Image-Based Video Search Engine

Keyframe Extraction

L. Zheng and R. Bos

EE3L11 - Bachelor Graduation Project
June 13, 2022

Group H1

Image-Based Video Search Engine

Keyframe Extraction

by

L. Zheng and R. Bos

to obtain the degree of Bachelor of Science
at the Delft University of Technology,
to be defended publicly on Friday June 20, 2022, at 9:00 AM.

Student number:	5120225, 4730674	
Project duration:	April 19, 2022 – June 24, 2022	
Thesis committee:	Prof. dr. N. Llombart,	TU Delft, jury chair
	Dr. ir. J. Dauwels,	TU Delft, supervisor/ jury member 1
	Dr. ir. E.W. Bol,	TU Delft, jury member 2

TUDelft

Abstract

In this report, the analysis and design of a system that extracts *keyframes* from videos is detailed. The need for such a sub-module stems from the similarity of frames in a video. To aid in reducing the computation time of the content based video search engine, the Keyframe Extraction Module reduces the amount of frames by discarding frames that are similar in information. Determining what frames can be considered similar is one of the main challenges, as there are many ways of assigning values to how much frames differ.

In the past decades, many research has been done on keyframe extraction and video summarization and many methods are proposed to form keyframe selections, varying in what is considered salient information and varying in computation time. The most challenging part of the design is that there is a time constraint present, which called for a proper analysis in what methods are suitable. After all, this limitation when creating video summaries is often not a large topic in research papers.

This report will cover Shot Detection techniques along with various Keyframe Extraction methods that can be categorized in clustering, visual content, fixed selection and uniform sampling. Furthermore, evaluation methods like the Fidelity measure for the performance of particular methods are also addressed, as determining how well a keyframe selection is is not trivial. It is concluded that out of the techniques analyzed, a combination of VSUMM clustering and histogram matching along with histogram-based shot based detection with a CFAR threshold and pre-sampling is most suitable for the general case under a time constraint. Future work could include looking at hierarchical clustering methods and optimizing the Shot Boundary Detection module following the most recent papers.

Preface

This thesis was written as part of the Bachelor Graduation Project [EE3L11] to conclude our Bachelor Electrical Engineering at the Delft University of Technology. This project was proposed by our supervisor, Justin Dauwels, as part of an ongoing research project for Engineering Historical Memory [26]. Although the original project proposal was named “Image Search Engine for Digital History” and was later changed to ”Image-Based Video Search Engine”, we were just as eager to develop a working prototype of a video search engine, especially since the project was now similar to the work of previous students yet posed an extra challenge. If any future students decide to iterate on this thesis’ topic of Keyframe Extraction, we would be happy to receive a copy of their results (should they want to share them).

Furthermore, we would like to thank Justin Dauwels, Ioan Lager and Yuanyuan Yao for their guidance during the project. Lastly, we want to thank the other subgroups [15] [36] for the collaboration in working towards a functional engine.

L. Zheng an R. Bos
Delft, June 2022

Contents

1 Introduction

Image-Based Video Search Engine

This thesis is part of an ongoing research project ”Engineering Historical Memory” [26]. This thesis builds upon the work of one of the Bachelor Graduation Project [20] groups of the TU Delft from academic year 2021-2022 on a Search Engine for Digital History [12] [35]. The goal of their research was to create a search engine that can detect whether an object appears in a database of images. Since then, the CAS group of the TU Delft has worked on improving the image search engine with a team of MSc students. The next step in this project is to create a similar search engine that can detect instances of a desired object in a query video. This Image-Based Video Search Engine is to be designed for a variety of use cases, transcending the historical use case.

State-of-the-art Analysis

Instance-level image retrieval (IIR) is the problem of detecting an instance of an object that appears in an image and then retrieving images from a database that contain the same instance of this object. IIR and its application to video footage are both active fields of research. As video data and surveillance coverage are becoming ever more prevalent, developing information systems that can process and query this data is becoming increasingly important [5], as it is unfeasible to comb through all this footage by hand. Thus, there are endless applications for video-based IIR: such as person/vehicle identification, assistance in copyright claims, querying historical footage, etc.

Several examples exist of using IIR systems for finding images in a database of images [5], and for finding videos in a database of videos [10]. However, relatively little was found in using IIR systems for detecting images in a video. This suggests potential for research into the field. One of the rare implementations of a video-based IIR system is the work of A. Araujo *et al.* [2]. Their research focuses on reducing storage requirements when using IIR systems for video databases when using local feature matching by determining optimal descriptors. A similar existing system is Video Google [30]. The approach used by Video Google is to perform the search similar to how Google does text document retrieval. Frames of a video are evaluated based on two types of regions. One based on gradients and a second one based on how stationary objects are. Vector descriptors can be made based on these regions and evaluated using text retrieval methods. In this case an inverted file structure is used that stores the descriptors as visual words. Similar to how commonly occurring words (such as ’the’) are excluded in a text based scenario, the commonly occurring descriptors are put in a ’stop list’ that suppresses these occurrences. However, the reported system takes frames from the video as inputs. Both of these methods make use of local features for comparing the query images to the video dataset. Using local features leads to accurate results but in both papers time considerations are ignored.

Looking into existing video-based IIR systems for historic systems, only one implementation was found in the work of Condorelli *et al.* [9]. Their work focuses on detecting segments of video which contain lost cultural heritage in historical video footage. This method focuses on the accuracy of the resulting video segments and the length of these segments as compared to the length of the original footage.

However, similar to the existing systems explained above, the research does not take computing time into consideration.

From this State-of-the-art Analysis it can be concluded that research into video-based IIR systems has been done, but most research focuses on optimising accuracy of the system and barely any research focuses on the duration of the system. Thus, there is a lot of potential for research into the field.

Problem scoping and bounding

Utilising IIR for finding images in a database of images is in itself already valuable. However, with the rapidly growing amount of visual content, not only the amount of images is increasing but the amount of video material as well. Extending the IIR to videos opens up new possibilities to explore video material. From easier access by searching for images in a library of cultural heritage videos [10] to finding appearances of a companies products in extremist videos.

The main objective is to develop a system that is capable of retrieving occurrences of one or more desired objects in a set of given videos, based on a set of given images. The second objective is to document the process of developing this system. The first objective will be completed if it complies with the requirements as specified in Chapter 2. The second objective will be completed if it complies with the requirements as described in the BAP manual [20].

The main limitation of the project is the time constraint of 10 weeks. This is the time allocated to build the entire system and document the process. The system itself is limited to use content based methods only. This ensures that the content of the image is taken into account and no interpretations are made of the image. Text-based methods have the limitation of the terms that describe the image [24], which also limits search across cultures when these text-terms are not supported. Further constraints are placed on which methods to be used. Existing methods should be used to develop the search engine, in order to ensure the allocated time is put to good use on developing the system, rather than on improving existing methods. Lastly, the development is restricted to using Python [37] and its associated libraries and tools.

To create alternative systems to the Image-Based Video Search Engine, one could look into implementing different methods for each of the modules, specifically regarding Feature Extraction. This module is the slowest of the system, followed by the Keyframe extraction module. By looking into different methods for these modules, the speed of the system could be improved. Furthermore, the Feature Extraction module could be trained on different data in order to improve the performance of the system even further. Finally, further tuning of the module-specific parameters could also be done to improve the performance. The system-wide performance can be gauged by the Key Performance Indicators: mean average precision (mAP), recall, and the amount of time saved.

Problem statement

Based on the analysis, the problem scoping and the problem bounding, the following problem statement was derived:

Develop a system that can detect whether an instance of a desired object appears in a given video, based on a given set of images containing the desired object.

System Overview

Subdivision of the System

According to the thesis guidelines [20], each BAP group has to be split into three subgroups of two people each. As such, this Image-Based Video Search Engine has to be split into three modules. These modules were selected to be:

1. Key-Frame Extraction (KFE), which cuts the query video down into frames and removes unnecessary frames.

2. Feature Extraction (FE) [36], which translates the keyframes and the query image into feature vectors.
3. Data Compression & Nearest Neighbour Search (DCNNS) [15], which compares the extracted features of the keyframes to those of the query image.

The three modules will tackle the following subproblems respectively:

1. *Develop an algorithm that finds keyframes to reduce the amount of video frames to be evaluated.*
2. *Develop an algorithm that can extract the features of the keyframes and of the query images.*
3. *Develop an algorithm that can compare the features of the query image(s) to the features of the keyframes.*

This thesis will describe the Keyframe Extraction submodule.

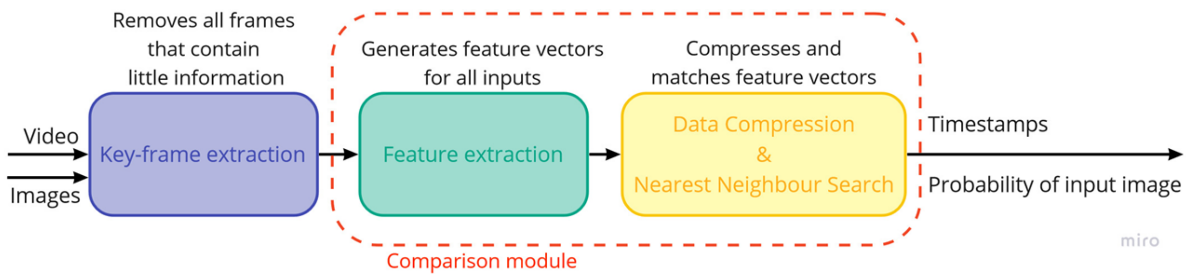

Removes all frames that contain little information
Generates feature vectors for all inputs
Compresses and matches feature vectors
Video
Images
Key-frame extraction
Feature extraction
Data Compression & Nearest Neighbour Search
Timestamps
Probability of input image
Comparison module
miro

Figure 1.1: Pipeline of the complete Image-Based Video Search Engine

Keyframe Extraction

The first module is the Keyframe Extraction submodule. It focuses on sub-problem 1 of the problem statement. In essence, the module consists of an algorithm that decodes the video into frames and sends a selection of these frames to the Feature Extraction module, along with the corresponding indices and the framerate of the video, to maintain information for determining the timestamps of where a certain frame appears in the video. The goal is to optimize this comparison by evaluating different methods and their strengths and weaknesses under varying conditions. These conditions include different types of video-content, playback time and resolution.

Document Structure

This thesis describes the Data Compression and Keyframe Extraction submodule. In Chapter 2 the programme of requirements is explained. In Chapter 3, existing methods are analysed. In Chapter 4 the design of the module is explained and some of the results are shown. In Chapter 5 the implementation and validation of the prototype are explained. In Chapter 6 the results are discussed and the conclusion is presented.

2 Requirements

Mandatory Requirements	**Trade-off Requirements**
Functional Requirements	*Functional Requirements*
1. The search engine must detect whether an instance of a desired object appears in a given video, based on a given set of images containing the desired object. 2. The search engine must return the timestamp(s) where the object appears in the video. 3. Matching of images to frames must be based on visual content.	1. The engine should be able to handle multiple input videos. 2. The engine should be able to handle multiple input images. 3. Mean Average Precision should be as high as possible. 4. The codebase should be structured clearly and properly documented. 5. The supported number of image formats should be as high as possible. 6. The supported number of video formats should be as high as possible. 7. The execution time should be as low as possible. 8. The system should be able to process videos with a large duration.
Non-Functional Requirements	*Non-Functional Requirements*
1. The system must support video files of the type mp4. 2. The full implementation must be completed within 10 weeks by a group of 6 students. 3. The system must be written in Python version 3.9 or higher. 4. *Conda version 4.10 or higher* must be used for synchronizing Python environments. 5. The engine must be able to be tested with hardware that is available to the group. 6. For a single query image, the engine should be able to process a video shot at 30 frames per second in half the duration of the video. 7. Mean Average Precision must be at least 65 %. 8. Mean Average Precision must be at least 65%	1. The extraction time per image should be as low as possible. 2. The resolution of the images should be as high as possible (to prevent information loss). 3. The MAP should be as high as possible. 4. The Recall should be as high as possible.

Table 2.1: Program of Requirements (PoR) of the entire engine

The Programme of Requirements lists the restrictions and functionality of the Image-Based Video

Search Engine. The requirements are divided in mandatory requirements and trade-off requirements. The mandatory requirements must be met and specify the core of the system. The trade-off requirements lists requirements that improve the system when they are met. Both sections are divided in functional requirements and non-functional requirements. The complete overview is shown above.

2.1. Program of Requirements for Keyframe Extraction submodule

Apart from the general Program of Requirements for the video search engine, an auxiliary Program of Requirements of the Keyframe Extraction module was constructed, which is depicted in Table 2.2. The metrics for the trade-off requirements will be elaborated in Chapter 3.

Mandatory Requirements	**Trade-off Requirements**
Functional Requirements	*Functional Requirements*
1. The submodule must reduce the amount frames of the video to be analyzed by subsequent modules. 2. The submodule must keep as many frames as possible where the instance of interest appears if it exists.	1. The Compression Ratio should be as high as possible. 2. The difference in content between frames that are selected should be as high as possible.
Non-Functional Requirements	*Non-Functional Requirements*
1. The submodule must support video files of the type *mp4* of variable duration and resolution. 2. The full implementation must be completed within 9 weeks by a group of 2 students. 3. The submodule must be able to function with up to 8GB RAM available. 4. The submodule should have as output (which form the input of the Feature Extraction submodule): RGB-data of the selection of frames, corresponding indices of the input video and the original framerate of the video. 5. The codebase should be structured clearly and properly documented	1. The Precision should be as high as possible. 2. The Recall should be as high as possible. 3. The F-measure should be as high as possible. 4. The Fidelity should be as high as possible. 5. The computation time should be as low as possible

Table 2.2: Keyframe Extraction Program of Requirements (PoR)

2.2. Computation time constraint

In addition to the requirements posed in the previous section, a constraint on the computation time of the method to be used comes to light when considering the performance of subsequent modules. One could define the primary function of the Keyframe Extraction module as reducing the overall search time of the video search engine by reducing the amount of frames for the Comparison Module to process. Consequently, if the computation time for discarding the redundant frames is comparable to that of the processing time of feature extraction and nearest neighbour search per frame, the effectiveness of the module comes into question. This leads to a constraint in computation time depedent on the performance of the other submodules in the system.

2.2.1. Assumption based time constraint

The main assumption is that if you were to uniformly sample (Section 3.3) a video at $5fps$ (which requires minimal computation time), the instance of interest always appears in the frames, if it exists.

From this perspective, the objective is to discard as much redundant frames from these 5 as possible. Equation 2.1 describes the time constraint that is posed on any chosen method in the design. Important to note is that $T_{computation\_keyframes}$ and $T_{uniformly\_sampling}$ are computation times per second of the input video. The constraint is based on the following assumptions.

- If one were to uniformly sample a video at 5 frames per second, the instance of interest is always in the selected frames if it exists.
- The chosen method has a recall of 100%.
- The computation time for the Comparison Module per frame does not depend on the amount of frames sent.

$$T_{computation\_keyframes} < (5 - N_{extracted}) * T_{comparison\_per\_frame} - T_{uniformly\_sampling} \qquad (2.1)$$

As an example, if uniformly sampling takes no time, and if the Comparison module uses 0.5 s per frame to process ($T_{computation\_keyframes} = 0.5s$), and the chosen method selects on average 3 frames per second of video, the computation time of the Keyframe Extraction submodule should be less than $(5 - 3) * 0.5 = 1s$ per second of video.

3 Analysis

There are several dependencies regarding the input and output that determine the scope of the design and the analysis of the keyframe extraction methods in this project. First is that there is no prior information known about the content, class or semantics of the input images. The video search engine is created for the general case; the instance appearing in the input image(s) is unknown. The system does not allow pre-training or method adjustment based on the type of instance. For this reason, along with other reasons that became clear from supervisor discussions, the Optimization module that was proposed in the early stages of the project (that would only extract frames containing the same class as the input images(s) and was pre-trained for thousands of object classes) has been omitted. Secondly, the main point of attention in the selection method such as movement of vehicles, color, logos, or faces cannot be determined beforehand, leading the design towards a method suitable for a set of keyframes that differ *in general* from one another, which will be discussed in this chapter.

3.1. Determination of keyframes

The extraction of *keyframes* is closely linked to video abstraction or video summarization, where the summary contains the most appropiate information, while preserving originality of the video[27]. This mechanism is often used to gain perspective of frames without watching the entire video. Video skimming is another form of video abstraction, where short clips of the video are merged with corresponding audio [29]. For the design described in this report however, this type of abstract is not applicable. Object-based video abstraction, which extracts objects for content-based analysis of videos [27], is also not applicable due to the lack of semantics involved in this design.

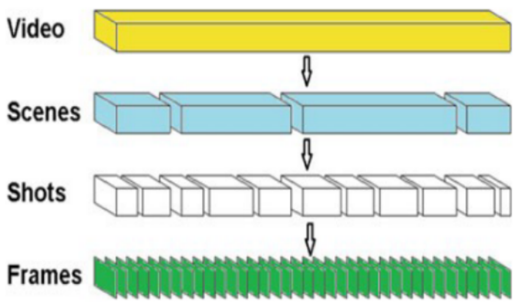


Figure 3.1: Video hierarchy [29]

When looking at the general hierarchy of a video in Figure 3.1 (which is undisputed by any paper adhering to this topic found), most important to note is that frames can be grouped into shots. Most research focuses determining the boundaries of these shots (also referred to as segments) before applying the selected method to form a video abstraction [32][21]. Shots are sequences of frames that are grouped by visual attributes, which are low-level features [27].

Although shots (or segments) often have abrupt transitions, determining the boundaries is not uncomplicated when there are transitions present that span over multiple frames. Part of the objective is to create a set of keyframes that are low in similarity, which for example means that a still image in a video would only result in one keyframe taken for the duration it is shown. Some of the approaches to compare frames are to use histograms, edge detection, (non-deep learning) feature extraction and clustering. For machine learning techniques, only unsupervised based methods can be made use of, as prior information about the dataset is unknown [29]. The categorization and analysis of different keyframe extraction methods are shown in Section 3.3.

In some works, a framework for video summarization is used that defines the properties continuity, priority and repetition [7]. Continuity means that the summary must be as uninterrupted as possible, priority means that some objects or events are more important than others and repetition means that is important that events may not be presented over and over again [7]. This shows that video summarization cannot be considered the same as Keyframe Extraction, especially for the design subject to this report, as creating a comprehensive summary of the video is not as important. Nevertheless, the approaches for both topics are very similar.

3.2. Evaluation metrics

The four general metrics that assess the performance of a generated set of keyframes are the Compression Ratio (CR), precision, recall and the F-measure, which are given in eq. 3.1 through 3.4. In these metrics, N_e is the number of keyframes extracted from the video, N_a is the number of accurately extracted keyframes, N_f is the total number of frames in the video and N_k is the number of ”true” keyframes in the video. Using the baseline fps of the time constraint assumption in Section 2.2, the CR should always be higher than $(1 - 5/30) * 100\% = 83.3\%$ for a video with a framerate of 30 fps. The main drawback of these metrics is that one cannot appoint fixed ”true” keyframes in video, which calls for more creativeness by for example setting ranges in where a keyframe is expected. The F-Measure is obtained by combining precision and recall using the harmonic mean [29]. A high recall would contribute to a higher accuracy in finding an instance appearing in a video, while a high precision would be beneficial time-wise, as less frames would need to be sent to the Comparison module.

$$CR = 1 - (N_e/N_f) \quad [*100\%] \tag{3.1}$$

$$Precision = N_a/N_e \tag{3.2}$$

$$Recall = N_a/N_k \tag{3.3}$$

$$F\_measure = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{3.4}$$

3.3. General techniques hierarchy

The range of complexity of methods to select frames from a shot is large. The most basic approaches include simply taking the first (and the last) frame in the shot, taking frames at set location in a shot, taking frames at intervals depending on the length of the shot, and uniformly sampling regardless of shot boundaries. In this section, the methods for determining the locations of the shots and the methods used to compute keyframes within shots are laid out. For further choices in design, the trade off between the performance and computation time is the main factor in determining the suitability of a given method, as also pointed out in [21]: *”The most simple techniques compromise extracted quality but the most sophisticated are computationally expensive”*. Although Keyframe Extraction methods can be categorized in different ways, in this analysis the hierarchy shown in Figure 3.2 is constructed.

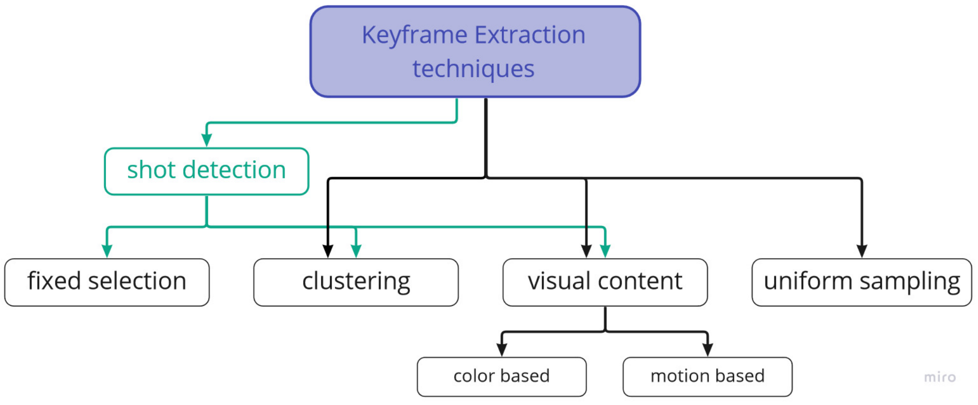


Figure 3.2: Hierarchy of methods discussed in this report

Fixed selection
A traditional approach is to select the first and the last frame (and perhaps middle frame) of every extracted shot [29][21]. This often results in a very condensed video summarization, yet the amount of keyframes is dependent on the length of shots. Although this technique is useful for generating an impression of the video content, it is highly likely that it would miss frames where the instance of interest occurs.

Uniform sampling
Uniform sampling is a simple technique that selects every kth frame of the original video as a keyframe, without laying importance on any visual content or dynamics, as with the fixed selection method [21]. This method is computationally cheap and the number of keyframes that is extracted can be fixated (and thus the Compression Ratio). The drawback of uniform sampling is that too few keyframes may be taken in a period where the information density is high, and vice versa. As mentioned in Chapter 2, the uniform sampling rate of $5fps$ (every 6th frame) is assumed to result in a recall of 100% for most videos (which gives a Compression Ratio of 83.3%). This is considered the baseline method, although for many videos it would result in a large amount of redundant frames, unnecessarily increasing the computation time of the entire system.

Visual content
Techniques based on visual information make use of consecutive frame differences in color histograms or histogram of orients (HOG). A simple consecutive frame difference method is to set a threshold for the histogram difference while reading through the frames of the video. The Compression Ratio would not be predetermined in this manner. Another method is the compute the histograms for all frames and select keyframes making use of the mean and standard deviation.

Clustering
Clustering techniques are based on fragmenting frames into clusters based on visual similarity and selecting the frames that are closest to the mean of the largest clusters as keyframes [27][21]. This method provides better performance than uniform sampling, as the amount of keyframes during an interval is dependent on the information density. An action movie would result in more keyframes than a romantic movie, for example [27]. Clustering based methods generate even less redundant summaries compared to consecutive frame difference based techniques [21]. However, a drawback is that one must determine the amount of clusters beforehand. A more complex method is hierarchical clustering, which obtains clusters on different levels of abstraction as the position in time of the keyframes is also taken into account [13].

3.3.1. Shot based detection
The shot-based detection (SBD) technique segments the video frames into a number of smaller shots by detecting the shot boundaries or transitions. This detection is done by finding dissimilarities of visual content. A transition is detected when a change above a certain threshold is detected.

Different papers have different SBD categorizations. In [32], 44 papers were reviewed for their approaches to segmentation and categorized. The four SBD categories in this paper are Cut-based, Machine learning, Color-based and Entropy-based. In [29], the SBD algorithm consists of three core elements: Frame representation, Dissimilarity measure and thresholding. In the resulting frames, the keyframe extraction is performed. Some methods of SBD are:

- pixel-based technique (PBT)
- histogram-based technique (HBT)
- Statistical-based technique (SBT)
- Edge-based technique (EBT)
- Machine learning based technique (MLBT)

For further analysis, the categorization of [29] is taken since this paper provides a more in-depth and clear explanation of the different methods. In this project, PBT and HBT with partial SBT is used since EBT is less reliable than histogram based algorithms in terms of computational time and performance [29]. Additionally, MLBT are also computationally expensive due to the well-trained network model [29] which makes it inappropriate to implement due to the hardware limitations. Additionally, [22] states: ”The pixel-based methods are highly sensitive to motion of objects.” and ”Histogram-based methods completely lose the location information of pixels.” which should be taken into account.

Partial SBT is selected for the reason that a fully implemented SBT requires a high computation time due to the statistical calculations [29]. However, a variable threshold is desirable due to dynamical nature of videos received as input. Thus, it is decided to partially implement SBT in a optimal balance of performance and computational time.

3.4. Keyframe Extraction methods

3.4.1. VSUMM

Video summarization (VSUMM) is a algorithm which generates summaries based on visual features which has been one of the fundamental unsupervised techniques for video summarization [18]. The visual features of each frame are extracted with the use of color histograms and clustered afterwards using the K-means [25] algorithm as proposed in [11]. In the clusters, one frame is selected and that is identified as the keyframe. Optionally, one can also eliminate the keyframes that are too similar, to refine the summary. the K-means algorithm is chosen because it is one of the most simple unsupervised learning algorithms to solve clustering problems [28].

There are two approaches to select the keyframes of the clusters. With (VSUMM2) and without keycluster selection (VSUMM1) [18]. The approach with keycluster selection (VSUMM2) only selects a frame from a cluster when it is a keycluster [40]. A cluster is a keycluster when the size of a cluster is greater than a certain cut-off point, in [40], half of the average cluster size is used as cut-off point. The selection of the keyframe itself is identical in both approaches, the frame at the smallest distance of the cluster centroid measured in Euclidean distance is selected.

To avoid too similar keyframes, elimination of similar keyframes is done. This is done by comparing the keyframes among themselves with the use of color histograms. If the similarity is lower then a certain threshold, the keyframe is removed from the summary [11]. Additionally, [11] also points out that two frames does not need to be identical in order to be considered too similar.

3.4.2. SIFT

”Scale Invariant Feature Transform (SIFT) has been one of the most prominent local features in computer vision.” [18] The descriptors of SIFT are robust to be used as local features since they are partially invariant to illumination and fully invariant to small deformations, rotations, translations and scaling. There are several variations of SIFT used for keyframe extraction as proposed in [38] and [3] which are also considered. However, the basic/general method as proposed in [18] is chosen because of the room of tailoring it offers and discussing every variation would be outside the scope for this thesis.

The proposed SIFT in [18] first defines important localizations with the use of smoothed and resized images in scale space and applying difference of Gaussian function to determine the maximum and minimum responses. To ensure a collection of interesting and distinct keypoints, non maxima suppression is done and the putative matches are discarded. The dominant orientation of the localized keypoints are found by dividing the image into patches and performing histogram of oriented gradients (HOG).The keypoints extracted are the local features and a threshold is taken which takes a certain percentage of the total video frames as keyframes.

3.4.3. Histogram 3x3 block clustering

A different method using clustering is apply a dynamic clustering method making use of singular value decomposition (SVD) [19]. The algorithm separates every video frame into blocks (Figure 3.3) and concatenates their flattened 6-bin color histograms (creating a feature vector of $216 * 9 = 1944$ elements each). These are then stacked into a matrix and then transposed (creating a $1944xframe\_count$ matrix). After performing SVD (with 63 singular values) to reduce the matrix dimension, the first two frames are added to a cluster and the mean is taken as the centre. For the remaining frames, a frame is added to the last created cluster (and the centre is updated) if it is above a preset Cosine Similarity threshold. If this is not the case, a new cluster is formed. If there are 25 frames or more frames in a cluster, the last frame in the cluster is taken as a keyframe. The advantages of this type of dynamic clustering is that the amount of clusters does not need to be determined beforehand and not all clusters will result in a corresponding keyframe. The drawback is that the clusters are formed chronologically, if frames are very similar but far apart, they will be added to different clusters.

H1	H2	H3
H4	H5	H6
H7	H8	H9

Figure 3.3: 3x3 frame separation into 9 different histogram vectors.

3.5. Evaluation methods

3.5.1. Fidelity measure

Although the *Fidelity measure* as introduced in [7] (and utilized in [21], [33] and [22]) does not directly compute the evaluation metrics (Section 3.2), it does provide an algorithm with the ability to compare sets of keyframes extracted based on visual frame descriptors. The Fidelity measure can be considered an objective general purpose summary evaluation, which can be applied to all video sequences regardless of genre and does require human summaries [7]. It makes use of color histograms, wavelet statistics and edge direction histograms to form a frame difference measure. Taking $V_{seq} = [F_1, F_2, ..., F_N]$ as the set of all frames in the input video and $KF = [KF_1, KF_2, ..., KF_M]$ as the set of extracted keyframes, the Fidelity measure is computed using the following equations.

$$Fidelity(V_{seq}, KF) = MaxDiff - DIST(V_{seq}, KF) \tag{3.5}$$

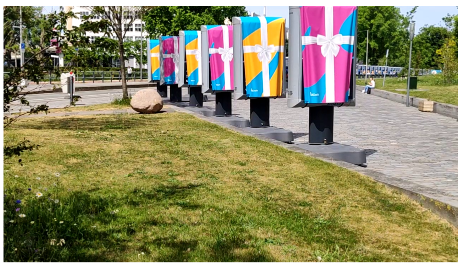
$$DIST(F, KF) = Min\{\ Diff(F, KF_j)\ \}, \quad j = 1\ to\ M \tag{3.6}$$

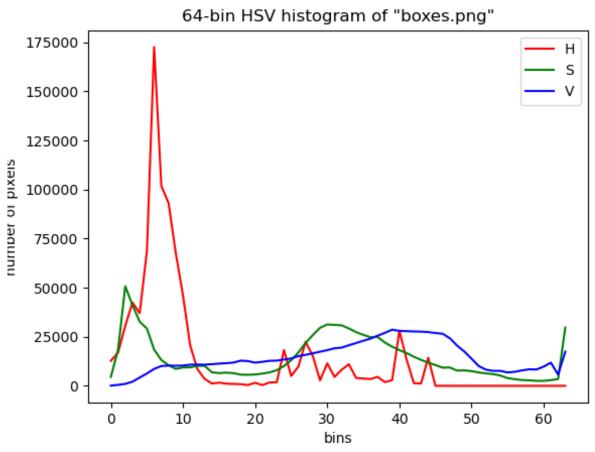
$$DIST(V_{seq}, KF) = Max\{\ DIST(F_i, KF)\ \}, \quad i = 1\ to\ N \tag{3.7}$$

The frame difference measure is computed using eq. 3.8, from which it becomes apparent that a high increase in Diff is only achieved when two ore more feature descriptors have high difference values [6]. This means that with the use of three descriptors, only if the changes in the sequence are significant in terms of color, texture and edges d_{HWD} will result in high values [6]. Generally, a single descriptor does not capture all pictorial details needed to estimate the changes in frames [7] Although high Fidelity values indicate a good representation of the visual content of a video, it does not focus on local details [23].

$$Diff(\cdot) = D_{HWD} = d_H d_w + d_W d_D + d_D d_H \tag{3.8}$$

For the difference measure of histograms (d_H), 64 bin HSV (Hue, Saturation and intensity Value) color space histograms are used [7]. The intersection measure of the two histograms of the frames to be compared form d_H.


(a) *boxes.png*



(b) HSV color histogram

Figure 3.4: *boxes.png* and corresponding HSV histogram with 64 bins

The edge direction histogram based difference (d_D) uses 72 bins with intervals of 2.5°. This type of histogram is used to detect the gradients of pixels, and in particular horizontal and vertical edges. If the Sobel filters [31] as shown in eq. 3.9 and 3.10 exhibit a gradient above a threshold, they are added in the bins (after computing the gradient angle using eq. 3.12). This threshold is heuristically set at 4% of the maximum gradient value to remove background noise [7].

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{3.9}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.10}$$

$$\theta = \sqrt{\arctan \frac{G_y}{G_x}} \tag{3.11}$$

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.12}$$

In computer vision, the magnitude $|G|$ is often computed using $0.5(|G_x| + |G_y|)$ in order to save computation time. The results of applying the Sobel filters can be seen in Figure 3.5 and Figure 3.6. As opposed to the HSV histograms, the difference measure d_D of two Sobel histograms is computed using the Euclidean distance [7] after normalizing for the amount of pixels.

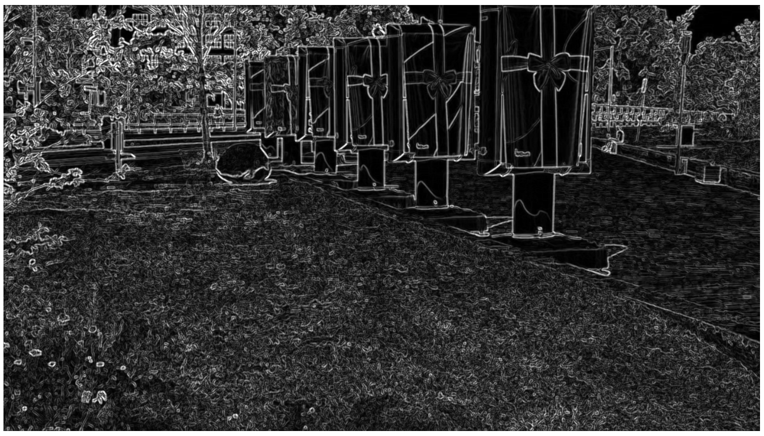


Figure 3.5: Built-in OpenCV Sobel filters applied to *boxes.png* and plotted for magnitude

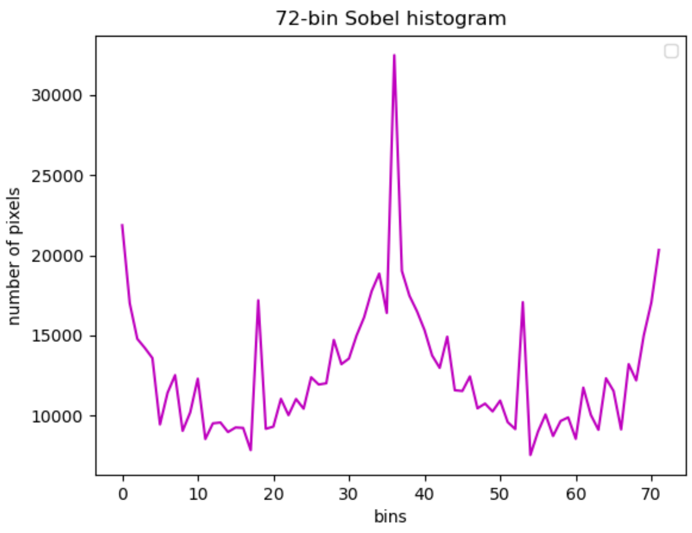


Figure 3.6: Corresponding 72-bin (180 degrees) Sobel histogram

The third descripor difference d_W is computed by multiresolution wavelet analysis [7]. This provides information about the overall texture of the image at different levels [6].

3.5.2. Shot reconstruction degree

In [23], a criterion to evaluate keyframes called the shot reconstruction degree (SRD) is proposed. This criterion focuses more on local details and the evolution trend of a video shot, as compared to the Fidelity. The better the shot cane reconstructed through the interpolation of the keyframes, the higher the shot reduction degree. According to [23], a high SRD will guarantee good information maintenance and retainment of motion dynamics of the original video. For this project however, the importance of capturing motion dynamics and local details in the selected keyframes is considered less than capturing as much instances as possible that appear in a video. Although it is therefore questionable if SRD is the most valuable evaluation method, it is stated that a high SRD will also lead to a high fidelity. The SRD has local evaluation and the Fidelity employs a global strategy [7].

3.5.3. SumMe

A popular benchmark for video summaries is SumMe [14]. This benchmark makes use of 25 videos varying in content consisting of one shot, annotated by 15 users each for ”interestingness”. Users were instructed to select ranges of the video of around 5 seconds to form a summary within 5-15%

of the entire video/shot. This data is inserted into a matrix of $frame\_count * user\_amount$ of which the columns can be averaged to form an array that displays the likely-hood for all frames in the shot of being a keyframe (gt_score). This provides a method for determining the Precision and Recall, and consequently the F-measure. An example of this is shown in Figure 3.7.

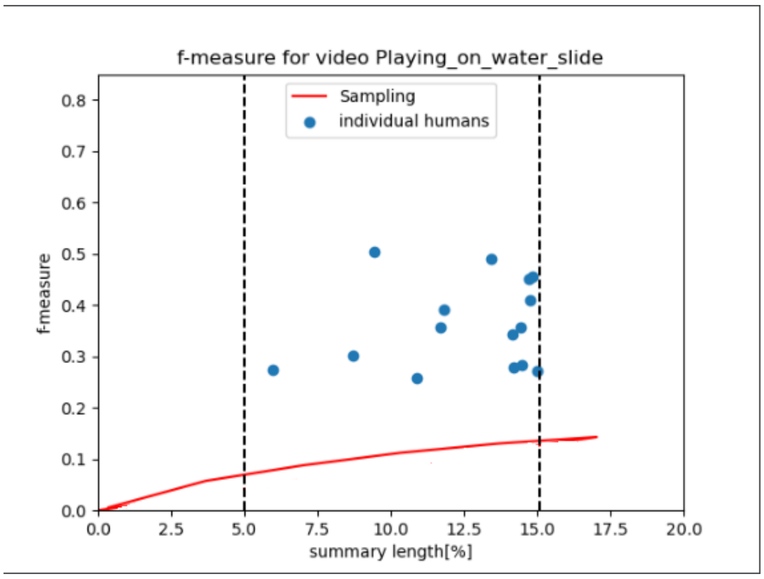


Figure 3.7: F-measure for *Playing_On_Wateslide.mp4*

Although the SumMe benchmark is a valuable tool in video summarization, it was not considered suitable for this reports applications. The reasons for this are that the evaluation is done for summary lengths within 5-15%, while the desired summary length is much lower than 5% and that the users that annotated the videos selected segments based on ”interestingess”, which is a vague determination of salient content. Appendix B shows more details of this benchmark.

4 Design

In this chapter, the design of various methods and techniques discussed in Chapter 3 are laid out. The performance metrics will be evaluated for all methods and compared in the last section for a general comparison. Most tables with results are compacted to show the most interesting data in order to sustain readability. The complete tables can be found in Appendix A. All results were performed on a Intel Core i7-7700HQ CPU @ 2.80GHz in combination with a NVIDIA Quadro M1200 GPU. Although the extracted keyframes indices are deterministic, the execution time will be lower for a better GPU.

4.1. Code development

To read and process frames from videos the OpenCV Python library [1] is often used throughout the code for this design. This library is very prominent in the Computer Vision field.

4.1.1. Random Access Memory management and data allocation

When striving for a low computation time, one should want to read in a frames from a video only once for further processing and give as output to the Feature Extraction module to prevent unnecessary repetition and thus computational delay. However, storing RGB-data of frames into an array would quickly scale the amount of RAM needed with increased video duration. For a video with a resolution of $1280x720$ pixels, each frame would require roughly $2.8\ megabytes$. The requirement given in the PoR (Table 2.2) that only $8GB$ RAM is available would mean that only 2850 frames can be stored at a time. This can be problematic for videos of large duration and resolution. As the array with the RGB-data is the main output of the module, it determines the size of the input video. 2850 keyframes with the assumption of a minimum compression of 85% and a video with a resolution of $1280x720$ and frame-rate of $30fps$ would mean that the duration of the video is restricted to 10.55 minutes. Furthermore, the intermediate steps for determining the keyframes can also not make use of more than 2850 frames in an array at a time, meaning that frame descriptors must be generated in parallel with Shot Based detection, if is shot detection is used. If a video would exceed this limit, the choice could be made to process videos into parts for the entire video search engine.

4.2. Retrieving (key)frame data from indices

In this design, it has been chosen to decode the the data from the keyframe indices once more from the input video, for the reasons mentioned in 4.1.1. This means there is some repeatability involved which could increase the computation time of the sub-module, but at the benefit of reduced complexity and increased video duration capacity. When retrieving RGB-data from indices of keyframes (or by sampling), one can choose to read decode RGB-data from indices using the built-in OpenCV function $CV2.CAP\_PROP\_POS$ or by using $grab()$ and $retrieve()$. The plot of the computational time for both ways is plotted in Figure 4.1. For rates higher than 0.7 fps, or on average every 43tĥ frame, or a CR of 97.6 %, it is faster to read all frames and discard the frames that are not desired. This switch-case has been implemented in the code. This plot also shows that for keyframe selections with a Compression Ratio above this threshold, the overall computation time of the system decreases with increased Compression Ratio (aside from the decreased computation time for the entire video search engine).

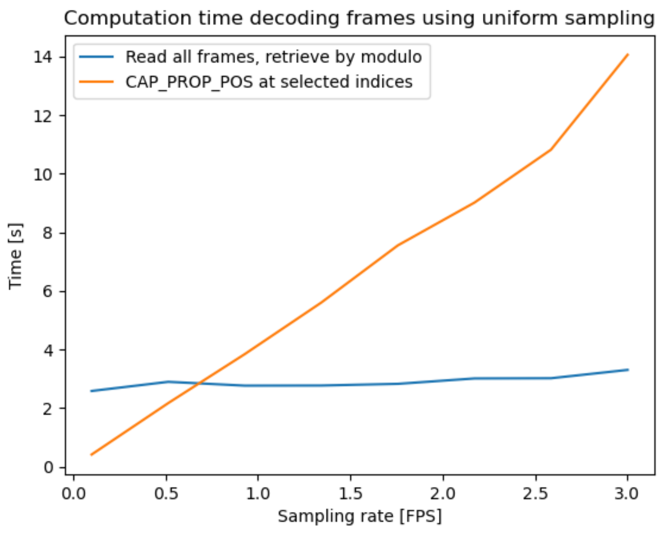


Figure 4.1: Comparison in speed for the amount of indices selected (video: *postbus.mp4*)

4.3. Selected videos for demonstration of results

For evaluating methods for performance while maintaining a good overview of the characteristics for different types of videos, 4 videos where created, varying in content, duration, and amount of shots. The content overview of them can be seen in the following figures. These video can be found in the GitHub video folder [4].

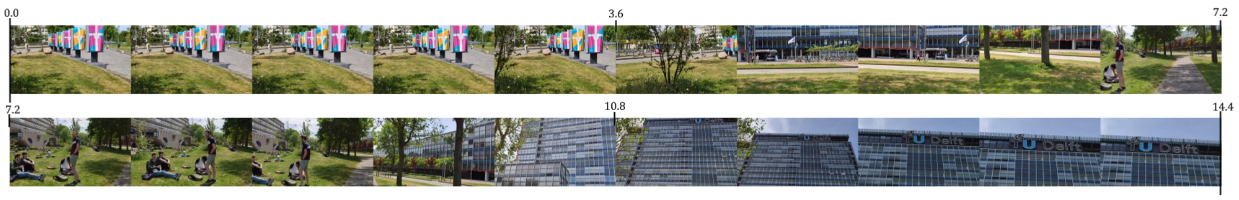


Figure 4.2: Video content overview of *ewi-tudelft.mp4* (14.44s, 30fps, 1280x720p), which consists of one continuous segment (shot).

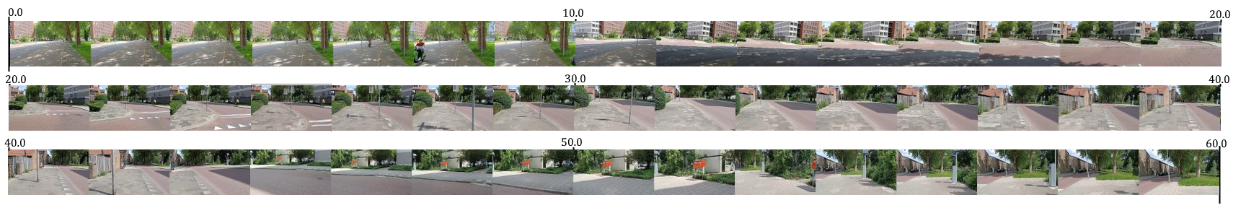


Figure 4.3: Video content overview of *mailbox-street.mp4* (14.44s, 30fps, 1280x720p), which consists of one continuous segment (shot).

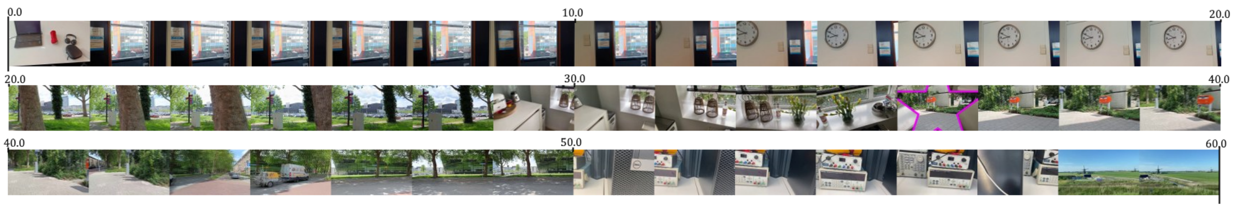


Figure 4.4: Video content overview of *multishot.mp4* (14.44s, 30fps, 1280x720p), consisting of 8 different shots with 7 different transitions: 5 abrupt transitions, 1 *star-swipe* transition and 1 smooth fade transition.

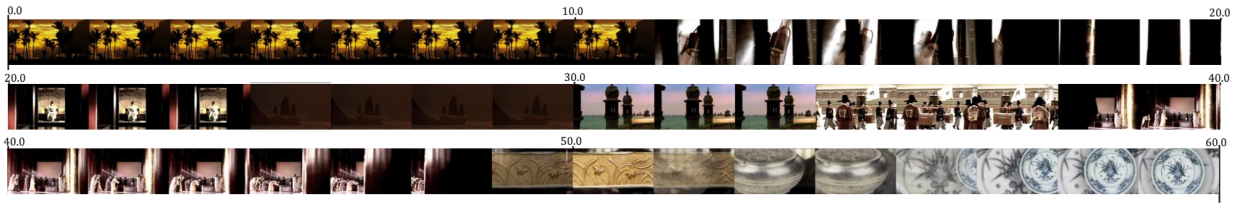


Figure 4.5: Video content overview of *zheng-he.mp4* (14.44s, 30fps, 1280x720p), a clipped video taken from the Engineering Historical Memory video database [26] consisting of 10 different shots with 9 different transitions: 5 abrupt transitions and 4 smooth fade transitions.

4.4. Fidelity implementation

The implementation for computing the Fidelity measure is highly comparable to the proposition in Section 3.5.1. While the histogram difference d_H does not differ from the analysis in any way, the difference descriptor d_D using Sobel filters makes use of a Gaussian blur filter proceeding the histogram to reduce the noise (instead of a 4% magnitude threshold). This is not considered to influence the output of the Fidelity measure. Both descriptors were normalize to have their values mapped between [0, 1]. Unfortunately, the wavelet descriptor d_w required extensive analysis to implement, which the time constraint for this project did not allow. The resulting d_{HWD} was thus calculated using eq. 4.1.

$$Diff(\cdot) = D_{HWD} = d_D d_H \tag{4.1}$$

4.5. Shot Based Detection

Because of the dynamic nature of videos, different shots could have different amount of 'action'. This means that if a variable threshold and a video with very active and inactive shots are used. There is a chance that the action of the relative active shot will overwhelm the inactive ones, leaving some whole shots extracted. In order to prevent this, shot based detection (SBD) is done before the keyframe extraction. This will separate each shot from one another so that the keyframe extraction can extract the keyframes of each shot individually. This section discusses the methods of SBD used and implemented which is, as mentioned in 3.3.1, PBT and HBT with partial SBT.

4.5.1. HBT

The HBT as proposed in [18] starts with reading the RBG values of the frames. The histogram will be computed with the use of the function $cv2.calcHist()$ and with the histogram values extracted, the following equation is used:

$$\frac{\sum_{c\in\{channels\}}\sum_{b=0}^{bins}\left|H(n,c,b)-H(n-1,c,b)\right|}{2*|pixels|*|channels|} > \tau \tag{4.2}$$

Where $H(n,c,b)$ and $H(n-1,c,b)$ are the values of bin b of the color channel c in the histogram of the current and previous frame, $pixels$ the total number of pixels, $channels$ the number of channels (3 in this case) and τ the threshold.

HBT threshold
To make the algorithm viable for a wider range of action in the videos, a variable threshold is required. However, a total SBT implementation has a high computational time [16]. Hence, a simplified approach is taken as a middle ground to minimize the computational time while maximizing the robustness for different videos. The threshold τ is taken as:

$$\tau = factor * average_{hist-diff} \tag{4.3}$$

Where $factor$ is a multiplication factor and $average_{hist-diff}$ the average histogram difference. After some testing, a factor of 6 is taken since it returns the best results in a wide range of non-dynamic and dynamic videos. The resulting comparison will be done to determine the shot boundaries:

$$H_{diff}(n) > \tau = factor * average_{hist-diff} \tag{4.4}$$

Thus, a frame is classified as a shot boundary if the histogram difference is higher than a multiplication factor multiplied by the average histogram difference. However, after testing this algorithm in a wide variety of videos, it became apparent that this threshold lacks in performance in longer videos. The threshold performed well at the start since it was firstly tested on shorter videos (duration of the video was less than one minute). Hence, the addition of the CFAR threshold is also added in HBT which is discussed in 4.5.2 threshold section which explains this threshold in detail.

4.5.2. PBT

The PBT proposed by [29] first converts the image to a gray-image. After the conversion, the intensity of each pixel is summed together and subtracted by the summation of the previous frame. Lastly, the value is normalized and compared to a certain threshold, if the value is higher, that frame will be labeled as a shot boundary. The equation is shown below:

$$\frac{\left|\sum_{x=1}^{row}\sum_{y=1}^{col} k(m,x,y) - \sum_{x=1}^{row}\sum_{y=1}^{col} k(m-1,x,y)\right|}{256 * row * col} > \tau \tag{4.5}$$

Where $k(m,x,y)$ and $k(m-1,x,y)$ are the intensity values of each pixel of the current and previous frame, row the number of pixel rows in the frame, col the number of pixel columns in the frame and τ the threshold. However, this method is very inaccurate since it only uses the intensity difference of the whole frame instead of pixel by pixel. So if the intensity in the frame itself shifts while the overall intensity stays the same, a boundary will not be detected. This is why it is chosen for this implementation instead:

$$\frac{\sum_{x=1}^{row}\sum_{y=1}^{col} \left|k(m,x,y) - k(m-1,x,y)\right|}{256 * row * col} > \tau \tag{4.6}$$

This computed the intensity differences pixel-by-pixel and thus, will also detect shots when the intensity changes in certain regions while the overall intensity stays the same.

PBT threshold
To increase the robustness of the algorithm, a variable threshold is needed. There are two versions made for variable threshold. The first one is based on the threshold of HBT as discussed in 4.5.1, using the same approach. For the PBT proposed by [29], the same factor of 6 is taken. But for the modified method a multiplication factor of 2 is taken instead since the peaks are more defined. Part of the frame differences and its threshold of $zheng$-he.mp4 is plotted and shown below in figure 4.6 and 4.7. The blue line represents the frame difference and the green line is the threshold taken.

As it can be seen from figure 4.6 and 4.7, due to the fade transition around the 450th frame, the proposed algorithm will return false boundaries. Additionally, the differences in the region around the 200th and 300th frame, which is a relatively high dynamical shot compared to the rest of the video, comes very close to the threshold. Furthermore, in the modified method, not only will the algorithm return false boundaries at around the 450th frame, it would also return false boundaries at the region between around the 200th and 300th frame. However, setting the threshold lower will result in completely not detecting the fade transition at around the 450th frame. The second variable threshold is implemented in order to prevent this.

The second variable threshold approach is based on a Continuous False Alarm Rate (CFAR) threshold as proposed in [39]. This threshold is designed to find the correct peaks in noisy circumstances. The noise in this case is the fact that some shots could be significantly more dynamical compared to other shots, which means that the algorithm could label multiple false shot boundaries within the dynamic shots if only an average threshold (as discussed above) is used, which was not the case with HBT.

The CFAR threshold averages out a region of the differences by convolving the frame differences with an array of n ones. The frame differences is defined as:

$$D_{proposed} \left| \sum_{x=1}^{row} \sum_{y=1}^{col} k(m,x,y) - \sum_{x=1}^{row} \sum_{y=1}^{col} k(m-1,x,y) \right| \tag{4.7}$$

$$D_{modified} = \sum_{x=1}^{row} \sum_{y=1}^{col} \left| k(m,x,y) - k(m-1,x,y) \right| \tag{4.8}$$

Where $D_{proposed}$ is used for the method proposed by [29] (see eq. 4.5) and $D_{modified}$ for the modified method (see eq. 4.6). It is chosen to convolute the frame differences with:

$$h = [0.5, 0.5, 0.5, 0.5, 0.5] \tag{4.9}$$

Which is equal to an array of 5 elements all consisting ones and the result divided by 2. This results in a 'multiplication' factor of 2.5 which is chosen based on the results of multiple tests. A higher value will result in not detecting a shot since the threshold is too high and a lower value will result in detection of false boundaries in shot transitions. Additionally, the array size of 5 is chosen since it is a balance of averaging while remaining the required steepness in the threshold. This second threshold is also plotted and shown below in figure 4.6 and 4.7.

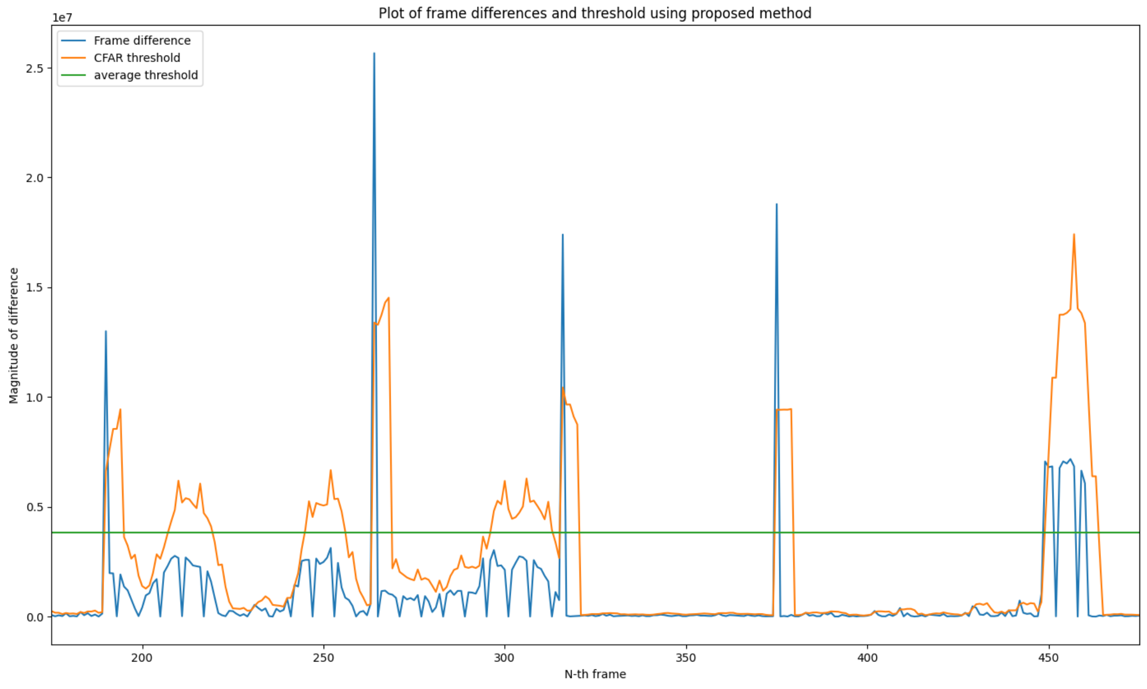


Figure 4.6: The frame differences and the thresholds of *zheng-he.mp4* between the 175th and 475th frame of the proposed method and $factor$ = 6

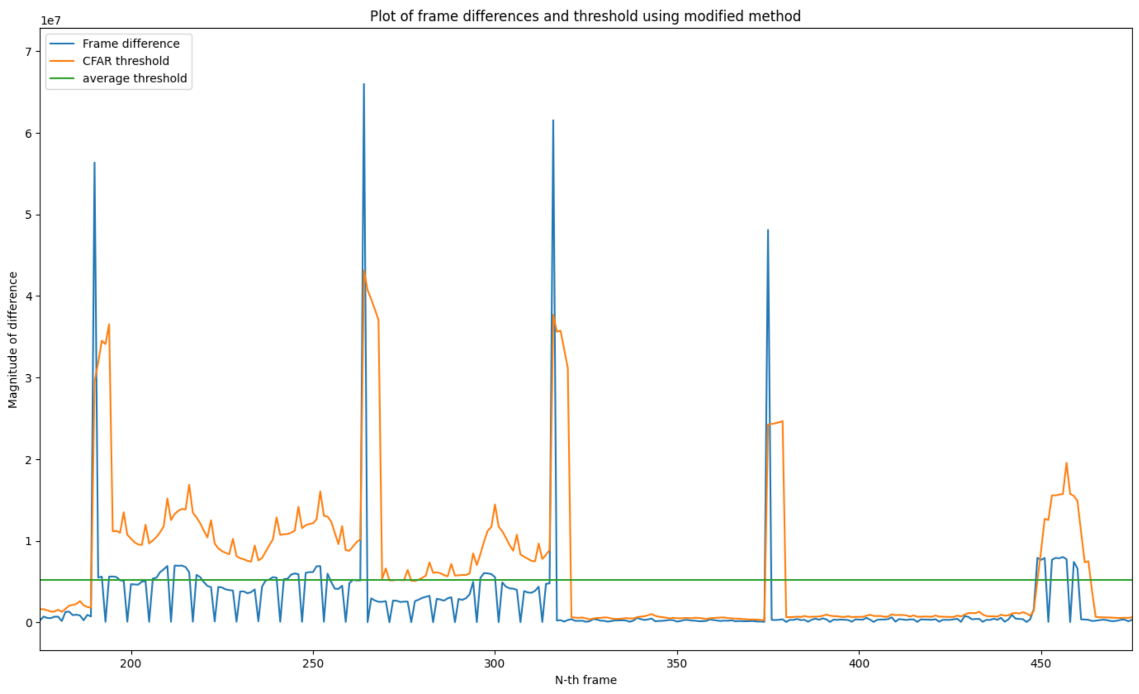


Figure 4.7: The frame differences and the thresholds of *zheng-he.mp4* between the 175th and 475th frame of the modified method and $factor$ = 2

As it can be seen from figure 4.6 and 4.7, the second threshold takes care of the 'noisy' peaks and the first threshold eliminates the 'white noise'. Thus, in order for a frame to be considered as a shot boundary, the frame difference has to be greater than the first and second threshold, or:

$$D_{method} > \tau = max(\tau_{version1}, \tau_{version2}) \tag{4.10}$$

Where D_{method} is the frame difference of the method, $\tau_{version1}$ is the threshold based on averaging and $\tau_{version2}$ the threshold resulted by convoluting.

4.5.3. SBD Results

The algorithm is tested with the use of videos of the TRECVID 2001 dataset. In the data-set the videos and truth tables are provided. The videos are described and shown below in table 4.1.

File Name	Video Title	Number of Frames	Cut transitions
anni005	“NASA_25th_Anniversary-Show_Segment_5”	11,363	38
anni009	“NASA_25th-Anniversary-Show_Segment_9”	16,587	38
nad31	“Spaceworks - Episode 6”	52,405	187
nad33	“Spaceworks - Episode 8”	49,768	189
nad53	“A&S_Reports_Tape_#4_-_Report_#260”	26,115	83
nad57	“A&S_Reports_Tape_#4_-_Report_#264”	12,781	44

Table 4.1: Videos of the TRECVID 2001 data-set used

It has to be noted that the videos of the data-set does not only consist of 'cut'-transitions but also other different kinds of transitions. However, since the designed SBD is not optimized for a 'fading' transition, only 'cut'-transitions or abrupt transitions are considered. A few of the results are shown below in table 4.2, a complete overview can be found in Appendix A.1.

Video	HBT			HBT + CFAR			PBT proposed			PBT modified		
Name	R	P	F	R	P	F	R	P	F	R	P	F
anni005	66.7	42.6	52	86.8	49.3	62.9	84.2	10.3	18.4	100	19.5	32.6
anni009	50	47.5	48.7	81.6	59.6	68.9	94.7	12.2	21.6	100	82.6	90.5
nad31	89.3	70.2	78.6	94.7	71.1	81.2	94.1	21.2	34.6	95.7	83.3	89.1
nad33	95.8	79	86.6	100	82.5	90.4	95.8	32.8	48.9	95.8	91	93.3
nad53	87	60.8	71.6	97.6	72.3	83.1	94	32.6	48.4	97.6	89	93.1
nad57	93.2	95.1	76.7	100	74.6	85.5	95.5	30	45.7	100	69.8	82.2
average	80.3	65.9	69	93.45	68.2	78.7	93.1	23.2	36.3	98.2	72.5	80.1

Table 4.2: Performance of SBD methods. Both PBT methods use the CFAR threshold as standard. ”R” is recall, ”P” precision and ”F” the F-measure

Note that the precision for all the methods are generally low, this is because in the videos itself, there are also other shots transitions, however, those are not counted as shot boundaries for the simplicity of testing and the fact that the algorithm is not yet optimized for fading transitions since it is focused on ’cut’-transitions. Additionally, in the later part of ”anni005” the frames begin to stutter, which explains the high amount of false detected shot boundaries detected by the modified PBT. However, even though the modified PBT had a video with bad results, the F-measure is still the highest of all the four methods. Furthermore, it is also noticeable that adding the CFAR threshold greatly improves the performance of HBT. Lastly, the speed performance per frame is shown below in table 4.3, a complete overview can be found in Appendix A.1.

File Name	Video Title	HBT	HBT + CFAR	PBT proposed	PBT modified
anni005	“NASA_25th_Anniversary-Show_Segment_5”	0.388	0.429	0.351	0.463
anni009	“NASA_25th-Anniversary-Show_Segment_9”	0.265	0.266	0.252	0.341
nad31	“Spaceworks - Episode 6”	0.371	0.373	0.351	0.526
nad33	“Spaceworks - Episode 8”	0.347	0.35	0.428	0.546
nad53	“A&S_Reports_Tape_#4_-_Report_#260”	0.354	0.361	0.406	0.51
nad57	“A&S_Reports_Tape_#4_-_Report_#264”	0.333	0.352	0.369	0.495

Table 4.3: Average computational time of different SBD methods per frame in milliseconds

It is noticeable that HBT with the implementation of the CFAR threshold only takes slightly more time per frame (except for ”anni005”). The modified PDT takes longer compared to the proposed PBT since it compared every single pixel with each other and thus, this difference is expected. Lastly, even though the modified PDT returns the most accurate results as seen in table 4.2, HBT with CFAR threshold serves as an alternative when a faster SBD is preferred.

4.6. Keyframe extraction

4.6.1. Uniform sampling baseline

Uniform sampling with $5fps$ is used as a baseline to compare all other methods to (corresponding to a Compression Ratio of 83.3%, as mentioned in the elaboration of the time constraint in Section 2.2. Table 4.4 shows the performance for the four selected videos. The third column shows the average computation time per second of input video.

Video	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	0.98	0.068	72	83.13	0.982
mailbox-street	3.22	0.054	301	83.18	0.992
multishot	1.06	0.035	150	83.35	0.885
zheng-he	0.97	0.032	150	83.33	0.8275

Table 4.4: Uniform sampling at $5fps$. The times are calculated using the average of 10 runs (n = 10).

Table 4.5 shows the results for a sampling rate corresponding to a CR of 98%, which is estimated to be close to the CR of other methods. It is immediately apparent that this reduces the computation time for all videos, but the Fidelity value as well.

Video	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	0.63	0.044	9	97.89	0.842
mailbox-street	2.52	0.042	37	98.93	0.947
multishot	0.53	0.018	18	98.00	0.695
zheng-he	0.5	0.017	18	98.00	0.634

Table 4.5: Uniform sampling for a Compression Ratio of 98%. The times are calculated using the average of 10 runs (n = 10).

4.6.2. Crude histogram matching

A simple histogram matching algorithm was implemented that compares each frames' histogram to the previously selected keyframe. The histogram similarity value was computed using the Bhattacharyya distance (eq. 4.11) and the threshold was heuristically set at 0.30. Shot detection does not add any value to this type of keyframe extraction, thus it has been omitted in Table 4.6 as it only increases the computation time.

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{\overline{H}_1 \overline{H}_2 N^2}} \sum_I \sqrt{H_1(I) * H_2(I)}} \tag{4.11}$$

Video	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	3.47	0.241	13	96.96	0.932
	Yes	2.03	0.141	12	97.19	0.932
mailbox-street	No	11.7	0.195	10	99.44	0.929
	Yes	5.59	0.093	11	99.05	0.916
multi-shot	No	6.55	0.218	33	96.34	0.756
	Yes	2.97	0.099	32	96.45	0.715
zheng-he	No	6.34	0.211	37	95.89	0.837
	Yes	2.81	0.094	28	96.89	0.778

Table 4.6: Performance results of crude histogram matching. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$.

4.6.3. VSUMM

The code that was appended to [18] on GitHub [17] is used as the base for the video summarization (VSUMM). The provided code was user-friendly but not optimized for speed, fast performance and required a manual input for every single shot. Optimizations are made in automation, speed and RAM use. The resulting program reads and computes the descriptors for every frame. The descriptors of VSUMM are generated using $cv2.calcHist$ and the resulting array is a matrix of n_{frames} x $n_{channels}$ * n_{bins}, where n_{frames} is the number of frames of the video, $n_{channels}$ the number of channels and n_{bins} the number of bins. Usually, 16 bins are taken for each channel [18] and for $n_{channels}$ the value of 3 is taken since the channels used are: 'r', 'g' and 'b'. The resulting descriptors are clustered in a variable number of clusters and transformed into cluster-space using the K-means algorithm. The number of clusters is taken as a percentage of the total frame count.

It is decided to not use key-cluster selection (VSUMM2) and to use VSUMM1 instead since it saves computational time and while VSUMM2 does give a higher precision compared to VSUMM1, it lacks in recall compared to VSUMM1 [34] which is more important for a implementation in a search engine. Furthermore, in the code provided, there was a option to create 3-D Tensor histograms but that has been discarded since it required too much computation time (due to hardware limitations).

Additionally, to potentially save computational power and time, two versions are made: one which do not compare the extracted keyframes and eliminating those which looks similar and one which does. This last step is done with the crude histogram, hence, it is a 'combination' of two keyframe extraction techniques. The version without comparison extract a maximum of 2% of the total keyframes and the version with a maximum of 5%. This is done since this version eliminates similar frames and otherwise the first version would return too much similar frames. A short summary of the results are shown below in table 4.7 and 4.8. A complete overview of the results can be found in appendix A table A.6 and A.7.

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	Yes	1.29	0.089	2	99.53	0.773
	Yes	Yes	1.45	0.100	2	99.53	0.773
mailbox-street	No	Yes	5.29	0.088	12	99.33	0.928
	Yes	Yes	5.65	0.094	10	99.44	0.929
multi-shot	No	Yes	2.19	0.073	5	99.45	0.389
	Yes	Yes	2.91	0.097	7	99.22	0.632
zheng-he	No	Yes	2.32	0.077	5	99.44	0.477
	Yes	Yes	3.20	0.106	8	99.11	0.570

Table 4.7: Performance results of VSUMM. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$ and maximum of 2% of the frames are extracted as keyframes

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	Yes	1.79	0.124	8	98.13	0.871
	Yes	Yes	1.95	0.135	8	98.12	0.908
mailbox-street	No	Yes	5.26	0.088	11	99.39	0.932
	Yes	Yes	6.27	0.104	12	99.33	0.932
multi-shot	No	Yes	2.92	0.097	21	97.67	0.709
	Yes	Yes	3.55	0.118	23	97.45	0.761
zheng-he	No	Yes	2.96	0.099	25	97.22	0.806
	Yes	Yes	3.83	0.127	22	97.56	0.714

Table 4.8: Performance results of VSUMM in combination with crude histogram matching. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$ and a maximum of 5% of the frames are extracted as keyframes

4.6.4. SIFT

Originally, the code was based on the code provided from Github [17], however, after several tests, it was determined that this implementation is far too slow to be used in this application (even after optimization, the run time was several times longer than the video itself). Hence, it is decided to not continue with this method. However, in the same method, the usage of color moments are used to extract the keyframes which is computationally less draining and significantly faster compared to the original SIFT. The method using color moments is discussed below in section Color moments.

4.6.5. Color moments

This method is found and implemented because the initial SIFT method was too computationally expensive to extract keyframes. This method also computes the descriptors with the use of $cv2.calcHist$, however, instead in rgb, this is done in gray gray-scale. From the descriptors, the color moments are computed. The color moments computed are: mean, standard deviation and the skewness. Originally, the Euclidean distance of the color moments are calculated and the differences are ranked from the greatest to lowest and the top number of frames are taken as keyframes, where the number of frames taken depends on the percentage of the total frames to be extracted. However, this approach resulted in many redundant keyframes. Hence, it is implemented that it will compare the euclidean distance with a threshold τ where:

$$\tau = factor * average_{euclidean} \tag{4.12}$$

Where factor is a factor of multiplication and $average_{euclidean}$ the average euclidean difference. Frames with a higher euclidean distance compared to the threshold will be taken as keyframes. Lastly, to make it more robust, it compares the number of keyframes taken with the percentage of the total frames to be extracted and if it higher, the original approach is used. Thus, it will never return more keyframes, but it can reduce the keyframes taken.

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	No	3.41	0.237	8	98.13	0.539
	Yes	No	3.91	0.271	8	98.13	0.539
mailbox-street	No	No	13.4	0.223	35	98.04	0.903
	Yes	No	15.8	0.263	31	98.27	0.903
multi-shot	No	No	5.65	0.188	17	98.11	0.356
	Yes	No	7.51	0.25	16	98.22	0.622
zheng-he	No	No	5.59	0.186	10	98.89	0.565
	Yes	No	7.86	0.262	12	98.67	0.647

Table 4.9: Performance results of color moments. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$ and factor is set at 6

4.6.6. Histogram 3x3 block clustering

In [19], the $MxN = 1944 * frame\_count$ matrix that is generated for a video is decomposed using Singular Value Decomposition with a default number of 63 singular values to compute (k = 63). However, for some short shots in a video, the frame count is lower than 63, making this not possible as the maximum value of k is $Min(M, N)$. The algorithm is adjusted to set $k = frame\_count$ if the frame count is lower than 63.

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	Yes	1.60	0.111	6	98.60	0.802
	Yes	Yes	1.78	0.123	6	98.60	0.802
mailbox-street	No	Yes	6.42	0.107	16	99.11	0.931
	Yes	Yes	6.62	0.110	16	99.11	0.931
multi-shot	No	Yes	2.66	0.088	19	97.89	0.670
	Yes	Yes	3.13	0.104	19	97.89	0.670
zheng-he	No	Yes	2.32	0.077	9	99.00	0.440
	Yes	Yes	3.01	0.100	12	98.67	0.677

Table 4.10: Performance results of Histogram 3x3 block clustering. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$.

4.7. General performance

To put the performance of different methods into perspective, the results of the previous sections can be seen side by side in Figure 4.8 to 4.10. The three performance metrics, Fidelity, Compression

Ratio and computation time show to form a triangle of trade-offs. For uniform sampling at $5fps$, the execution time is small and the Fidelity measure is high due to the large amount of frames, but the Compression Ratio is very low. A property of the Fidelity measure is that its value for a set of keyframes $KF = [KF_1, KF_2, ..., KF_M]$ with any extra frame added (that is not already in the set) will always be higher than the original set. This means that a just comparison requires the amount of keyframes and thus the Compression Ratio to be equal.

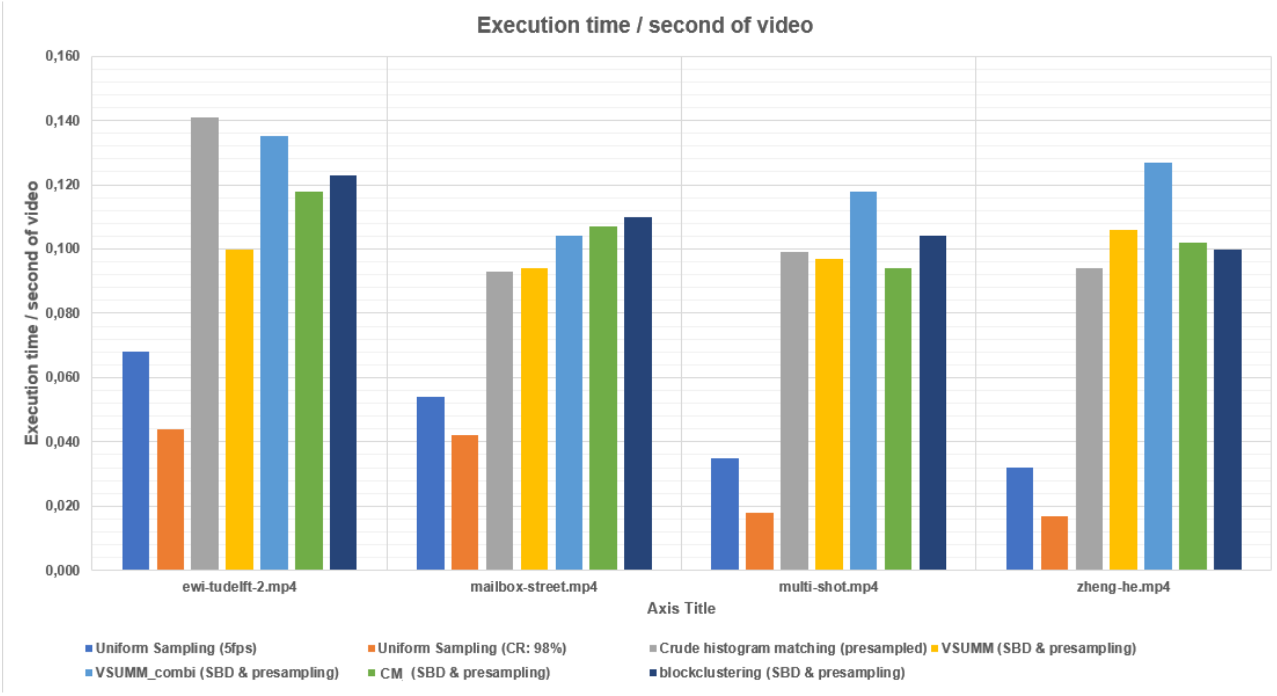


Figure 4.8: Computation time for tested methods.

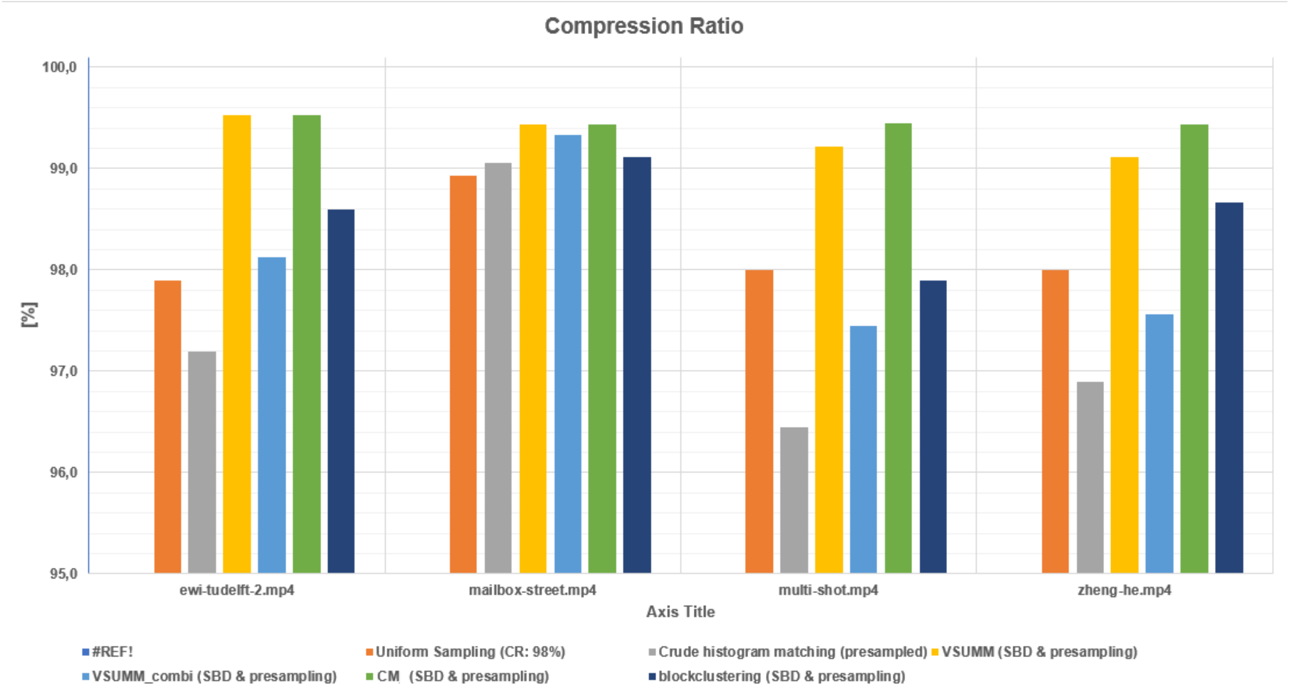


Figure 4.9: Compression Ratio for tested methods (the baseline uniform sampling at $5fps$ (corresponding to a CR of 83.3%) is omitted to magnify the differences

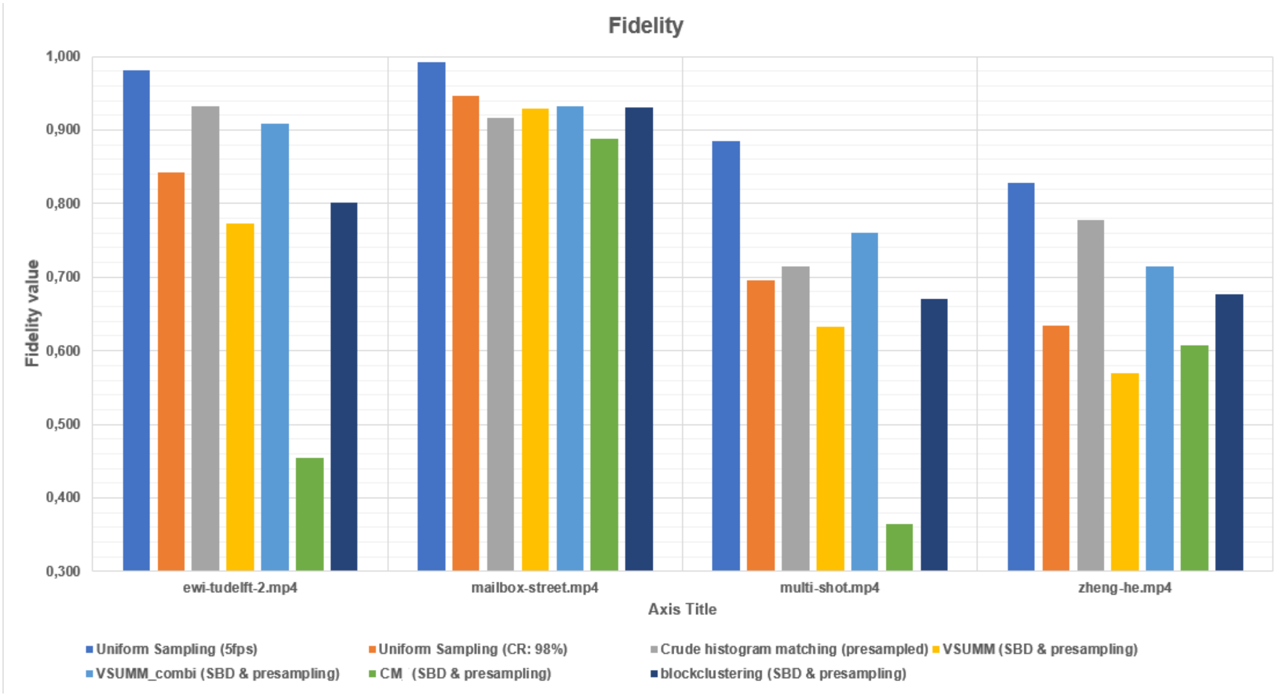


Figure 4.10: Fidelity measure for tested methods.

For uniform sampling at 98% compression, the amount of keyframes is in the neighborhood of the analyzed methods and has the lowest computation time of all, partly due to the reason mentioned in 4.2. Although the Fidelity appears to also be close to the other methods, it can also be attributed to the type of video. *mailbox-street.mp4* consists of one shot with slow panning of the camera at a constant speed. One would expect the desired keyframe indices to then be equidistant, which translates itself in the results. For more dynamic videos with multiple shots that vary in information density (which corresponds more to *zheng-he.mp4*), the Fidelity decreases.

5 Prototype

5.1. Keyframe Extraction prototype

For creating a submodule that is able to apply various methods and options to extract keyframes, a prototype was created that functions for all implemented methods, which can be seen in Figure 5.1. Some parts of the general overview have been omitted, such as the ability to extract the first or last frame from a shot or to print the statistics of the selection of keyframes or execution time of the extraction process. For further work on this code, new methods with corresponding descriptors and settings for presampling and shot based detection can easily be adjusted. The code can be found on the GitHub [4] repository.

5.2. Image-Based Video Search Engine

5.2.1. Implementation

For the prototype of the entire system it is important that it is both easy to use and fast in execution. Otherwise, it is not attractive for other people to use or improve on. To that end the following constraints for the prototype are chosen:

- The three modules will each reside in their own Python sub package, making them easy to develop/maintain individually.
- The final prototype will run as a Python application on a host machine. The end-user will be able to upload the video
- The input to the complete system consists of: a single query video (in mp4 format), and one or more query images (in jpg format).
- The output of the complete system contains the timestamps in which the object of the query image(s) appears.

In- and Outputs

In order to ensure smooth development between the different modules, the in- and outputs of each module will be defined. The scheme can be found in Figure 5.2.

Timings

One of the most important performance metrics of the system is the execution time. In order to optimise the total execution time, the running time of all three modules will be measured so that the modules can be fine-tuned. The most time consuming module will most likely be the Feature Extraction module (FE) (if all frames of the query video will be used). For each of the images received by FE it will run a lot of calculations. A way to reduce this execution time is to reduce the number of images it has to process. This reduction is performed by the Keyframe Extraction module (KFE). The goal of KFE is to reduce the execution time of FE without increasing the total execution time or losing visual content. This is equivalent to minimising Equation 5.1:

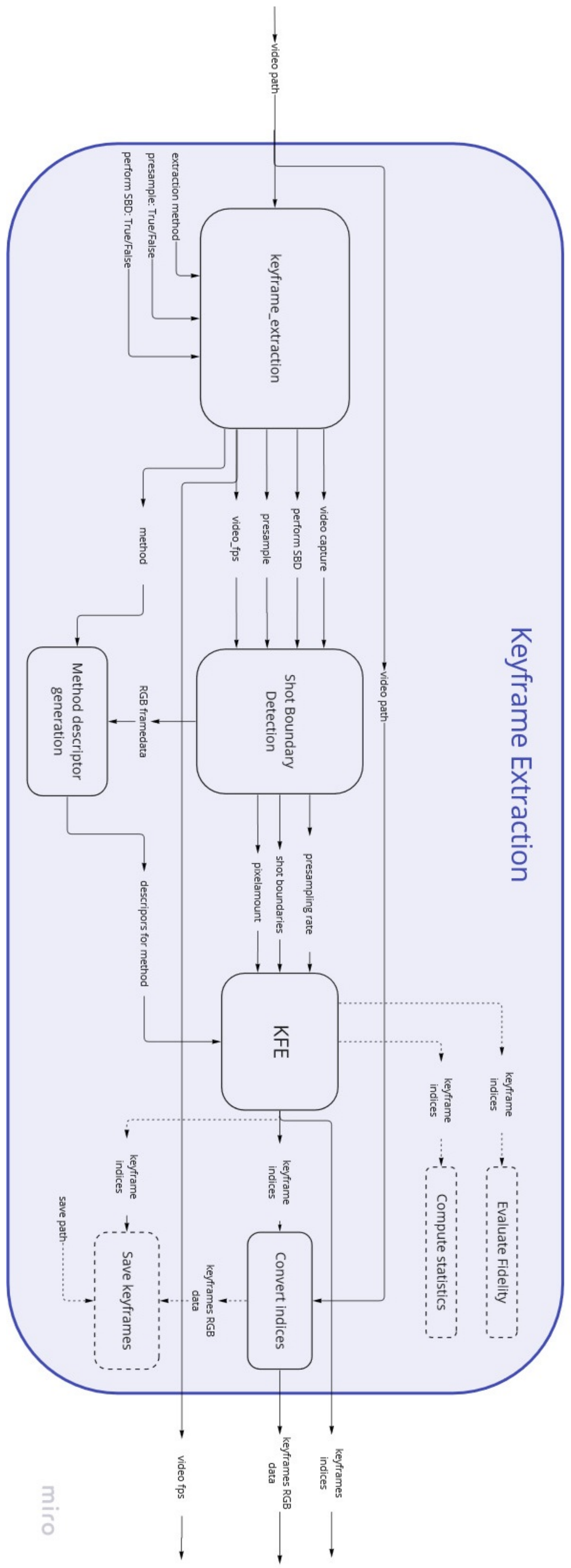


Figure 5.1: Overview of the flow of data in the Keyframe Extraction submodule

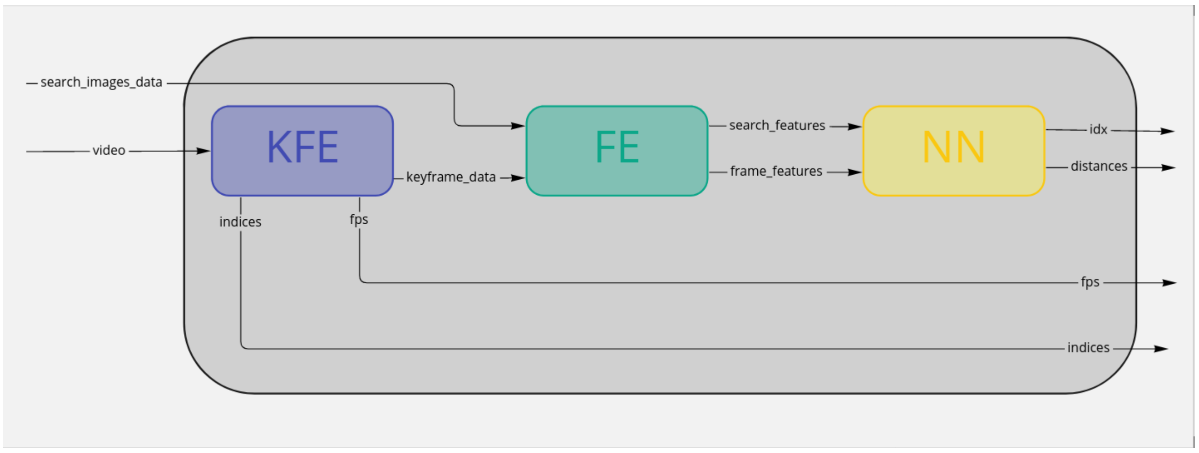


Figure 5.2: The in- and outputs of all the modules.

$$t_{sum} = t_{KFE} + t_{FE} + t_{NN} \tag{5.1}$$

where $t_{[\cdot]}$ corresponds to the execution time of module $[\cdot]$.
For the measurement of the system the overhead of the Python application will not be considered, so only the time measurement of the individual modules will be performed. For each of the modules the time will be measured individually and the total time t_{total} (from the start to the end of the script) of the prototype will be measured. The total execution time is equal to:

$$t_{total} = t_{sum} + t_{overhead} \tag{5.2}$$

where $t_{overhead}$ is the extra time (overhead) of loading in the query images and other calculations performed outside of the modules t_{KFE}, t_{FE} and t_{NN}.
To fulfill system requirement 6, the total time should be converted to a ratio. This can be done using Equation 5.3.

$$\frac{t_{total}}{t_{video}} \leq 0.5 \tag{5.3}$$

where t_{video} is the length of the video.

Precision
The precision is just as important as the execution time. The system could execute really fast, but if the results do not show the correct timestamps of the video then the small execution time has no value. For that reason the mean Average Precision (mAP) will be used to evaluate the precision of the complete system. The mAP will be calculated as explained in the Feature Extraction thesis [36].
The average precision AP_i that is used for the mAP calculation is obtained using the following formula:

$$AP@k = \frac{1}{k}\sum_{n}^{k} P@n \times rel@n \tag{5.4}$$

where k refers to the total number of timestamps at the output of the system (which is 7% of the keyframes according to the DCNNS module [15]), n refers to the rank of the timestamp at the output, $P@n$ refers to the precision@n and $rel@n$ is the relevance function at n. The relevance function equals either one or zero: $rel@n = 1$ if the timestamp at rank n is relevant and $rel@k = 0$ otherwise. The precision@n can be calculated as explained in the Feature Extraction thesis [36].

5.2.2. Validation

Timings
In Table 5.1 the timing results can be found for the 'easy' dataset. For all of the test scenarios, system requirement 6 is satisfied, as can be seen in column 'Ratio'. The column 'Video' shows the name of the

query video in combination with the query image, so *Battuta1_1* corresponds to the first query video of Battuta and the first query image. The dataset of videos as explained in [36] also shows the different query videos with the amount of available query images. Due to time constraints, not all videos and query images were evaluated.

Precision

The performance of the system will not only be described using the ratio between the execution time of the system and the duration of the query video, but also with the help of the mean Average Precision. The mAP was calculated following Subsection 5.2.1. The mAP calculations can be found in Tables 5.2 and 5.3. The tables show the mAP with and without applying the distance filter. The filter reduces the amount of results by discarding all matches above a certain distance threshold and retaining only those above that threshold. By using the filter, the mAP of the system increases significantly.

Table 5.2: mAP results of the prototype for the 'easy' dataset.

	mAP	
	Without Filter	With Filter
Battuta1	0.74	1
Battuta2	0.59	1
Battuta3	0.31	1
Battuta4	0.16	0.67

Table 5.3: mAP results of the prototype for the 'hard' dataset.

	mAP	
	Without Filter	With Filter
Ewi1	0.125	No matches
Ewi2	0.25	1
Dutch mailbox	0.29	1

Table 5.1: Time measurements of the prototype for various query videos and images from the 'easy' dataset. The *Ratio* is defined as in Eq. 5.3. A resize of $1024 \times 576p$ was used [36].

Video	t_{video} [s]	t_{KFE} [s]	t_{FE} [s]	t_{NN} [s]	t_{total} [s]	Ratio
Battuta1_1	261	28.25	75.50	0.0023	103.80	**0.40**
Battuta1_2	261	27.73	76.84	0.0018	104.63	**0.40**
Batutta1_3	261	28.17	76.29	0.0019	104.55	**0.40**
Battuta1_4	261	28.57	75.98	0.0016	104.57	**0.40**
Battuta2_1	188	39.61	56.97	0.0018	96.68	**0.51**
He1_1	274	26.66	56.31	0.0047	83.06	**0.30**
He1_2	274	27.04	57.35	0.0019	84.43	**0.31**
He1_3	274	26.68	58.41	0.0022	85.18	**0.31**
He2_1	113	1.56	30.84	0.0049	32.43	**0.29**
He3_1	187	13.33	23.53	0.0070	36.89	**0.20**
He3_2	187	13.48	24.44	0.0018	37.92	**0.20**
He3_3	187	13.98	23.70	0.0050	37.70	**0.20**
Polo1_1	323	32.55	71.37	0.0022	103.95	**0.32**

6 Conclusion

6.1. Discussion of results

Reconsidering the time constraint posed in Section 2.2, the computation time requirement can be assessed for all extraction methods. In [36], it was concluded that frames are processed by the Feature Extraction module at $0.73s/frame$. Using the minimum value for the computation time per second of video for uniform sampling, $T_{uniformly\_sampling} = 0.032$, eq. 6.2 is obtained. For a $30fps$ video, this corresponds to eq.6.3. All methods satisfy this requirement with and without pre-sampling and shot based detection.

$$T_{computation\_keyframes} < (5 - N_{extracted}) * T_{comparison\_per\_frame} - T_{uniform\_sampling} \tag{6.1}$$

$$T_{computation\_keyframes} < 3.618 - 0.73 * N_{extracted} \tag{6.2}$$

$$T_{computation\_keyframes} < 21.9 * CR - 18.28 \tag{6.3}$$

Mandatory non-functional requirement 1 has been met since the algorithm accepts mp4 files in any (reasonable) resolution, mandatory non-functional requirement 3 and 4 have been met since the algorithm reads, extracts and stores data as efficiently as possible and gives the required RGB-data output. Mandatory non-functional requirement 2 and 5 are to be fulfilled in the following week.

Usage of HPT SBD with CFAR threshold and VSUMM in combination with crude histogram is recommended. While the modified PBT returns slightly better results, the increase in computational time makes it less preferable (Trade-off non-functional requirement 5).

The non-functional trade-off requirements could not be directly evaluated for the entire sub-module, as the analysis proved that appointing keyframes is ambiguous. These metrics could however be assessed for the shot detection part, as shot boundaries are fixated. The functional requirements are not applicable for the SBD as it only returns shot boundaries, however, it meets all the non-functional requirements as shown in the results.

There are several shortcomings that have become apparent in this report, which are mostly caused by the constraint of time or by exploring branches of research that could not be applied to this project. These are:

- The tested database of videos is small, consists of videos that are too similar in content and duration, and does not account for the general case.
- The Fidelity measure consists of 2 out of 3 descriptors that are used for its computation. The addition of wavelet descriptor d_W (which corresponds to texture) would benefit the validity of this evaluation method.

- The Shot Reduction Degree measure has not been implemented.
- The amount of consecutive frame difference keyframe extraction methods that has been explored is too small.
- For evaluation, the parameters of different methods could have been adjusted to have the same Compression Ratio in order to compare the Fidelity.

The part of the computation time that is reserved for reading frames could be improved by sampling video frames with multi-threads. Reading frames in parallel by having each worker thread read a continuous segment of a video would decrease the computation time and thus improve the general sub-module.
Furthermore, although the four videos that were subject to the application of the keyframe extraction methods are small in number, the results show that performance is very dependent on the type of video. For example, slow moving long shots would not benefit from Shot Based Detection a large amount.

6.2. Conclusion

After analyzing the keyframe extraction methods subject to this report, the most suitable method could be chosen. For the general case that the input video may vary to extremes in content and visual dynamics, VSUMM combined with histogram matching is considered to be the most suitable method. This clustering method performs well in the triangle of trade-offs: the computation time may be on the high side, but the Fidelity remains high for different types of videos. Furthermore, the implementation of detecting fade transitions also has great potential to improve the performance of the shot detection. Lastly, it is also recommended to look at the SURF algorithm, as it requires less computation time than SIFT and the literature shows great performance potential.
Further work could include optimizing the parameters of methods to give the lowest computation time of the system. When $T_{comparison\_per\_frame}$ (= 0.73), the time saved with the reduction of frames (which translates to the Compression Ratio) can be calculated and combined in calculation with the time required for the module to extract the keyframes ($T_{computation\_keyframes}$). One could optimize these parameters to give the minimum overall execution time of the entire system (taking Fidelity into account as well).

6.3. Image-Based Video Search Engine

The requirements from Chapter 2 have been fulfilled and will be discussed in this section. To reiterate the problem statement from Chapter 1:

- *Develop a system that can detect whether an instance of a desired object appears in a given video, based on a given set of images containing the desired object.*

The core functional requirements 1, 2 and 3 are met as explained in section 5.2. Requirements 1, 3 and 4 have been fulfilled and the results can be found on the GitHub [4]. Requirements 2 and 5 have both been fulfilled, because a prototype has been developed that can run on a laptop with requirements specified in the Program of Requirements as described in [36]. Requirement 6 and by extension requirement 7 have been met as described in section 5.2. Requirement 8 and by extension requirement 3 have not yet been fulfilled. The mAP will be calculated for the final prototype and included for the final version of the thesis.

The trade-off requirements specify requirements that lead to increasing customer satisfaction as they are fulfilled. Requirements 1 and 2 are met as the prototype is able to deal with multiple query videos and images. This can be seen in the GitHub [4]. Requirement 4 has not yet been fulfilled but will be fulfilled during the final weeks of the project. Requirements 5 and 6 are based on the video and image reading libraries used, which support a variety of formats. These are OpenCV [1] and Pillow (PIL) [8] respectively and the supported formats can be found in their respective documentations. Requirement 8 has been tested but is not fully met yet. The system works for videos of duration’s up to 15 minutes and support for larger video duration’s will be implemented before the thesis defence.

6.3.1. Discussion

The system meets almost all of the stated requirements, but it can not handle longer videos. This is caused by the Keyframe Extraction Module requiring large amounts of working memory, that the specified hardware rig does not possess.

6.3.2. Future Work

For future research and future BAP groups working on this project, investigating faster Keyframe Extraction and Feature Extraction implementations could significantly speed up the system. For a 6-person group working on the next generation of the project, the team could be split into three students working on the Keyframe Extraction module and three students working on the Feature Extraction module, while re-using the Data Compression and Nearest Neighbour Search implementation explained in this thesis. Additionally, this Image-Based Video Search Engine was designed for the general use case and performance can be improved by focusing on a specific use case. For such a situation, selecting a Feature Extraction network that was trained for that use case will yield even better results.

A Performance metrics tables

The complete results of some methods are given in this chapter. All computation times were calculated using the average of 10 runs.

A.1. SBD

Video		HBT		HBT + CFAR		PBT proposed		PBT modified	
Name	Total shots	Correct	Total	Correct	Total	Correct	Total	Correct	Total
anni005	38	20	47	33	67	32	310	38	195
anni009	38	19	40	31	52	36	294	38	46
nad31	187	167	238	177	249	176	830	179	215
nad33	189	181	229	189	229	181	551	181	199
nad53	83	73	120	81	112	78	239	81	94
nad57	44	41	63	44	59	42	140	44	63

Table A.1: Performance of SBD methods, both PBT methods use the CFAR threshold as standard.

Video	HBT			HBT + CFAR			PBT proposed			PBT modified		
Name	R	P	F	R	P	F	R	P	F	R	P	F
anni005	66.7	42.6	52	86.8	49.3	62.9	84.2	10.3	18.4	100	19.5	32.6
anni009	50	47.5	48.7	81.6	59.6	68.9	94.7	12.2	21.6	100	82.6	90.5
nad31	89.3	70.2	78.6	94.7	71.1	81.2	94.1	21.2	34.6	95.7	83.3	89.1
nad33	95.8	79	86.6	100	82.5	90.4	95.8	32.8	48.9	95.8	91	93.3
nad53	87	60.8	71.6	97.6	72.3	83.1	94	32.6	48.4	97.6	89	93.1
nad57	93.2	95.1	76.7	100	74.6	85.5	95.5	30	45.7	100	69.8	82.2
average	80.3	65.9	69	93.45	68.2	78.7	93.1	23.2	36.3	98.2	72.5	80.1

Table A.2: Performance of SBD methods, both PBT methods use the CFAR threshold as standard. ”R” is recall, ”P” precision and ”F” the F-measure

File Name	Video Title	HBT	HBT + CFAR	PBT proposed	PBT modified
anni005	“NASA_25th_Anniversary-Show_Segment_5”	4.41	4.874	3.993	5.265
anni009	“NASA_25th-Anniversary-Show_Segment_9”	4.401	4.411	4.18	5.656
nad31	“Spaceworks - Episode 6”	19.47	18.98	18.38	27.559
nad33	“Spaceworks - Episode 8”	17.29	17.42	21.28	27.195
nad53	“A&S_Reports_Tape_#4_-_Report_#260”	9.25	9.44	10.59	13.311
nad57	“A&S_Reports_Tape_#4_-_Report_#264”	4.26	4.51	4.72	6.327

Table A.3: Total computational time of different SBD methods in seconds

File Name	Video Title	HBT	HBT + CFAR	PBT proposed	PBT modified
anni005	“NASA_25th_Anniversary-Show_Segment_5”	0.388	0.429	0.351	0.463
anni009	“NASA_25th-Anniversary-Show_Segment_9”	0.265	0.266	0.252	0.341
nad31	“Spaceworks - Episode 6”	0.371	0.373	0.351	0.526
nad33	“Spaceworks - Episode 8”	0.347	0.35	0.428	0.546
nad53	“A&S_Reports_Tape_#4_-_Report_#260”	0.354	0.361	0.406	0.51
nad57	“A&S_Reports_Tape_#4_-_Report_#264”	0.333	0.352	0.369	0.495

Table A.4: Total computational time of different SBD methods for whole video in seconds

A.2. Crude histogram matching

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	No	3.47	0.241	13	96.96	0.932
	No	Yes	2.03	0.141	12	97.19	0.932
	Yes	No	4.20	0.291	13	96.96	0.932
	Yes	Yes	2.22	0.154	12	97.19	0.932
mailbox-street	No	No	11.7	0.195	10	99.44	0.929
	No	Yes	5.59	0.093	11	99.05	0.916
	Yes	No	15.0	0.250	17	99.05	0.916
	Yes	Yes	6.82	0.113	12	99.33	0.931
multi-shot	No	No	6.55	0.218	33	96.34	0.756
	No	Yes	2.97	0.099	32	96.45	0.715
	Yes	No	8.38	0.297	33	96.34	0.756
	Yes	Yes	3.63	0.121	32	96.45	0.715
zheng-he	No	No	6.34	0.211	37	95.89	0.837
	No	Yes	2.81	0.094	28	96.89	0.778
	Yes	No	8.62	0.287	35	96.11	0.844
	Yes	Yes	3.67	0.122	27	97.00	0.778

Table A.5: Average computational time of different SBD methods per frame in milliseconds

A.3. Histogramblockclustering

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	No	3.57	0.247	13	96.97	0.889
	No	Yes	1.60	0.111	6	98.60	0.802
	Yes	No	4.36	0.302	13	96.96	0.889
	Yes	Yes	1.78	0.123	6	98.60	0.802
mailbox-street	No	No	22.9	0.382	18	98.99	0.927
	No	Yes	6.42	0.107	16	99.11	0.931
	Yes	No	15.1	0.250	20	98.88	0.927
	Yes	Yes	6.62	0.110	16	99.11	0.931
multi-shot	No	No	7.76	0.258	32	96.45	0.750
	No	Yes	2.66	0.088	19	97.89	0.670
	Yes	No	7.79	0.259	32	96.45	0.750
	Yes	Yes	3.13	0.104	19	97.89	0.670
zheng-he	No	No	7.04	0.234	9	99.00	0.461
	No	Yes	2.32	0.077	9	99.00	0.440
	Yes	No	7.48	0.249	14	98.44	0.765
	Yes	Yes	3.01	0.100	12	98.67	0.677

A.4. VSUMM

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	No	2.81	0.195	8	98.13	0.803
	No	Yes	1.29	0.089	2	99.53	0.773
	Yes	No	3.48	0.241	8	98.13	0.803
	Yes	Yes	1.45	0.100	2	99.53	0.773
mailbox-street	No	No	11.4	0.190	35	98.05	0.971
	No	Yes	5.29	0.088	12	99.33	0.928
	Yes	No	13.8	0.230	33	98.16	0.972
	Yes	Yes	5.65	0.094	10	99.44	0.929
multi-shot	No	No	5.68	0.189	17	98.11	0.662
	No	Yes	2.19	0.073	5	99.45	0.389
	Yes	No	7.66	0.255	16	98.22	0.662
	Yes	Yes	2.91	0.097	7	99.22	0.632
zheng-he	No	No	5.97	0.199	17	98.11	0.731
	No	Yes	2.32	0.077	5	99.44	0.477
	Yes	No	8.25	0.275	15	98.33	0.680
	Yes	Yes	3.20	0.106	8	99.11	0.570

Table A.6: Performance results of VSUMM. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$ and maximum of 2% of the frames are extracted as keyframes

A.5. VSUMM and crude histogram combination

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	No	3.19	0.221	10	97.66	0.916
	No	Yes	1.79	0.124	8	98.13	0.871
	Yes	No	3.84	0.266	10	97.66	0.926
	Yes	Yes	1.95	0.135	8	98.12	0.908
mailbox-street	No	No	11.42	0.190	12	99.33	0.930
	No	Yes	5.26	0.088	11	99.39	0.932
	Yes	No	15.21	0.253	16	98.11	0.916
	Yes	Yes	6.27	0.104	12	99.33	0.932
multi-shot	No	No	6.44	0.021	29	96.78	0.705
	No	Yes	2.92	0.097	21	97.67	0.709
	Yes	No	8.15	0.271	28	96.89	0.722
	Yes	Yes	3.55	0.118	23	97.45	0.761
zheng-he	No	No	6.75	0.225	38	95.78	0.818
	No	Yes	2.96	0.099	25	97.22	0.806
	Yes	No	8.84	0.294	39	95.67	0.818
	Yes	Yes	3.83	0.127	22	97.56	0.714

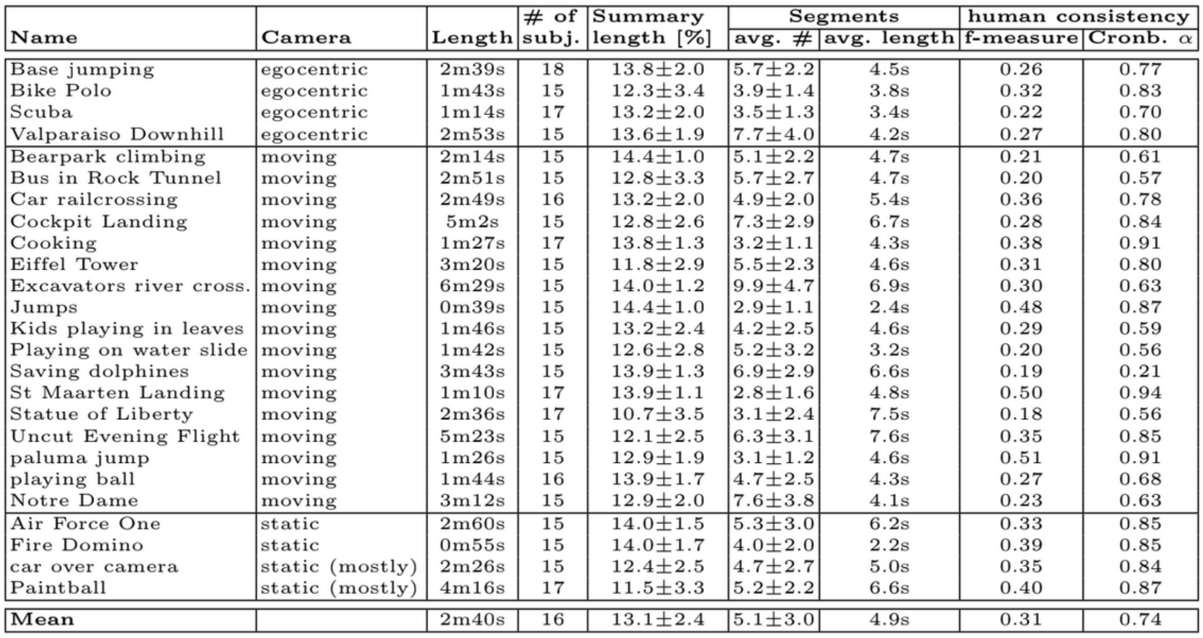
Table A.7: Performance results of VSUMM in combination with crude histogram matching. The times are calculated using the average of 10 runs (n = 10). The presampling rate, if applied, is $10fps$ and a maximum of 5% of the frames are extracted as keyframes

A.6. SIFT

Video	SBD	Presampling	Avg. Comp. time [s]	Avg. Comp. time / dur.	#KF	CR [%]	Fidelity
ewi-tudelft-2	No	No	3.41	0.237	8	98.13	0.539
	No	Yes	1.57	0.109	2	99.53	0.455
	Yes	No	3.91	0.271	8	98.13	0.539
	Yes	Yes	1.70	0.118	2	99.53	0.455
mailbox-street	No	No	13.4	0.223	35	98.04	0.903
	No	Yes	5.86	0.098	11	99.39	0.884
	Yes	No	15.8	0.263	31	98.27	0.903
	Yes	Yes	6.40	0.107	10	99.44	0.888
multi-shot	No	No	5.65	0.188	17	98.11	0.356
	No	Yes	2.25	0.075	5	99.44	0.356
	Yes	No	7.51	0.25	16	98.22	0.622
	Yes	Yes	2.83	0.094	5	99.45	0.365
zheng-he	No	No	5.59	0.186	10	98.89	0.565
	No	Yes	2.21	0.074	5	99.44	0.422
	Yes	No	7.86	0.262	12	98.67	0.647
	Yes	Yes	3.07	0.102	5	99.44	0.607

B

SumMe Benchmark



Name	Camera	Length	# of subj.	Summary length [%]	Segments		human consistency	
					avg. #	avg. length	f-measure	Cronb. α
Base jumping	egocentric	2m39s	18	13.8±2.0	5.7±2.2	4.5s	0.26	0.77
Bike Polo	egocentric	1m43s	15	12.3±3.4	3.9±1.4	3.8s	0.32	0.83
Scuba	egocentric	1m14s	17	13.2±2.0	3.5±1.3	3.4s	0.22	0.70
Valparaiso Downhill	egocentric	2m53s	15	13.6±1.9	7.7±4.0	4.2s	0.27	0.80
Bearpark climbing	moving	2m14s	15	14.4±1.0	5.1±2.2	4.7s	0.21	0.61
Bus in Rock Tunnel	moving	2m51s	15	12.8±3.3	5.7±2.7	4.7s	0.20	0.57
Car railcrossing	moving	2m49s	16	13.2±2.0	4.9±2.0	5.4s	0.36	0.78
Cockpit Landing	moving	5m2s	15	12.8±2.6	7.3±2.9	6.7s	0.28	0.84
Cooking	moving	1m27s	17	13.8±1.3	3.2±1.1	4.3s	0.38	0.91
Eiffel Tower	moving	3m20s	15	11.8±2.9	5.5±2.3	4.6s	0.31	0.80
Excavators river cross.	moving	6m29s	15	14.0±1.2	9.9±4.7	6.9s	0.30	0.63
Jumps	moving	0m39s	15	14.4±1.0	2.9±1.1	2.4s	0.48	0.87
Kids playing in leaves	moving	1m46s	15	13.2±2.4	4.2±2.5	4.6s	0.29	0.59
Playing on water slide	moving	1m42s	15	12.6±2.8	5.2±3.2	3.2s	0.20	0.56
Saving dolphines	moving	3m43s	15	13.9±1.3	6.9±2.9	6.6s	0.19	0.21
St Maarten Landing	moving	1m10s	17	13.9±1.1	2.8±1.6	4.8s	0.50	0.94
Statue of Liberty	moving	2m36s	17	10.7±3.5	3.1±2.4	7.5s	0.18	0.56
Uncut Evening Flight	moving	5m23s	15	12.1±2.5	6.3±3.1	7.6s	0.35	0.85
paluma jump	moving	1m26s	15	12.9±1.9	3.1±1.2	4.6s	0.51	0.91
playing ball	moving	1m44s	16	13.9±1.7	4.7±2.5	4.3s	0.27	0.68
Notre Dame	moving	3m12s	15	12.9±2.0	7.6±3.8	4.1s	0.23	0.63
Air Force One	static	2m60s	15	14.0±1.5	5.3±3.0	6.2s	0.33	0.85
Fire Domino	static	0m55s	15	14.0±1.7	4.0±2.0	2.2s	0.39	0.85
car over camera	static (mostly)	2m26s	15	12.4±2.5	4.7±2.7	5.0s	0.35	0.84
Paintball	static (mostly)	4m16s	17	11.5±3.3	5.2±2.2	6.6s	0.40	0.87
Mean		2m40s	16	13.1±2.4	5.1±3.0	4.9s	0.31	0.74

Figure B.1: The videos in the SumMe dataset [14].

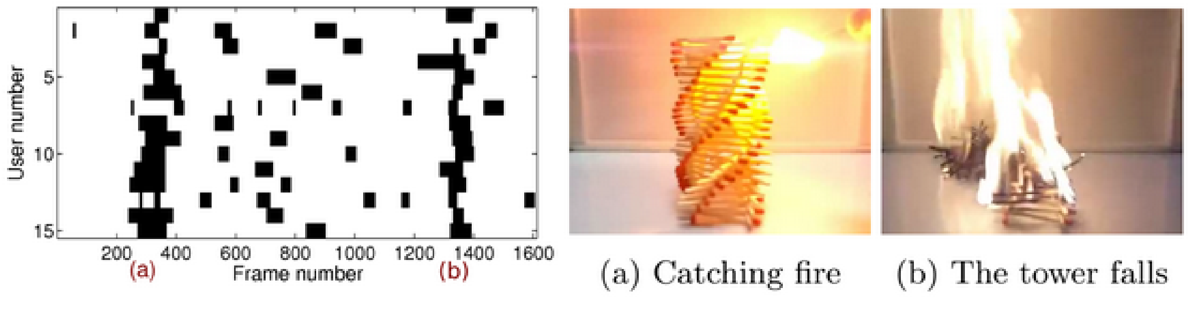


Figure B.2: Examp.le user annotation of $Fire]_{D}omino.mp4[14]$

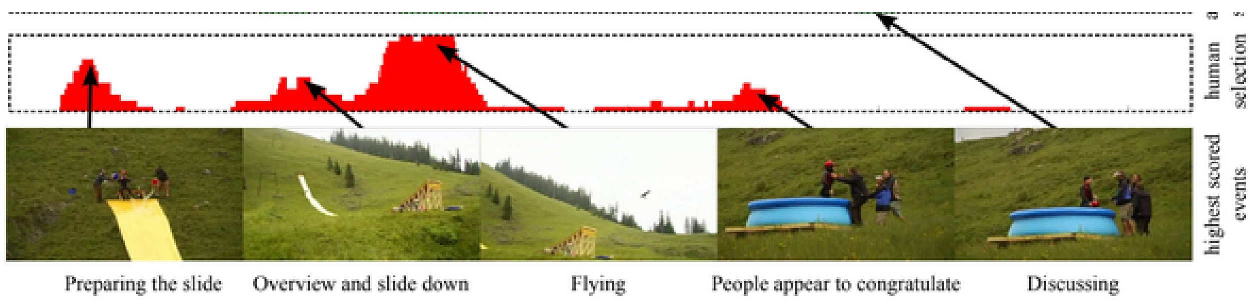


Figure B.3: Example user annotation of *Jumps.mp4*, which shows that the ”interestingness” by human interpretation is not necessary at a range where large visual changes occur [14].

Bibliography

Bibliography

[1] Opencv: Image file reading and writing. URL https://docs.opencv.org/4.x/d4/da8/group__imgcodecs.html%7D.

[2] A. Araujo, M. Makar, V Chandrasekhar, D. Chen, S. Tsai, H. Chen, R. Angst, and B. Girod. Efficient video search using image queries. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 3082–3086, 2014. doi: 10.1109/ICIP.2014.7025623.

[3] Tamires Barbieri and Rudinei Goularte. Ks-sift: A keyframe extraction method based on local features. *Proceedings - 2014 IEEE International Symposium on Multimedia, ISM 2014*, pages 13–17, 02 2015. doi: 10.1109/ISM.2014.52.

[4] Robert Bos, Leo Zheng, Aron Hoogeveen, Max van Oort, Lars Hoogland, and Matthijs Korevaar. Image-based video search engine. Available at https://github.com/aron-hoogeveen/ibvse (2022/06/13).

[5] Wei Chen, Yang Liu, Weiping Wang, Erwin M Bakker, TK Georgiou, Paul Fieguth, Li Liu, and MSK Lew. Deep image retrieval: A survey. *ArXiv*, 2021.

[6] Gianluigi Ciocca and Raimondo Schettini. Dynamic key-frame extraction for video summarization. pages 137–142, 01 2005. doi: 10.1117/12.586777.

[7] Gianluigi Ciocca and Raimondo Schettini. Erratum to: An innovative algorithm for key frame extraction in video summarization. *J. Real-Time Image Processing*, 1:69–88, 03 2006. doi: 10.1007/s11554-006-0001-1.

[8] Alex Clark. Pillow: Image file formats. URL https://pillow.readthedocs.io/en/stable/handbook/image-file-formats.html.

[9] Francesca Condorelli, Fulvio Rinaudo, Francesco Salvadore, and Stefano Tagliaventi. A neural networks approach to detecting lost heritage in historical video. *ISPRS International Journal of Geo-Information*, 9(5):297, 2020.

[10] Francesca Condorelli, Fulvio Rinaudo, Francesco Salvadore, and Stefano Tagliaventi. A neural networks approach to detecting lost heritage in historical video. *ISPRS International Journal of Geo-Information*, 9(5), 2020. ISSN 2220-9964. doi: 10.3390/ijgi9050297. URL https://www.mdpi.com/2220-9964/9/5/297.

[11] Sandra Eliza Fontes de Avila, Ana Paula Brandão Lopes, Antonio da Luz, and Arnaldo de Albuquerque Araújo. Vsumm: A mechanism designed to produce static video summaries and a novel evaluation method. *Pattern Recognition Letters*, 32(1):56–68, 2011. ISSN 0167-8655. doi: https://doi.org/10.1016/j.patrec.2010.08.004. URL https://www.sciencedirect.com/science/article/pii/S0167865510002783. Image Processing, Computer Vision and Pattern Recognition in Latin America.

[12] Max Deutman, Philip Groet, and Owen van Hooff. Image search engine for digital history, a standard approach. URL http://resolver.tudelft.nl/uuid:d0b96e9b-d383-448e-9342-db0b2560b560.

[13] A. Girgensohn and J. Boreczky. Time-constrained keyframe selection technique. In *Proceedings IEEE International Conference on Multimedia Computing and Systems*, volume 1, pages 756–761 vol.1, 1999. doi: 10.1109/MMCS.1999.779294.

[14] Michael Gygli, Helmut Grabner, Hayko Riemenschneider, and Luc Van Gool. Creating summaries from user videos. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 505–520, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10584-0.

[15] Lars Hoogland and Matthijs Korevaar. Image-based video search engine: Data compression and nearest neighbour search.

[16] H. Y. Wang J. H. Yuan and B. Zhang. A formal study of shot boundary detection. *Journal of Transactions on Circuits and Systems for Video Technology*, 17, 02 2007.

[17] Shruti Jadon and Mahmood Jasim. Video summarization using keyframe extraction and video skimming. *arXiv preprint arXiv:1910.04792*, 2019.

[18] Shruti Jadon and Mahmood Jasim. Unsupervised video summarization framework using keyframe extraction and video skimming. pages 140–145, 10 2020. doi: 10.1109/ICCCA49541.2020.9250764.

[19] Sawan Kumar. Key-frames-extraction-from-video. *GitHub*, 2020. URL https://github.com/sawankumar94/Key-Frames-Extraction-from-Video.

[20] Ioan E. Lager, Victor Scholten, Egbert Bol, Cristina Richie, and Seyedmahdi Izadkhast with special thanks to Koen Bertels. Bachelor graduation project manual, 2022.

[21] Shaoshuai Lei, Gang Xie, and Gaowei Yan. A novel key-frame extraction approach for both video summary and video index. *TheScientificWorldJournal*, 2014:695168, 03 2014. doi: 10.1155/2014/695168.

[22] Guozhu Liu and Junming Zhao. Key frame extraction from mpeg video stream. In *Proceedings of the Second Symposium International Computer Science and Computational Technology(ISCSCT)*, pages 007–011, 2009.

[23] Tie-Yan Liu, Xu Zhang, Jian Feng, and Kwok-Tung Lo. Shot reconstruction degree: A novel criterion for key frame selection. *Pattern Recognition Letters*, 25:1451–1457, 09 2004. doi: 10.1016/j.patrec.2004.05.020.

[24] Bo Luo, Xiaogang Wang, and Xiaoou Tang. World-Wide-Web-based image search engine using text and image content features. In Simone Santini and Raimondo Schettini, editors, *Internet Imaging IV*, volume 5018, pages 123 – 130. International Society for Optics and Photonics, SPIE, 2003. doi: 10.1117/12.476329. URL https://doi.org/10.1117/12.476329.

[25] J. MACQUEEN. Some methods for classification and analysis of multivariate observations. page 17, 1967.

[26] A. Nanetti. Engineering historical memory. URL http://engineeringhistoricalmemory.com.

[27] Milan Kumar Asha Paul, J. C. Kavitha, and P. Arockia Jansi Rani. Key-frame extraction techniques: A review. *Recent Patents on Computer Science*, 2018.

[28] Peter E. Hart Richard O. Duda and David G. Stork. *Pattern Classification*. Wiley-Interscience, 1973. ISBN 0471056693.

[29] Bashir Sadiq, Bilyamin Muhammad, Muhammad Abdullahi, Gabriel Onuh, Abdulhakeem Ali, and Adeogun Babatunde. Keyframe extraction techniques: A review. *ELEKTRIKA- Journal of Electrical Engineering*, 19:54–60, 01 2020.

[30] Sivic and Zisserman. Video google: a text retrieval approach to object matching in videos. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 1470–1477 vol.2, 2003. doi: 10.1109/ICCV.2003.1238663.

[31] Irwin Sobel. An isotropic 3x3 image gradient operator. *Presentation at Stanford A.I. Project 1968*, 02 2014.

[32] Newton Spolaôr, Huei Lee, Weber Takaki, Leandro Ensina, Claudio Coy, and Feng Wu. A systematic review on content-based video retrieval. *Engineering Applications of Artificial Intelligence*, 90:103557, 04 2020. doi: 10.1016/j.engappai.2020.103557.

[33] C. Sujatha and Uma Mudenagudi. A study on keyframe extraction methods for video summary. In *2011 International Conference on Computational Intelligence and Communication Networks*, pages 73–77, 2011. doi: 10.1109/CICN.2011.15.

[34] Ningli Tang, Fang Dai, and Wenyan Guo. Video summary generation based on density peaks clustering with temporal characteristics. In *2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA)*, pages 1421–1425, 2021. doi: 10.1109/ICIEA51954.2021.9516340.

[35] Mathijs van Geerenstein, Philippe van Mastrigt, and Laurens Vergroesen. Image search engine for digital history, a deep-learning approach. URL `http://resolver.tudelft.nl/uuid:f1a2902b-14be-416c-ae1a-ce4f179a0425`.

[36] Max van Oort and K.A. Hoogeveen. Image-based video search engine: Feature extraction.

[37] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009. ISBN 1441412697.

[38] Zhenyu Wu, Ruiqing Wu, Hong Yu, and Bin Tang. Key frame extraction towards kernel-sift identification. 08 2013. doi: 10.2991/icacsei.2013.40.

[39] Leo Zheng. Mimo radar for multiple moving targets and position estimation. pages 0–6, Unpublished.

[40] Yueting Zhuang, Yong Rui, T.S. Huang, and S. Mehrotra. Adaptive key frame extraction using unsupervised clustering. In *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*, volume 1, pages 866–870 vol.1, 1998. doi: 10.1109/ICIP.1998.723655.