wide task, but some specific issues were tackled in their majority by single team members. For example, when it came to implementing the PCA algorithm, we were unable to properly produce the mean face. Ben's knowledge and coding skill were needed to fix the code and resolve the problem. It was also necessary to have team members focus in greater amounts on the implementation than on the research to meet weekly goals. Charlie therefore spent a majority of the time on the code base.

As mentioned above, in a project such as this, assigning specific labels to the team members does not cover all the responsibilities. Most of the time was in fact spent pair programming.

5.1.1 Pair programming

“Research strongly suggests that people are more creative when they enjoy privacy and freedom from interruption. [...] What distinguished programmers at the top-performing companies wasn't greater experience or better pay. It was how much privacy, personal workspace and freedom from interruption they enjoyed.” [2]

Before trying pair programming, most of the team members had a similar attitude towards the idea to the one quoted above. However, encouraged by the advantages of the method presented during Software Engineering Practices course and additional materials (e.g. *Extreme Programming book* [3] and articles on successful use of the technique [4]), we decided to adopt pair programming to our project.

This method requires one team member (the driver) to write the code and the other (the observer) to review the written code and to find test cases. This technique proved to be highly beneficial. Firstly, it helped everyone to learn faster and understand things better. Secondly, it greatly improved the intensity of work. It is far easier to focus when sitting next to a person that works on the same task and requires one's concentration to finish the task. Furthermore, it enforced more communication between team members, which is the first value of Extreme Programming [3]. Finally the code had less bugs and problems were detected much earlier than they might have been in the case of individual development.

For the first few weeks, the pairs completely changed after each group meeting. This way, everyone was alternatively a driver and an observer and got familiar with the entire code base. As the pairs weren't fixed, everyone learned to work together, thereby improving group cohesion. As the weeks went by, the team realised that it was more efficient to have a single person working on the weekly diaries, and the four others dealing with the code. This division of work lead to a maximisation of time spent on code.

The pairs also began to specialise in different sections of the code. Agnieszka, Roxana and Ben dealt with the majority of the clustering and segmentation while Jasper and Charlie focused on the front end. Thanks to frequent conversations, stand up meetings and weekly group meetings, everyone was able to stay up to date on the evolution of the code base while quickly producing code in their "area of expertise".

5.1.2 Workflow

It was quickly noticed that in order to achieve an effective workflow, it was necessary to keep all other team members up to date. Communication and quick integration in the main repository were fundamental, as was proximity. Working physically next to each other is an easy and natural way to share ideas, and stand up meetings are instantaneous and frequent. Sharing code is also easier: all it requires is a tap on the back of a team member to "git pull" and the habit of continuously integrating changes. This technique of continuous integration is focused on small commits and quick exchange of code. This reduces the workload on each pair when merging conflicts, and helps keep everyone updated on the code's current development.

5.1.3 Group organisation

Having learned about the iterative approach of the agile development, the team agreed that applying it to the project would enhance time management as a team, allow for gradual improvement of the product and provide regular client's feedback.

It was decided to work in weekly iterations. Each iteration would include a planning meeting for the current iteration. During the meeting a retrospective was performed with an analysis of what had worked well in the previous iteration and what could have been done better. Following the retrospective the current iteration was planned, including the decision which tasks needed to be completed, what was their complexity and estimated time of completion and who was going to work on which task. Each task with an assigned team member would then be put on the Trello board (a collaboration tool for task management - c.f. picture below). Until the end of the iteration the team would implement the tasks and test the code. In the meantime stand up meetings took place to discuss possible problems and challenges, and report the progress so far.

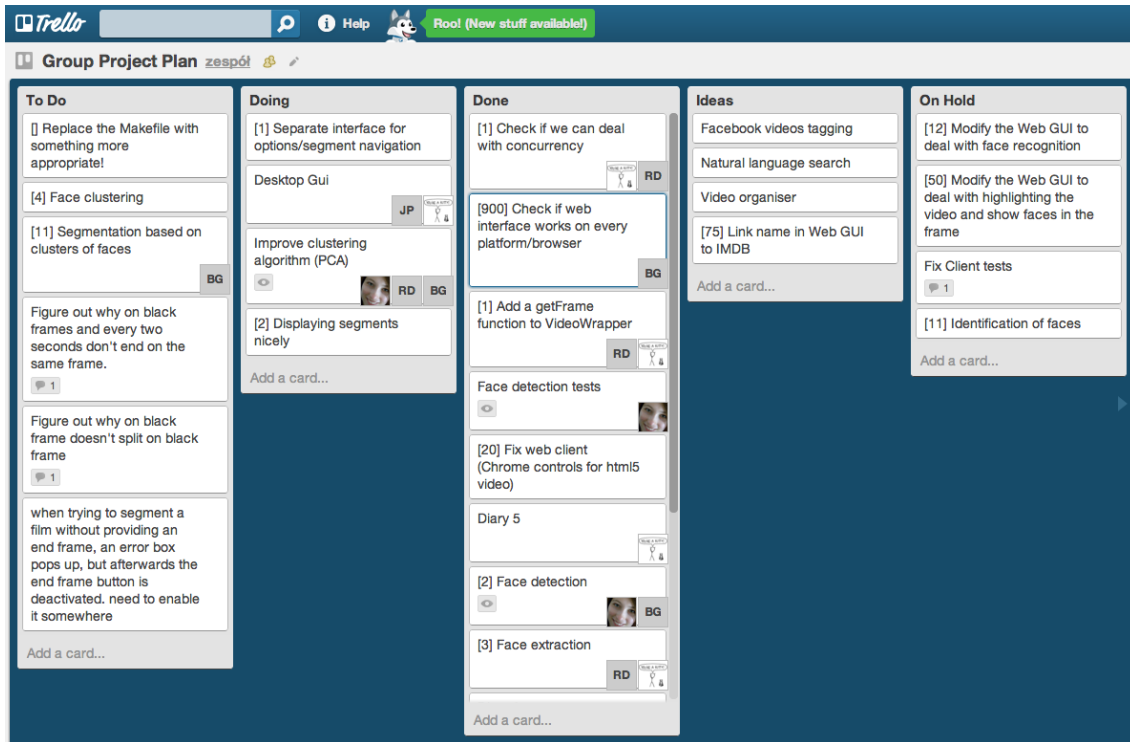


Figure 23: Trello board.

Thanks to applying these methods, a new version of our product was released at the end of each week. At the completion of an iteration the product was presented to the client in the form of a demo or a usability test. Client's feedback would be then taken into account in the next iteration. Frequent releases were a useful way to improve the product already in its first stages. Thanks to weekly client feedback the requirements of the final product were regularly made clear. Furthermore the iterative approach lead to systematic work, which proved to be beneficial not only for the final product but also for the team itself, especially towards the end of the project, when the workload from other courses started to accumulate.

5.2 Reflections

Looking back at the project management there are several options for improvement. One of them could be following test driven development from the beginning until the end of the project. Although TDD techniques were applied in the beginning of the project, in later iterations code was only tested after being written. Although writing tests for functionality yet to be implemented for the more complicated algorithms would be challenging, TDD had a potential to help in better understanding of the underlying concepts.

Furthermore more focus could have been put on assuring that each team member has a broad idea about the entire codebase. Again, this was achieved in the first iterations, however later on, when team members started specialising in their areas of responsibility, keeping track of the other parts became less important.

Overall the iterative approach of Extreme Programming was quite successful, giving early results and improving the product incrementally with new features every week. The regular meetings and pair programming contributed to a high level of communication and lead to thorough discussion and evaluation of every aspect of the project.