Jake Wright

Video Summarisation

Computer Science Tripos – Part II

Queens' College

May 12, 2015

Proforma

Name:	**Jake Wright**
College:	**Queens' College**
Project Title:	**Video Summarisation**
Examination:	**Computer Science Tripos – Part II, June 2015**
Word Count:	**11,436**[1]
Project Originator:	Jake Wright
Supervisor:	Tadas Baltrušaitis

Original Aims of the Project

The aim of this project was to compare the usefulness of a video summarisation technique based on the change in colour histograms between frames with a technique based on the optical flow field throughout the video. Additionally, the project aimed to compare a summary presentation method that applied an importance threshold to the frames with a method that created a fast-forward effect through the less important sections.

Work Completed

Both video summarisation algorithms were implemented and a program was written to take an input video and produce a summary, presented using either the proposed threshold strategy or the speedup strategy. A user study was carried out to evaluate the summaries' usefulness. It was found that the optical flow algorithm outperformed the colour-based method, and videos presented with the speedup strategy were more useful than those that used the threshold strategy. The summaries were shown to always be more useful than the original videos.

Special Difficulties

None.

[1]This word count was computed by `detex diss.tex | tr -cd '0-9A-Za-z \n' | wc -w`

Declaration

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Since the advent of surveillance cameras, the rate at which video footage is collected has continued to increase. With around 5 million Closed Circuit Television (CCTV) cameras in the UK alone [22], it is becoming increasingly difficult to browse the video and retrieve clips of interest. This means that CCTV footage is not always watched, events are missed, and crimes are ignored.

As more businesses, home owners and governmental agencies turn to surveillance cameras to protect their property and prevent crime [17], it is becoming more important to find useful ways to review and archive the footage. Archiving is particularly troublesome for an average home owner who may have very limited data storage facilities. A common solution to the overwhelming storage requirements is to 'multiplex' the video streams and combine the video from multiple cameras in a single video file or on a single tape [17]. The downsides of this are that the storage requirements are only reduced by a small factor and the resolution of the video is reduced by the same factor, making identification of subjects in the video more difficult.

Another attempt to overcome the problem is to create a time-lapse video by storing only a subset of frames sampled at certain time intervals instead of recording every single video frame from the camera. This reduces storage requirements and playback time. However, identifying the actions taken by a subject in the video becomes significantly more difficult because they may only appear in a single frame or not at all.

Video Summarisation is the process of reducing spatio-temporal redundancies in video [6]. This is useful because we can algorithmically scan through long videos to automatically pick out areas of interest that might otherwise be difficult to find, thereby reducing the playback time and the storage requirements but without reducing the resolution of the video. The summarised footage is more practical in terms of storage requirements and makes retrieval of events less labour intensive.

The benefits of such a system extend beyond CCTV footage and can be useful to those working in the media industry, for example, choosing clips of long videos to show in news broadcasts or finding the most exciting clips from a video recording of a sports game. Due to the varying requirements of each use case, this project focused on CCTV footage.

1.2 Previous Work

The wide variety of applications of video summarisation has given rise to a large amount of research in the past. I will briefly explore some previously investigated methods of video summarisation before explaining the approach that was taken in this project.

Fundamentally, there are two types of video summaries [14]. First, a still-image summary can be produced wherein a set of salient images is automatically extracted from the video. This set of images can be viewed as a storyboard that summarises the events that took place in the video. Another way is to produce a new (shorter) video instead of still frames and it is the approach taken in this project. Although not as compact as a small collection of images, it is easier for non-experts to understand, and preserves the time-evolving dynamic nature of video content [16]. Note the distinction between a video summarisation technique (the method used to extract the important information from a video sequence) and a video summary (the way in which the important information is presented).

For each type of video summarisation, there must be an algorithm that can extract the most important frames or sections from a video sequence. In cases where it is desirable to find the presence of a particular object, or find sections of the video in which a known behaviour is exhibited, object detection and face recognition methods can be employed. Given a high enough resolution, faces can be found and then compared to known suspects or perhaps unauthorised vehicles can be identified entering a restricted area.

Video footage may also include auditory channels, which can be used to assist in the summarisation. Even very simple analysis of such information can increase the quality of the results. Some examples include finding the n highest audio peaks of the track and selecting sections of the video centred on the peaks or detecting a particular type of sound such as singing or applause and correlating those with important events [13]. Since this project focuses on CCTV footage, which generally doesn't include audio, these techniques cannot be exploited.

A more advanced approach would be to analyse the speech in the video using speech recognition and listen for spoken words from a dictionary of significant phrases [15]. The practicality of such an approach depends on the context. Again, CCTV footage usually does not contain this kind of information and so alternate techniques needed to be employed to summarise the video.

1.3 Project Aims

This project compares the usefulness of a video summarisation technique based on colour change with a technique that uses optical flow. The advantages of two output presentation methods are also compared. The colour-based method is based on thresholding the distance between colour histograms of each pair of consecutive frames [11]. The second method uses the optical flow field throughout the video to detect the important sections [1].

The term ‘useful’ is subjective and so a user study was used to draw the conclusion. Participants were asked to watch either an original, un-summarised video, or a summary produced by one of the algorithms, and then answer some quantitative questions. A summary is regarded as useful if participants can correctly answer the questions and do so in a shorter time than if they had only watched the original video. Chapter 4 presents the results of the user study and the conclusions that were drawn.

Chapter 2

Preparation

This chapter details the work that was undertaken before implementation of the video summarisation tool began.

Section 2.1 begins by defining the problem to be solved and from this, the requirements of the solution are derived in Section 2.2. Once the requirements have been defined, the theory upon which the project is based is explored in Section 2.3. Section 2.4 discusses video codecs and the output write strategies, followed by Section 2.5, which outlines the web platform needed for the user study. Finally, Section 2.6 discusses the software engineering principles that were applied throughout development.

2.1 The Problem

There are two problems to solve as a result of collecting an immense amount of video footage from 24-hour surveillance systems. Firstly, playback of the collected footage is too time consuming and often infeasible. Secondly, retrieval of specific events becomes inconvenient, particularly when the video contains a high proportion of redundant frames. Removing this redundancy using an automated technique is a practical way to resolve both issues. This is more broadly known as video summarisation.

The problem of video summarisation consists of two sub-problems:

1. finding important sections in the input video sequence, and
2. representing these important sections in a summary video sequence.

To find important sections in the input video sequence, an *importance detection* phase takes place. This phase identifies how important each frame is relative to every other frame. Next, a *subsequence generation* phase thresholds the frame importances and groups frames into subsequences. See Figure 2.1 for a graphical representation of this phase. Finally, an *output phase* creates the new video that summarises the input video based on the results of the preceding phases.

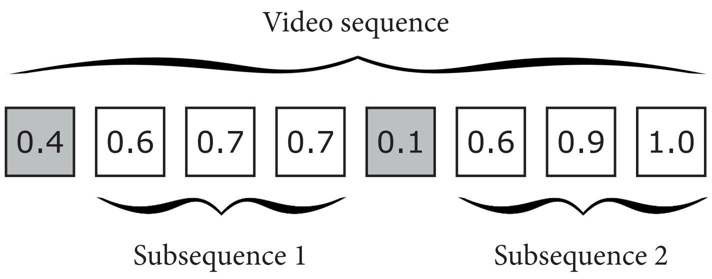


Figure 2.1: A representation of a video file made up of individual frames, each with an importance. A subsequence generation phase with a threshold of 0.5 would group the frames as shown.

Previous work has often included a phase to detect shot boundaries that are caused by camera movements and scene changes. It is assumed, however, that there is little or no camera motion in typical CCTV footage and therefore no requirement to include this phase. Many cameras have the ability to pan, tilt and zoom but this motion is slow, smooth and infrequent such that the results of the summarisation would not be significantly impacted by these effects.

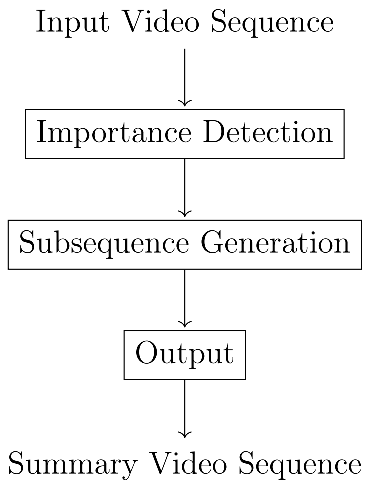


Figure 2.2: The problem of video summarisation broken down into its fundamental stages.

This basic structure, which can be seen in Figure 2.2, is later used as the foundation for the software architecture. However, before designing and implementing each phase, it is important to review what each phase should achieve. It is useful to have a formal and consistent definition of a video sequence and its components so that a framework can be unambiguously described.

Video Sequence A frame, f, is a matrix of pixels representing a still image. Let $\mathcal{F}$ denote the set of all possible frames.

A video sequence, $v \subseteq \mathcal{F}$, is a collection of these frames. Let $\mathcal{V}$ denote the set of all possible video sequences.

Define a subsequence, $s \subseteq v$, to be a set of consecutive frames in a video, v. Let $\mathcal{S}_v = \{s \mid s \subseteq v \wedge \text{consecutive}(s)\}$ denote the set of all possible subsequences in a video, v.

Importance detection phase During this phase, every frame, f, in the input video sequence, v, will be assigned a relative importance, $p \in [0, 1]$. This process will be completed by an interchangeable *detection strategy* function, d. The type of this function is described by

$$d : \mathcal{V} \rightarrow \mathcal{F} \rightarrow \mathbb{R} \tag{2.1.1}$$

In other words, given a video sequence, the detection strategy will return a mapping from frames to importances.

Subsequence generation phase Given the input video sequence, v, and a mapping from frames to importances, $m_f : \mathcal{F} \rightarrow \mathbb{R}$, as generated by the detection strategy, the sequence generation phase will output a set of disjoint subsequences that are deemed important according to some threshold, k.

The requirement for the subsequences to be disjoint is to prevent frames being duplicated in the summary video as this would create unnecessary temporal redundancy.

Output phase During this phase, an *output write strategy* is used to determine how the subsequences are written to a video file. That is, given a video, v, a set of subsequences, S, a mapping from frames to importances, m_f, and a mapping from sequences to importances, m_s, output a summary sequence, $v_s \subseteq v$, that best describes the input video sequence in as few frames as possible.

This project aims to compare the usefulness of two detection strategies and two output write strategies. Naturally, the term 'useful' is subjective in this context but this problem is discussed in much greater detail in Chapter 4.

2.2 Requirements Analysis

Not all requirements are of equal importance so those that were essential to the success of the project were prioritised. The MoSCoW method was used to help schedule development work effectively.

According to *A Guide to the Business Analysis Body of Knowledge* [3], the MoSCoW categories are defined as follows:

- **Must:** a requirement that must be satisfied in the final solution for the solution to be considered a success
- **Should:** a high-priority item that should be included in the solution if it is possible

- **Could:** a requirement which is considered desirable but not necessary
- **Won't:** a requirement that stakeholders have agreed will not be implemented in a given release

Requirements that are categorised as Must will be labelled as M1, M2, and so on.

The first requirement is derived from the purpose of video summarisation.

Requirement M1. Summary video sequences must be shorter than the input video sequence.

This could trivially be achieved by removing random frames from the video, so it is necessary to define another requirement.

Requirement M2. The summary must preserve at least 75% of the important events in a video while removing sections during which nothing happens.

This project focuses on CCTV footage for which a summary can make retrieval quicker. In order to achieve this, the summary must be an accurate representation of the events that took place in the original video sequence. To this end, the events must appear in the same order in the summary as they did in the original video otherwise confusion will arise when trying to seek to a particular event.

Requirement M3. Chronological consistency of events must be preserved in the summary sequence.

The purpose of the project is to compare two importance detection strategies and two output write strategies. This gives the next requirements:

Requirement M4. Both the colour-based and optical flow importance detection strategies should be implemented.

Requirement M5. Both the threshold output write strategy and the speedup output write strategy should be implemented.

The testing of each importance detection strategy should not be limited by substandard videos. To minimise storage requirements, it is unreasonable to expect CCTV footage to be high-definition, but to properly compare the algorithms, there should be sufficient detail in the videos.

Requirement M6. The sample videos that are collected for testing should be in colour and have a resolution of at least 640×360 pixels.

Due to the computational complexity of video summarisation, the summaries will not be produced instantaneously. However, since CCTV footage records non-stop, the summaries should be produced in real time or better, so that they can be streamed straight to disk. This prevents needing extra storage to act as a buffer.

Requirement S1. Summaries should be produced in real time or better.

A user interface for the video summarisation tool was proposed. This would create a nicer user experience when summarising videos. However, for the purpose of comparing video summarisation techniques, it was decided that this would not be a worthwhile endeavour.

Requirement W1. An easy-to-use graphical user interface for the video summarisation tool.

CCTV cameras are not always perfectly positioned so the footage may include areas in which nothing happens, for example, the top section of the video may just be the sky. It would be wasteful to spend time analysing the whole of each frame if it is known in advance that a particular area need not be considered.

Requirement C1. The ability to ignore areas of video frames during analysis.

Evaluation of the usefulness of a summary video is non-trivial due to the subjective nature of the task. To aid in evaluation, a user study should be performed. This should take the form of a survey that can be completed online. To facilitate a large-scale user study, a web platform should be produced.

Requirement S2. A web platform should enable a participant to complete a survey with a minimal amount of overhead per respondent.

To enable future work on the project, the software that is built should be extensible. It should be built such that components can be easily replaced and improved, and new algorithms implemented without restructuring the software's architecture.

Requirement S3. The software built should be extensible and have a modular design.

2.3 Theoretical Background

This section explores the theory that underlies the algorithms that were implemented.

2.3.1 Importance Detection Strategies

This project compares two detection strategies: a method that analyses colour change, and a method based on optical flow. I will describe each method in detail, which will justify why they are suitable approaches to take.

Colour Change

This algorithm is based on the colour change between frames. For CCTV footage, it is a reasonable assumption that the scene will remain static when there are no interesting events taking place. In this case, each frame will be almost identical to the preceding and succeeding frames. This can be measured by comparing the colour histograms of the frames. Similarly, if significant events are taking place in the video footage, it can be expected that the difference in colour between frames will be much greater. Figure 2.3 illustrates how a histogram changes between two frames. The distance between each pair of histograms can be used to indicate the relative importance of each frame.

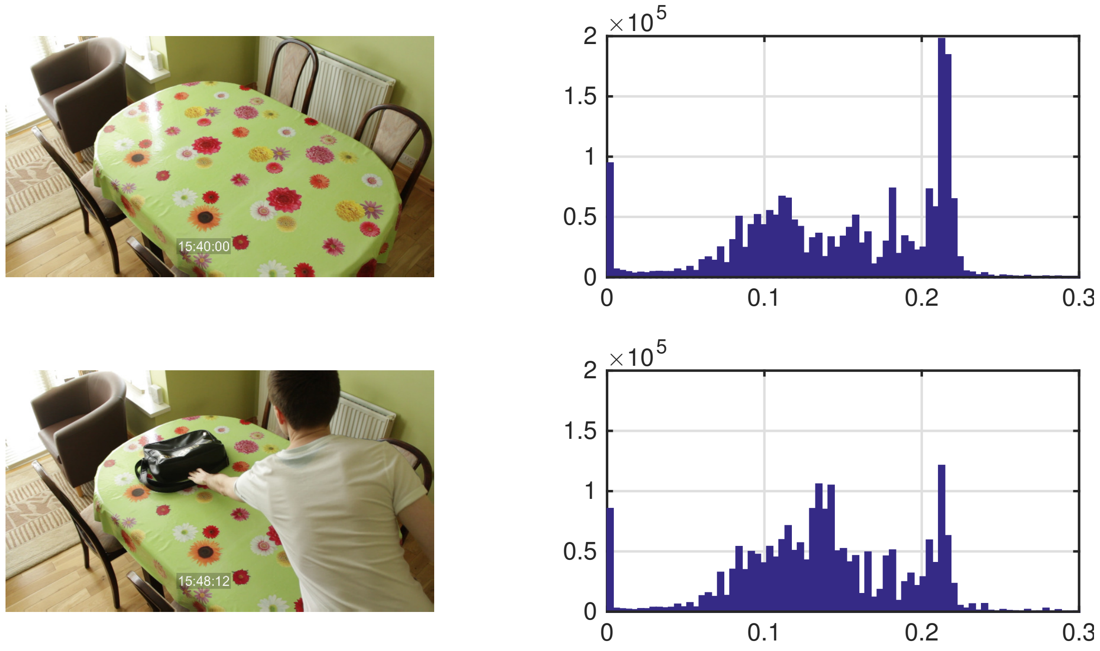


Figure 2.3: Two frames from a video and a section of their corresponding hue histograms.

Colour Histograms A colour histogram is obtained from a discrete colour space by discretising the image colours and counting how many times each discrete colour occurs in the image [24]. Discretising the images in this way as opposed to comparing pixel values between frames directly will avoid problems caused by noise in the video—pixel values will fluctuate slightly between frames even for a static scene, especially if the video was recorded in low lighting conditions. A histogram for an image is relatively invariant to rotation and translation about the normal [21] so unsteady video footage will not drastically change the histograms for the individual frames.

Converting from RGB to HSV The input frames from the video file use the RGB colour space, however, for the purpose of measuring the colour change between frames, it is useful to convert to a colour space that separates chroma (colour information) from luma (image intensity). It is only important to look at the change in colour and not changes in luma. This will make the algorithm more robust to shadows and lighting changes [5] which is useful because a lighting change will usually not signify an important event.

To this end, the first task is to convert each frame from RGB to HSV (hue-saturation-value) which is a cylindrical-coordinate representation of points in the RGB model. The HSV colour space was chosen because it fulfils the requirement of separating luma from chroma, and the conversion between RGB and HSV is easy to compute. The two properties that are most important are hue and saturation, because these contain the colour information, while value can be discarded in the analysis phase because the value is likely to indicate a change in lighting, e.g. from daytime to night time.

The RGB values can be converted to HSV as follows [21]:

$$V \leftarrow \max(R, G, B)$$

$$S \leftarrow \begin{cases} \frac{V - \min(R,G,B)}{V} & \text{if } V \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$H \leftarrow \begin{cases} 60(G-B)/V - \min(R,G,B) & \text{if } V = R \\ 120 + 60(B-R)/V - \min(R,G,B) & \text{if } V = G \\ 240 + 60(R-G)/V - \min(R,G,B) & \text{if } V = B \end{cases}$$

If $H < 0$ then $H \leftarrow H + 360$.

With a frame represented in the HSV colour space, a two-dimensional histogram can be calculated from the hue and saturation channels by discretisation of the values into a number of bins.

Histogram Comparison A method of comparing two colour histograms in the context of computer vision was first introduced by Swain and Ballard [24] and later generalised by Schiele and Crowley [20].

Three histogram comparison methods were considered and their results compared. The formula for each method is taken from the OpenCV documentation. The first method is histogram correlation, defined as

$$d_{correl}(H_1, H_2) = \frac{\sum_i (H_1(i) - \bar{H}_1)(H_2(i) - \bar{H}_2)}{\sqrt{\sum_i (H_1(i) - \bar{H}_1)^2 \sum_i (H_2(i) - \bar{H}_2)^2}}$$

where

$$\bar{H}_k = \frac{1}{N} \sum_j H_k(j)$$

and N is the total number of histogram bins. A perfect match is indicated by a score of 1 while a mismatch will give a score of -1.

Another histogram comparison formula is chi-squared [19],

$$\chi^2(H_1, H_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i)}$$

Where a value of zero indicates a perfect match and a mismatch is unbounded. This formula is derived from the χ^2 test-statistic [23].

Finally, the Bhattacharyya distance is defined as

$$d_{Bhattacharyya} = \sqrt{1 - \frac{\sum_i \sqrt{H_1(i) \cdot H_2(i)}}{\sqrt{\sum_j H_1(j) \cdot \sum_j H_2(j)}}}$$

where a perfect match gives a score of 0 and a mismatch gives a score of 1.

These methods were evaluated by comparing their performance in the sample videos. For each video,

- two consecutive frames were taken from a period during which nothing happened,
- two consecutive frames were taken from a period with noticeable activity, and
- histograms were computed for each chosen frame.

For each method, the first two histograms were compared and the second two histograms were compared. A histogram was also compared with itself to give the value for a perfect match. The three values were then normalised such that the minimum value was mapped to 0, the maximum value was mapped to 1 and the third value was scaled accordingly.

$$value = \frac{value - minimum}{maximum - minimum} \tag{2.3.1}$$

For correlation, the values were reversed

$$value = 1 - value \tag{2.3.2}$$

so all three methods could be compared.

Ideally, the frame compared with itself would have a result of 0, indicating a perfect match, the comparison between the frames in which nothing happened would be very close to 0 (noise in the video causes the histograms to differ slightly) and the frames with activity would give a result of 1, after the scaling.

The method chosen would be the one that was least susceptible to noise, indicated by having the lowest normalised value for the comparison of frames with no activity.

Some example frames used for the test are shown in Figure 2.4 and the results are summarised in Table 2.1.

χ^2 was least susceptible to noise in the tests and was chosen as the histogram comparison method to be used throughout the project. The software was built with a modular design so that this algorithm could easily be replaced by another in the future.

Comparison methods	**Same frame**	**No activity**	**Activity**
Correlation	0.0	0.0625	1.0
Chi-Square	0.0	0.0565	1.0
Bhattacharyya distance	0.0	0.342	1.0

Table 2.1: The normalised results from using each histogram comparison method to compare a frame with itself, two frames during which nothing happened, and two frames with motion.

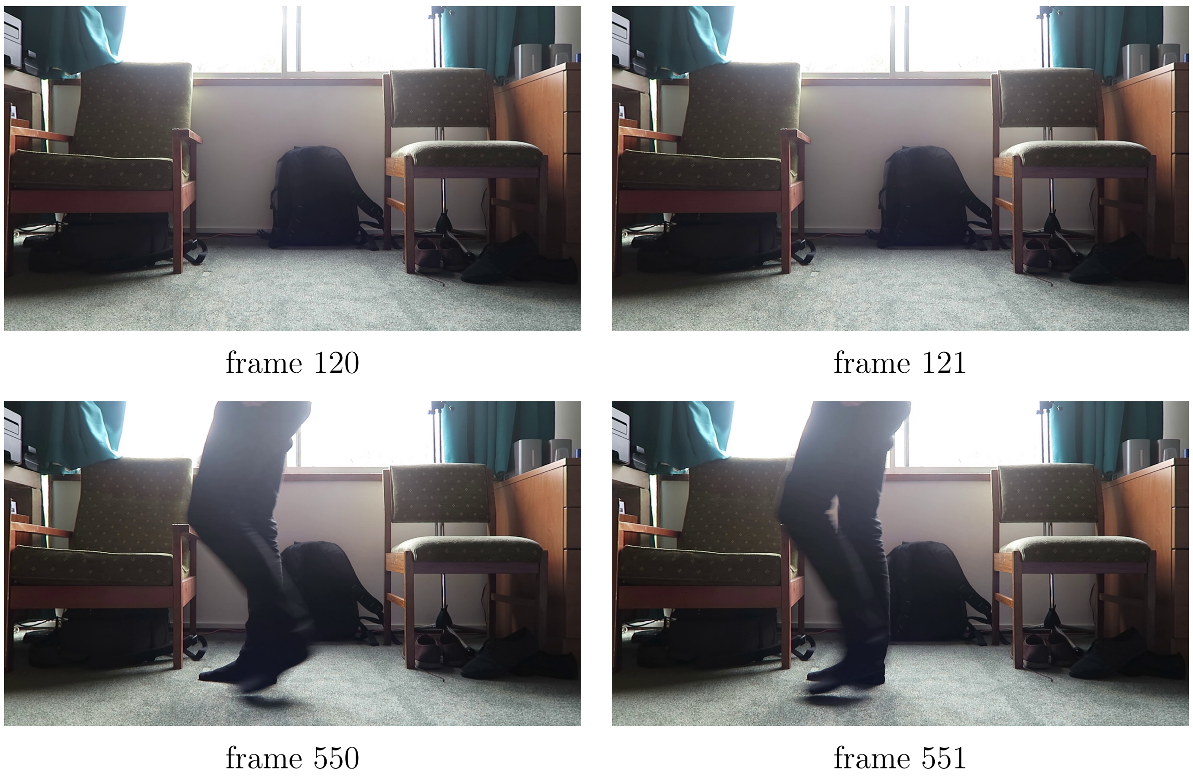


Figure 2.4: Top: a pair of consecutive frames from a period in which nothing happens. Bottom: a pair of consecutive frames from a period in which a person walks past the camera.

Optical Flow

The second video summarisation technique that will be explored is optical flow. Optical flow is the pattern of motion observed by the viewer in a scene [4]. The idea of optical flow was introduced by Gibson in 1950 [12] and a variety of algorithms have been developed since [10].

The problem of computing optical flow is the problem of estimating pixel motion between two frames. There are two key assumptions when calculating optical flow:

1. brightness constancy assumption, and
2. spatial smoothness.

The brightness constancy assumption comes from the observation that the intensity value of a surface will remain the same between frames despite a position change [2]. Spatial smoothness assumes that nearby pixels belong to the same surface and thus exhibit the same motion.

Farnebäck Algorithm Farnebäck proposed a method of estimating motion between two frames based on polynomial expansion and based on both the brightness constancy and spatial smoothness assumptions.

A neighbourhood of pixels is approximated using a quadratic polynomial

$$f(\mathbf{x}) \sim \mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{b}^T\mathbf{x} + c$$

where $\mathbf{A}$ is a symmetric matrix, $\mathbf{b}$ is a vector, and c is a scalar. The values of these coefficients are derived from a weighted least squares fit to the signal values in the pixel's neighbourhood. Solving for the translation of the polynomial between the frames provides the optical flow.

Assume the signal can be represented by the quadratic polynomial

$$f_1(\mathbf{x}) = \mathbf{x}^T\mathbf{A}_1 x + \mathbf{b}_1^T x + c_1 \tag{2.3.3}$$

If the polynomial is translated by a global displacement $\mathbf{d}$, then for non-singular $\mathbf{A}_1$, we can solve for $\mathbf{d}$. The following derivation is based on Farnebäck's paper on motion estimation [9].

Let $f_s(\mathbf{x})$ be the polynomial after a global displacement $\mathbf{d}$,

$$\begin{aligned} f_2(\mathbf{x}) &= f_1(\mathbf{x}-\mathbf{d}) \\ &= (\mathbf{x}-\mathbf{d})^T\mathbf{A}_1(\mathbf{x}-\mathbf{d}) + \mathbf{b}_1^T(\mathbf{x}-\mathbf{d}) + c_1 \\ &= \mathbf{x}^T\mathbf{A}_1\mathbf{x} + (\mathbf{b}_1 - 2\mathbf{A}_1\mathbf{d})^T\mathbf{x} + \mathbf{d}^T\mathbf{A}_1\mathbf{d} - \mathbf{b}_1^T\mathbf{d} + c_1 \\ &= \mathbf{x}^T\mathbf{A}_2\mathbf{x} + \mathbf{b}_2^T\mathbf{x} + c_2. \end{aligned} \tag{2.3.4}$$

Equating coefficients gives

$$\mathbf{A}_2 = \mathbf{A}_1, \tag{2.3.5}$$

$$\mathbf{b}_2 = \mathbf{b}_1 - 2\mathbf{A}_1\mathbf{d}, \tag{2.3.6}$$

$$c_2 = \mathbf{d}^T\mathbf{A}_1\mathbf{d} - \mathbf{b}_1^T\mathbf{d} + c_1. \tag{2.3.7}$$

By equation 2.3.6,

$$\mathbf{d} = -\frac{1}{2}\mathbf{A}_1^{-1}(\mathbf{b}_2 - \mathbf{b}_1). \tag{2.3.8}$$

In practice, the global displacement $\mathbf{d}$ is replaced with a spatially varying displacement field, $\mathbf{d}(\mathbf{x})$, and the global polynomial is replaced with local approximations. And, as previously stated, the equation is not solved pointwise, but over a neighbourhood of pixels.

2.4 Video Files

To test and evaluate the project, a variety of video files was needed. Originally, the possibility of using real CCTV footage was discussed, however, it was decided that this would not be optimal for a number of reasons.

Firstly, there are privacy concerns involved. Secondly, CCTV footage tends to be of low resolution and often is recorded in black and white. Ironically, this is due to limited storage space which is one of the problems that video summarisation can solve. Nevertheless, these constraints would limit the effectiveness of testing so it was decided that videos should be collected manually. This has the following advantages:

1. videos can be in colour so the colour-based detection strategy can be tested, and
2. scenes can be engineered from an evaluative perspective.

To elaborate on that second point, it can be ensured that videos contain the necessary features to fully exploit the algorithms' strengths and weaknesses. For example, robustness to changes in lighting can be tested by creating a video in which the brightness changes but everything else remains the same.

2.4.1 Video Coding Formats

Motion-JPEG was chosen as the video codec to be used when writing new video files because JPEG compression is computationally cheap [8] and the codec worked "out of the box" with the installation of Windows 8 and OpenCV. Of course, codecs exist that produce more compact video files (such as H.264) but I decided that it would have been unproductive to install and experiment with such codecs. Despite unwieldy file sizes partly motivating the project, efficient compression of the video files was not a requirement.

2.4.2 Output Write Strategies

While a fundamental part of this project is to compare the colour-based histogram comparison technique with the optical flow technique of video summarisation, the method used to write the results to a new file also plays a key part in producing a useful summary. Two output strategies were compared: a threshold-based strategy and a 'speedup'-based strategy.

Threshold-based Write Strategy This is the most simple method of output: given a map of subsequences to importances, output each subsequence that has an importance greater than a predefined threshold, maintaining the frames' original order from the input video file. In other words, all frames below a threshold are removed and the others are written to the output file. Essentially, this method discards frames that were deemed unimportant and only retains the significant sections, as computed by the analysis stage.

This method is easy to implement and produces the shortest summary possible by not including superfluous frames. The downside is that the optimal threshold will vary between videos. A more noticeable downside from the user's perspective is that the resulting video may lose semantic meaning and appear 'choppy'.

Speedup Write Strategy Instead of cutting all sequences that fall below a threshold, this strategy uses the frame importances to determine the playback speed of the summary. The speed will be inversely proportional to the importance so that the most importance sections are played at normal speed.

The idea behind this strategy is to maintain the context surrounding important sections of video, but 'fast-forward' through them so not much time is wasted during playback.

To achieve requirement M1, the maximum speedup should be as high as possible, making the output sequence as short as possible. If the maximum speedup is too high, however, we would lose the benefits of this strategy over the threshold write strategy. A sample video had its speed increased in steps until it became too fast to understand what was happening. A maximum speedup of about 200× was found to be optimal.

2.5 Web Platform

The user study was carried out via a web platform. Participants could use the website to watch a video, and then answer questions about its content. No particular research was undergone before building the platform, but the requirements of the user study were assessed before development began.

2.6 Software Engineering

2.6.1 Development Process

An agile software development method was adopted during development with test-driven sprint cycles to reduce risk. It was expected that new requirements and challenges would arise during implementation. A feature of the agile model is that testing is integrated into the development cycle, which allows a rapid response to change.

Typically, unit tests would be written for each new feature or module before development began. Once all of the unit tests successfully passed, the feature would be marked as complete. Afterwards, the new feature would be reviewed and integration testing would take place before starting the next sprint.

2.6.2 Software Libraries

OpenCV OpenCV is a cross-platform, open source computer vision library. It is widely used for computer vision applications and was chosen to be used in this project. It includes functionality to compute image histograms and optical flow between two frames.

Boost Boost is a set of C++ libraries that provide features such as smart pointers, optional return values, and function pointers with more features and a simpler syntax. The use of these libraries improved development efficiency and helped to prevent bugs.

2.6.3 Programming Languages

The main project was written in C++ so that it could interface with the OpenCV library. OpenCV also has Python and Java interfaces but I was unfamiliar with Python and was advised against using the Java interface due to missing implementations of some OpenCV functions.

The web platform was written using PHP, due to familiarity and an available Apache[1] server with PHP installed that was used for hosting.

2.6.4 Development Tools

Originally, I had planned to use Xcode[2] for development. The first two weeks of Michaelmas were devoted to preliminary familiarisation with OpenCV. During this period, it was noticed that reading and writing video files using the OS X implementation of OpenCV was very slow. For this reason, the OS X implementation was abandoned in favour of the Windows version.

Visual Studio[3] was chosen as the IDE due to its popularity and thus good community support, and a large feature set. Development was completed using Windows 8 in a virtual machine.

2.6.5 Version Control

Git was chosen as the version control system due to existing familiarity with the commands. A repository was hosted on Github[4], which acted as a backup solution and allowed

[1]https://www.apache.org/
[2]https://developer.apple.com/xcode/
[3]http://www.visualstudio.com/
[4]http://github.com

progress to be shared with my director of studies and project supervisor. The Team Explorer feature of Visual Studio was used to manage commits and push them to the Github service.

As a further precaution, incremental hourly backups were made of the project's source files and all related media. A second laptop was available throughout development in case of any problems with the main machine, but this was never needed.

2.7 Summary

This chapter looked at the work that was undertaken before development began.

- Section 2.1 gave an overview of the project's goals and formally described what the project should achieve
- Section 2.2 set out the requirements using the MoSCoW method by which the success of the project should be measured
- Section 2.3 looked at the underlying theory of the algorithms to be implemented, including histogram comparison and dense optical flow.
- Section 2.4 discussed video codecs to justify the decisions that were made regarding codec choice. It also looked at the output write strategies that were implemented and compared.
- Section 2.5 gave an overview of the web platform that needed to be produced for the user study.
- Finally, Section 2.6 described the software engineering techniques that were used throughout development.

Chapter 3

Implementation

This chapter describes how the theories presented in Chapter 2 were implemented in the project. The software engineering methods that were employed during development are outlined and details of the modular structure of the program are given along with the testing strategies used to ensure correct implementation of each module and of the program as a whole.

3.1 High-Level Program Structure

A basic implementation of the project's requirements only needs to run a single detection strategy on the input video's frames. This would be a very limited implementation so, in the interest of extensibility, it was decided that the detection strategies should be generalised to a *processor*. Multiple processors can be layered and run in order. Firstly, this allows detection strategies to be combined to improve results, should this be tested in the future. Secondly, it allows the subsequence generation phase to be modularised and represented as processors too. A central *processor manager* handles the processors.

The high-level structure of this concept can be seen in Figure 3.1. The arrows show the flow of control. A processor manager handles many processors, which are, in turn, given an opportunity to process each frame of the video sequence. Frames are provided to each processor through an abstraction, labelled as *Video Reader* in the figure. The output of the processor manager is passed to the *Output Manager*, which uses the data to write the new video sequence to an output file using the *Video Writer* abstraction.

At least one of the processors would be a frame importance detection strategy. This is a processor that calculates a value for each frame indicating its relative importance compared to all other frames. Consecutive frames with an importance value that exceeds a specified threshold are then defined as subsequences. This subsequence generation step takes place in a separate processor.

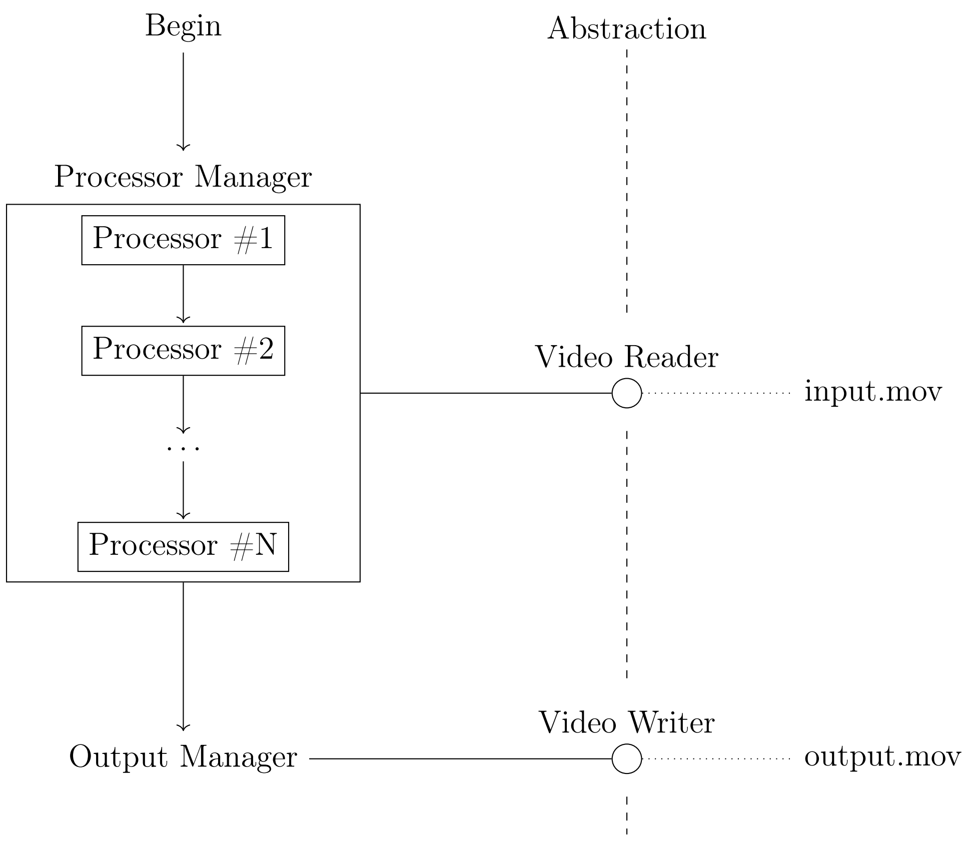


Figure 3.1: High-level structure of the video summarisation program.

Each processor is linearly connected to the next through a common interface. Designing each processor to do as few tasks as possible makes it easy to mix and combine various implementations of each step. It also makes unit testing simpler which is important because the final output of the video summarisation is hard to predict.

3.2 Unit Testing

A test-driven development process was used during implementation. Before a new feature was implemented, a test would be written that would pass if the feature was implemented successfully. This approach forces the requirements of each module to be fully realised before code is written, thereby preventing oversights from wasting time and causing bugs.

After each new feature was implemented, all unit tests were run, which would happen automatically upon building the project. The purpose of running all tests was to perform regression testing, to make sure the new code didn't introduce bugs elsewhere. If any tests failed, the source of the bug could be immediately pinpointed to the newly added or updated code. This development cycle is shown in Figure 3.2.

Integration testing took place at the end of each sprint cycle, to ensure that the program as a whole executed as expected with the addition of the new module.

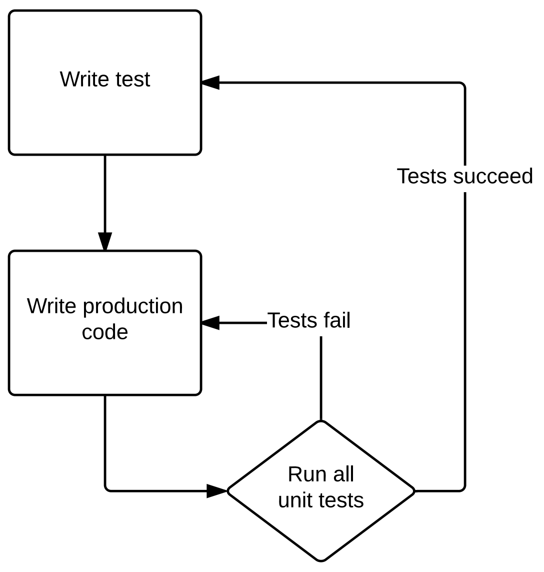


Figure 3.2: A flowchart showing the test-driven development cycle.

3.3 Input and Analysis

This section looks at the implementation details of the first part of the summarisation process. This part of the program performs the following tasks:

1. read frames from the video file,
2. apply image modifiers to the frames, and
3. run each processor in order.

Figure 3.3 is a UML class diagram showing the relationship and interfaces of the classes involved in this section of the program. The following subsections will describe the purpose of the classes in further detail.

3.3.1 Metadata

The `MetaData` class encapsulates all of the information that is generated about a video sequence during analysis. A metadata object is passed through the program with the flow of control and is updated at each stage. The main purpose of the object is to store the frame importances and the sequence ranges that are computed.

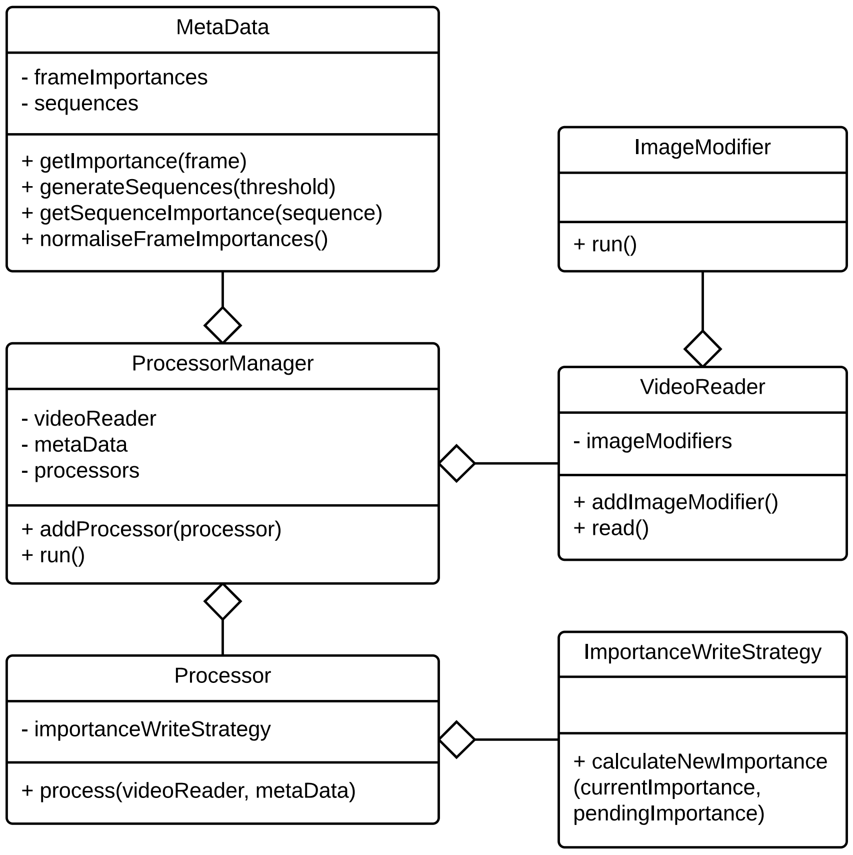

MetaData
- frameImportances
- sequences
+ getImportance(frame)
+ generateSequences(threshold)
+ getSequenceImportance(sequence)
+ normaliseFrameImportances()
ImageModifier
+ run()
ProcessorManager
- videoReader
- metaData
- processors
+ addProcessor(processor)
+ run()
VideoReader
- imageModifiers
+ addImageModifier()
+ read()
Processor
- importanceWriteStrategy
+ process(videoReader, metaData)
ImportanceWriteStrategy
+ calculateNewImportance (currentImportance, pendingImportance)

Figure 3.3: UML Class Diagram showing the aggregation between the objects that are involved in the input and analysis section of the video summarisation program.

The theory presented in Chapter 2 suggests that an object like this should maintain sets of frames. This possibility was considered but in the interest of efficiency, it was decided to represent subsequences as a pair of numbers representing the first frame of the subsequence and the final frame of the subsequence, respectively.

The mapping functions that were described throughout Chapter 2 were implemented using the C++ `map` template. The C++ map stores an ordered set[1] of key value pairs making it ideal for the frame importance map and the subsequence importance map.

As well as storing a map of subsequence ranges to importances, the metadata object has a public function to generate subsequences. This function can be called by processors to generate the subsequences that will be used in the output stage. Algorithm 3.1 shows the process in pseudocode. The actual implementation also takes into account the possibility of not having an importance for the current frame and the possibility of the subsequence already being defined, in which case, it is not overwritten because it may already have an importance itself. All new subsequences are given an importance of negative infinity.

After a processor has set an importance for each frame, the values should be normalised so that they are meaningful to a subsequent processor or the output phase. To allow code

[1]This is like the Java SortedMap rather than what one might expect from a map.

Algorithm 3.1 Subsequence Generation

```
$a \leftarrow nil$                                ▷ Used to hold the first frame in a subsequence
$b \leftarrow nil$                                ▷ Used to hold the last frame in a subsequence
for all frames $f$ do
    if $a \neq nil$                               ▷ If currently mid-subsequence then
        if $frame\ importance > threshold$ then
            $b \leftarrow f$                      ▷ Update end frame of current subsequence
        else
            Define range $<a, b>$ to be a subsequence
            $a \leftarrow nil$
            $b \leftarrow nil$
        end if
    else
        if $frame\ importance > threshold$ then
            $a \leftarrow f$                      ▷ Set start and end frame to start a new subsequence
            $b \leftarrow f$
        end if
    end if
    If $a$ and $b$ denote a valid frame range after the loop, define this as a final subsequence
end for
```

reuse, the normalisation algorithm is implemented in the metadata class. The function is called by the processor manager after each processor is run to ensure that each processor receives normalised metadata.

3.3.2 Video Reader

The underlying methods to open a video file and read frames are provided by OpenCV. The Facade design pattern was used to abstract the OpenCV methods away from the rest of the program. The facade

- makes the OpenCV library easier to use by presenting convenient functions for the required tasks,
- reduces dependencies between the program and the library by encapsulating OpenCV methods, allowing the underlying implementation to be more easily changed, and
- adds extra functionality to the video reader, i.e., the ability to add *image modifiers*.

A `VideoReader` object is instantiated with the file name of a video file. It is typically then passed to the constructor of a `ProcessorManager`. The processor manager then passes the video reader to each processor in turn, where it can be used to analyse the frames.

Image Modifiers

An image modifier is applied to each frame after it is read from the video file but before it is presented to the process requesting the frame. An image modifier can, for example, crop the image, or carry out any processing task that will later aid the analysis. The ability to crop the image partly satisfies requirement C1 by allowing parts of the image to be ignored.

The process of applying the image modifiers to the frames from the video file is shown in the UML activity diagram in figure 3.4.

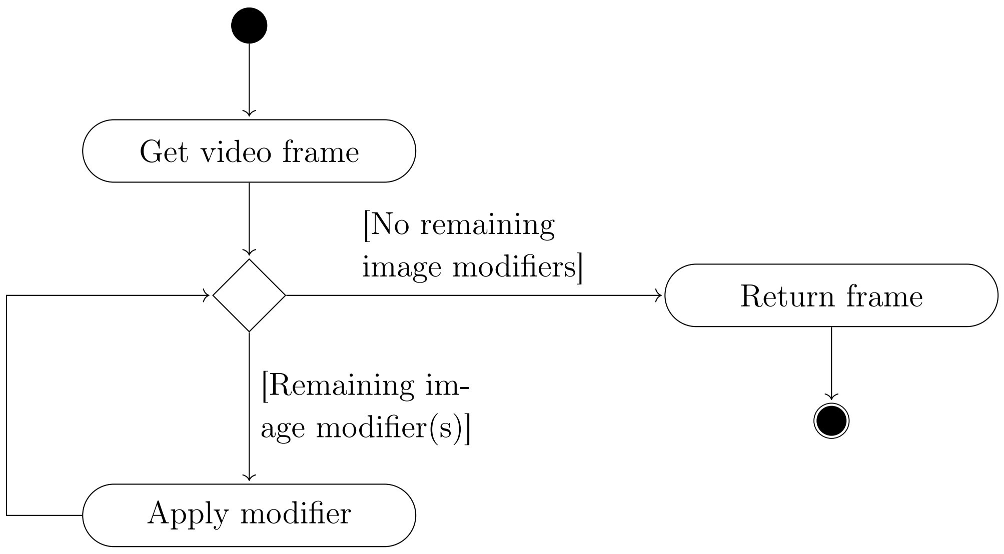


Figure 3.4: UML activity diagram showing how the Video Reader linearly applies image modifiers to the frames before returning them.

Resize image modifier For testing purposes, it was beneficial to run the the image processing at high speed so a `Resize` image modifier was implemented to reduce the resolution of video frames. OpenCV's `resize` function was used for the implementation, which can be seen in Listing 3.1.

3.3.3 Importance Detection Strategies

The two importance detection strategies described in Section 2.3 were implemented using functions provided by OpenCV. It was unnecessary to reimplement the algorithms that were already provided by the library. Each detection strategy was encapsulated in a `DetectionStrategy` object with a common interface. Figure 3.5 shows how each detection strategy inherits from a common base class. The process method takes a reference to a video reader, a metadata object and an importance write strategy, which defines how frame importances in the metadata object should be overwritten. This is discussed in more detail in Section 3.3.4.

Listing 3.1: Resize image modifier implementation

```
void Resize::run(Frame *frame)
{
  // Extract the image matrix from the Frame object
  cv::Mat mat = frame->getFrame();

  // Create a matrix to hold the resized version
  cv::Mat matResized;

  // Use OpenCV's resize function with the member variable factor_
  cv::resize(mat, matResized, cv::Size(), factor_, factor_);

  // Set the Frame's inner image matrix to the resized version
  frame->setFrame(matResized);
}
```

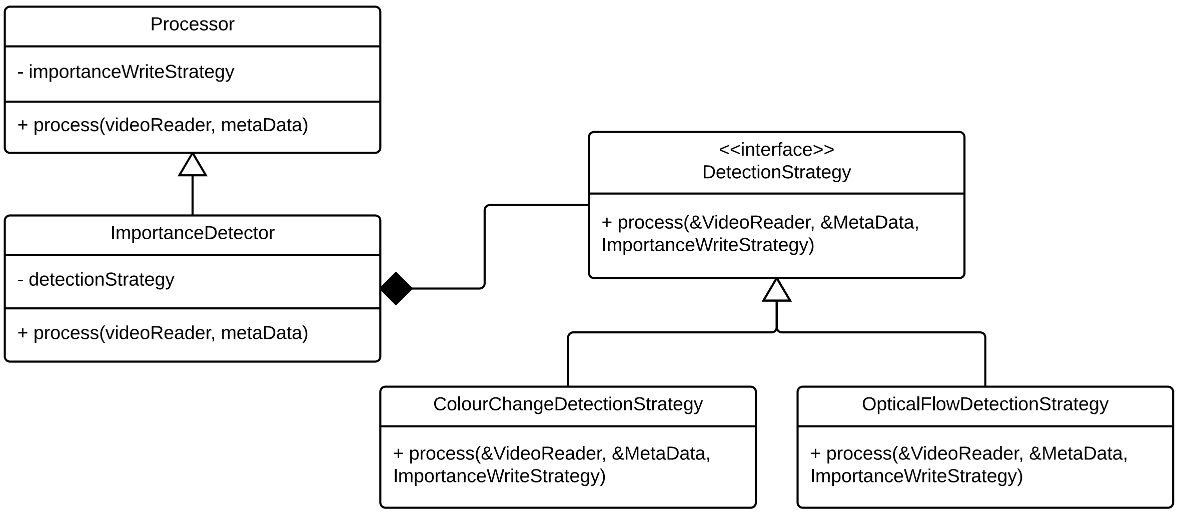

Processor
- importanceWriteStrategy
+ process(videoReader, metaData)
ImportanceDetector
- detectionStrategy
+ process(videoReader, metaData)
<<interface>>
DetectionStrategy
+ process(&VideoReader, &MetaData, ImportanceWriteStrategy)
ColourChangeDetectionStrategy
+ process(&VideoReader, &MetaData, ImportanceWriteStrategy)
OpticalFlowDetectionStrategy
+ process(&VideoReader, &MetaData, ImportanceWriteStrategy)

Figure 3.5: UML class diagram showing how the Strategy pattern was used to allow summarisation algorithms to be interchangeable.

Colour-Change Detection Strategy

This strategy calculates histograms for each frame and compares the difference between each consecutive pair of histograms. The larger the difference, the larger the importance that is assigned to the frames.

Firstly, to convert from the RGB colour space to HSV, the `cv::cvtColor` function was used.

```
  // Convert to HSV
  cv::cvtColor(frame, hsv, cv::COLOR_BGR2HSV);
```

where `frame` is a matrix representing the current frame in RGB, `hsv` is a destination matrix and `COLOR_BGR2HSV` is the colour space conversion code.

Next, the histograms are created from the new matrix

```
// Calculate the histograms for the HSV images
cv::calcHist(&hsv, nImages, channels, cv::Mat(), hist,
dims, histSize, ranges, uniform, accumulate);
```

where

- `hsv` is the image matrix
- `nImages` $= 1$ is the number of source images
- `channels` $= \{0, 1\}$ is the set of channels to build the histogram from (the 0th and 1st channels being hue and saturation)
- `cv::Mat()` represents an empty mask
- `dim` $= 2$ is the number of dimensions in the histogram
- `histSize` $= \{50, 60\}$ is a set containing the number of hue bins and saturation bins, respectively
- `ranges` $= \{\{0, 180\}, \{0, 256\}\}$ is a set of ranges for each dimension
- `uniform` $=$ `true` is a flag to evenly space the bins, and
- `accumulate` $=$ `false` makes sure the histogram is reset each time.

The histogram is normalised

```
// Normalise the histogram
cv::normalize(hist, hist, alpha, beta, cv::NORM_MINMAX, dType, cv::Mat());
```

where `alpha` $= 0$ and `beta` $= 1$ specify the range to which the histogram is normalised. `dType` $= -1$ specifies that the output array should have the same type as the input array, which, in this case, are the same array.

Finally, two histograms are compared—the current histogram and the previous frame's histogram.

```
// Compare the current histogram with the previous histogram
double comparison = cv::compareHist(prevHist, hist, compareMethod);
```

`compareMethod` can be any of four histogram comparison methods implemented in OpenCV. The χ^2 comparison was used as described in Chapter 2.

This value is used as the frame importance for the latter frame of the two that were compared. The first frame in the sequence gets an importance of zero by convention and the importances are normalised to the range $[0, 1]$.

Optical Flow Detection Strategy

This strategy calculates the dense optical flow for each pair of consecutive frames and sums up the absolute values of each vector in the resulting motion matrix. This sum is used to indicate the frame importance.

Firstly, the frames are converted to greyscale using the `cvtColor` function.

```
cv::cvtColor(frame, gray, cv::COLOR_BGR2GRAY);
```

The transformation is carried out using the formula[2]:

$$\text{RGB[A] to grey:} \quad Y \leftarrow 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \tag{3.3.1}$$

Next, the `calcOpticalFlowFarneback` function is used to produce a matrix of flow vectors.

```
// Calculate optical flow using Farneback algorithm
cv::calcOpticalFlowFarneback(prevGray, gray, flow, pyrScale,
levels, winSize, iterations, polyN, polySigma, flags);
```

The following list gives the definitions of each argument according to the OpenCV documentation[3] and the values that were used.

- `prevGray` is the first frame, and
- `gray` is the second frame to calculate optical flow between
- `flow` is the destination matrix for the optical flow calculations
- `pyrScale` = 0.5 builds a classical pyramid where each layer is twice as small as the previous one
- `levels` = 3 is the number of pyramid layers including the initial image
- `winSize` = 15 is the averaging window size
- `iterations` = 3 is the number of iterations performed at each level of the pyramid
- `polyN` = 5 is the size of the pixel neighbourhood used for the polynomial expansion
- `polySigma` = 1.2 is the standard deviation of the Gaussian used to smooth derivatives, and
- `flags` = 0 tells the algorithm not to use any additional options.

The OpenCV function returns a matrix of flow vectors, which is then turned into a double precision floating point number as shown in the following code listing.

[2]http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html
[3]http://docs.opencv.org/modules/video/doc/motion_analysis_and_object_tracking.html

Detection Strategy	Input	Expected output
Colour	Two solid black frames	Importance of zero assigned to both
Colour	A black frame followed by a white frame	Importance of zero assigned to both
Colour	Two frames with maximum change in hue and saturation HSV$(0, 0, 0)$ $\rightarrow$ HSV$(359°, 100\%, 100\%)$	An importance of 0 for the first frame and a maximal importance of 1 for the second frame
Optical flow	250 identical frames followed by 250 frames with a chequerboard pattern with alternating black and white squares, simulating motion.	An importance of 0 for the first 250 frames and a normalised importance of 1 for the second 250 frames

Table 3.1: The unit tests for the importance detection strategies.

```
static double calcAmountOfFlow(const cv::Mat& flow)
{
  double totalFlow = 0;
  for (cv::MatConstIterator_<cv::Point2f> it = flow.begin<cv::Point2f>();
  it != flow.end<cv::Point2f>(); it++)
  {
    // Compute the size of the vector and add it to the total flow
    totalFlow += sqrt(pow((*it).x, 2) + pow((*it).y, 2));
  }

  totalFlow = totalFlow / (flow.rows * flow.cols);
  return totalFlow;
}
```

Detection Strategy Testing

Chapter 4 details the extensive evaluation that took place in order to test the performance of each detection strategy. Nevertheless, during development it was useful to know that the algorithms were performing correctly in the extreme cases. For this reason, a test video reader object was written, which, instead of outputting frames from a real video file, would output synthesised frames with particular properties. The unit tests for the detection strategies are shown in Table 3.1.

3.3.4 Processors

A processor can be an importance detection strategy, perform a subsequence generation step, or transform the metadata object in another way. Each processor has access to a shared metadata object. This facilitates processors that modify the working range of the sequence, merge overlapping sequences, and further refine the summarisation that has taken place thus far. The implemented processors are:

- Trim Working Range
- Initialise Frame Importances
- Importance Detector
- Average Sequence Importance
- Remove Short Sequences
- Pad Sequences to Minimum Length
- Pad Sequences By Fixed Amount
- Merge Overlapping Sequences

Trim working range It may not be desirable to process the entire video sequence. This processor was implemented to allow only a subsection of the video to be analysed by adjusting the working range that is stored in the metadata object.

Initialise frame importances In general, this would be run before any frame importances are set, to initialise them all to zero. It is sometimes useful to initialise them to another value, perhaps a [normalised] value of 1 so that frames are included by default.

Importance detector This is the encapsulating processor for the main analysis algorithms. The Strategy design pattern was employed to allow the detection algorithm to be interchangeable. The importance detector processor takes a `DetectionStrategy` object as an argument.

Average sequence importance The metadata object can generate sequences from the frame importances and a given threshold. This creates the task of generating sequence importances. Using the processor-based structure, any number of approaches can be implemented. The one that was chosen simply averages the importances of the frames within a sequence.

$$\forall s \in S . m_s(s) = \frac{1}{|s|} \sum_{f \in s} m_f(f) \tag{3.3.2}$$

Remove short sequences It was found during early testing that the detection strategies often produced anomalous frame importances—a single important frame surrounded by frames of low importance. It was sensible to implement a processor that could discard the short sequences that resulted from this.

Pad sequences to minimum length ‘Choppy’ video was expected from the threshold write strategy since the video would jump with no transition between events. The speedup write strategy was expected to diminish this effect however to achieve the desired reduction in temporal redundancy, it was still necessary to totally disregard the subsequences of lowest importance. This meant that under both write strategies, the resulting summary sequence was often confusing to watch.

To alleviate this further, sequences were padded to conform to a minimum length (10-30 frames usually).

Algorithm 3.2 Pad sequences to minimum length

```
for all sequences $s$ do
    if length($s$) $<$ $threshold$ then
        $padLower \leftarrow$ TRUE
        while length($s$) $<$ $threshold$ do
            if $s.firstFrame = 0$ and $s.lastFrame = N$ then
                Break
            else if $s.firstFrame = 0$ then
                $padLower \leftarrow$ FALSE
            else if $s.lastFrame = N$ then
                $padLower \leftarrow$ TRUE
            end if
            if $padLower$ then
                $s.firstFrame \leftarrow s.firstFrame - 1$
            else
                $s.lastFrame \leftarrow s.lastFrame + 1$
            end if
        end while
    end if
end for
```

Pad sequences by fixed amount This processor was designed to retain semantic meaning in the summary videos by including more context surrounding each ‘important’ event.

Merge overlapping sequences The previous two processors that extend the sequences beyond what is defined by the frame importances causes the problem of overlapping sequences. If sequences overlap, the frames end up being output multiple times, increasing redundancy and confusing the chronological consistency of the events. It was therefore necessary to implement a processor to merge overlapping sequences.

The approach taken was to iterate through each frame of each subsequence and, in a new metadata object, set the importance of the corresponding frame to the importance of the subsequence. If the frame appeared in more than one subsequence, the new importance was the average of all subsequence importances. New subsequences could then be generated in the standard way on the new metadata object. This process is represented in Figure 3.6.

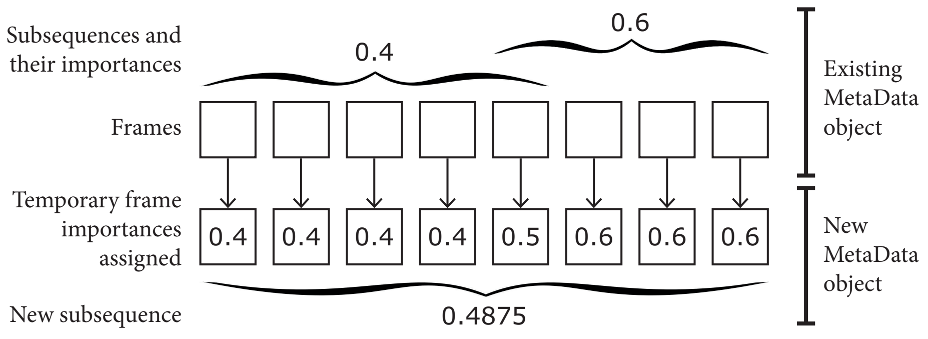


Figure 3.6: The approach taken to merge overlapping subsequences. Frame importances are set in a new metadata object as the average of the subsequence importances in which the frame appeared. New subsequences are then generated from the new metadata in the usual way.

Overwriting Frame Importances

The importance detector processor is intended to overwrite frame importances. However, the program's structure was created such that processors can be layered, including detection strategies. For example, it is possible to run an optical flow detection strategy and then run a colour-based detection strategy and combine the results to get a better summary. Therefore, it is necessary to allow a processor to do more than simply overwrite an existing frame importance.

Detection strategies implement a `process` method that takes an `ImportanceWriteStrategy` object as an argument (along with a video reader and metadata object). This importance write strategy defines how a new importance should be combined with an existing importance. The most simple strategy is to overwrite the importance while the more useful strategy is to average the importances. Of course, any number of strategies can be implemented to combine them, for example, multiplicatively or additively.

This introduced the problem of combining a normalised frame importance with an unnormalised importance. To solve this, it was necessary to implement a second frame importance map in the `MetaData` class called `pendingFrameImportance`. A processor uses it as a temporary workspace, then calls the `normalise()` method on it before using the importance write strategy to update the actual frame importance map.

3.3.5 Processor Manager

The processor manager stores a vector of processors and runs them all sequentially, giving them access to the shared metadata object. After each processor is executed, the processor manager normalises the frame importances in the metadata object by calling `MetaData::normaliseFrameImportances()`. Afterwards, it returns the resulting metadata to the caller.

3.4 Output

This part of the program takes the metadata from the previous step and uses an output write strategy to produce a summary video sequence.

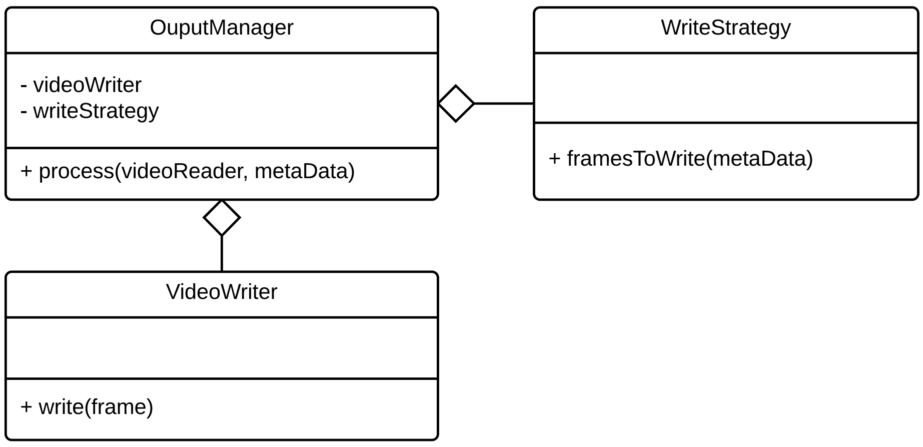


Figure 3.7: UML Class Diagram showing the aggregation between the objects involved in the output stage of the video summarisation tool. Note that only the relevant properties and methods are displayed.

3.4.1 Output Write Strategies

The two output write strategies described in Section 2.4 were implemented. Each write strategy extends a common abstract class, implementing a `framesToWrite` method. This method takes the metadata and returns the set of frames that should be written to the output video file.

The threshold write strategy has a simple implementation that returns the set of frames that are part of a subsequence and have an importance greater than a given threshold.

The speedup write strategy was implemented using a variable counter to only write every nth frame. This variable was computed from the importance of the current frame

$$counter = m_f(f) \times (1 - maxSpeedup) + maxSpeedup$$

where $m_f(f) \in [0, 1]$ is the normalised importance of the current frame. Testing found the output of such a naive implementation to be unsatisfactory. To produce a smoother speed ramp, the code was changed such that the counter was averaged over 50 frames' importances. Furthermore, a threshold was introduced so that more frames were slowed down to the normal speed to make the important events easier to follow. The effect of this threshold on the speed of the video can be seen more clearly in Figure 3.8. Finally, the variable was rounded to an integer. The resulting algorithm is as follows.

Algorithm 3.3 Speedup Output Write Strategy

```
$i \leftarrow 0$                                  ▷ A counter incremented on each iteration
$k \leftarrow 50$                                 ▷ The number of frames to average the speed over
$X \leftarrow \varnothing$                        ▷ The set of frames to write to the output file
for all subsequences $s \in S$ do
    for all frames $f \in s$ do
        if $i$ mod round($counter$) $= 0$ then
            $X \leftarrow X \cup \{f\}$
        end if
        $newCounter \leftarrow \frac{m_f(f)(1-maxSpeedup)}{threshold} + maxSpeedup$
        $counter \leftarrow counter + \frac{newCounter - counter}{k}$
        $i \leftarrow i + 1$
    end for
end for
```

3.4.2 Video Writer

As with the video reader, the video writing methods of OpenCV are encapsulated and abstracted away from the rest of the program.

The video reading and writing functions work with `Mat` objects, which represent matrices. Due to this being specific to OpenCV, this was also encapsulated within a custom `Frame` class.

3.4.3 Output Manager

The output manager administers the whole output process by completing the tasks:

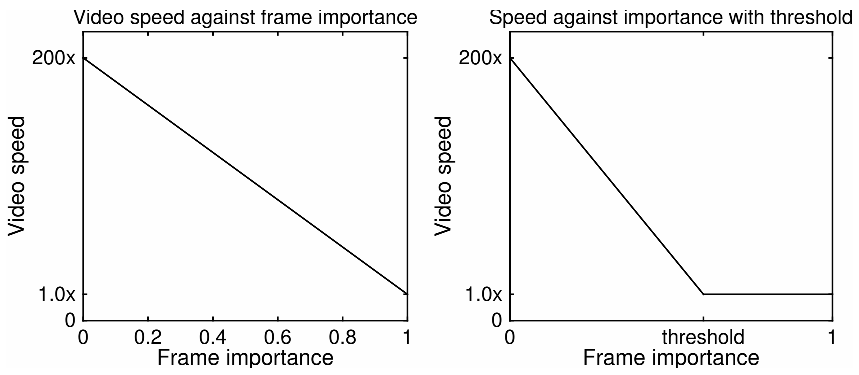


Figure 3.8: A comparison between the speed of a video with and without a threshold on the importance. The threshold was advantageous because it is unlikely that many frames would be assigned the maximal importance of 1, while it is desirable to have sections of the video, during which an event occurs, slowed down to normal speed.

1. run the output write strategy to get a set of frames to output,
2. read frames using the video reader, and
3. output frame if necessary using the video writer.

The output manager was implemented as a separate entity to the processor manager so that it can be run using existing metadata, without having to run the processors again.

3.5 Web Platform

The user study web platform was implemented as a PHP project with a MySQL database. The database was the first part of the system to be designed. The relational schema, shown in Figure 3.9, is in Boyce-Codd normal form [7]. Normalisation created an extensible, update-centric design with no data duplication.

The website was built using Bootstrap[4] as a front-end framework. The user interface can be seen in Figure 3.10. Users' inputs were sanitised to protect the system from MySQL injection attacks and a CAPTCHA prevented unwanted form submissions by bots.

Once the user study was over, another PHP script was executed to aggregate and output the results that were required for the evaluation. Part of this script is given in Appendix A.

[4]http://getbootstrap.com

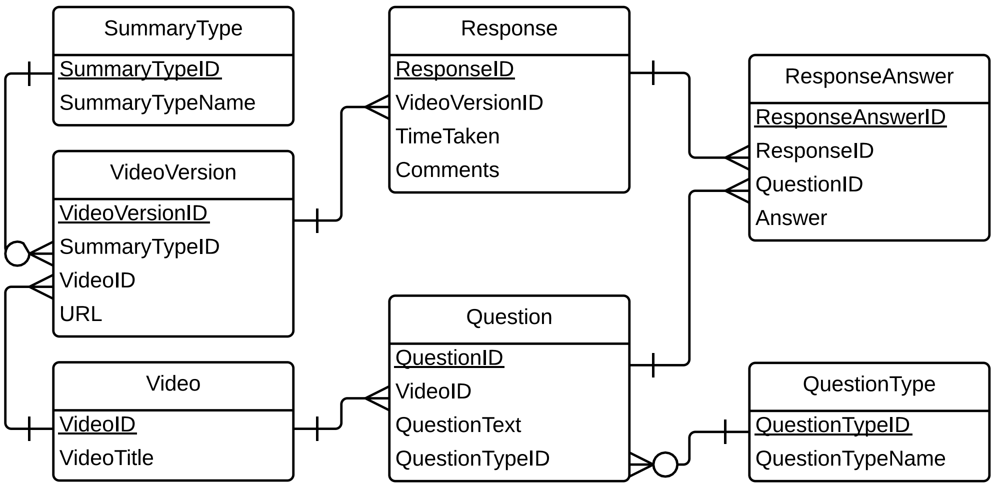
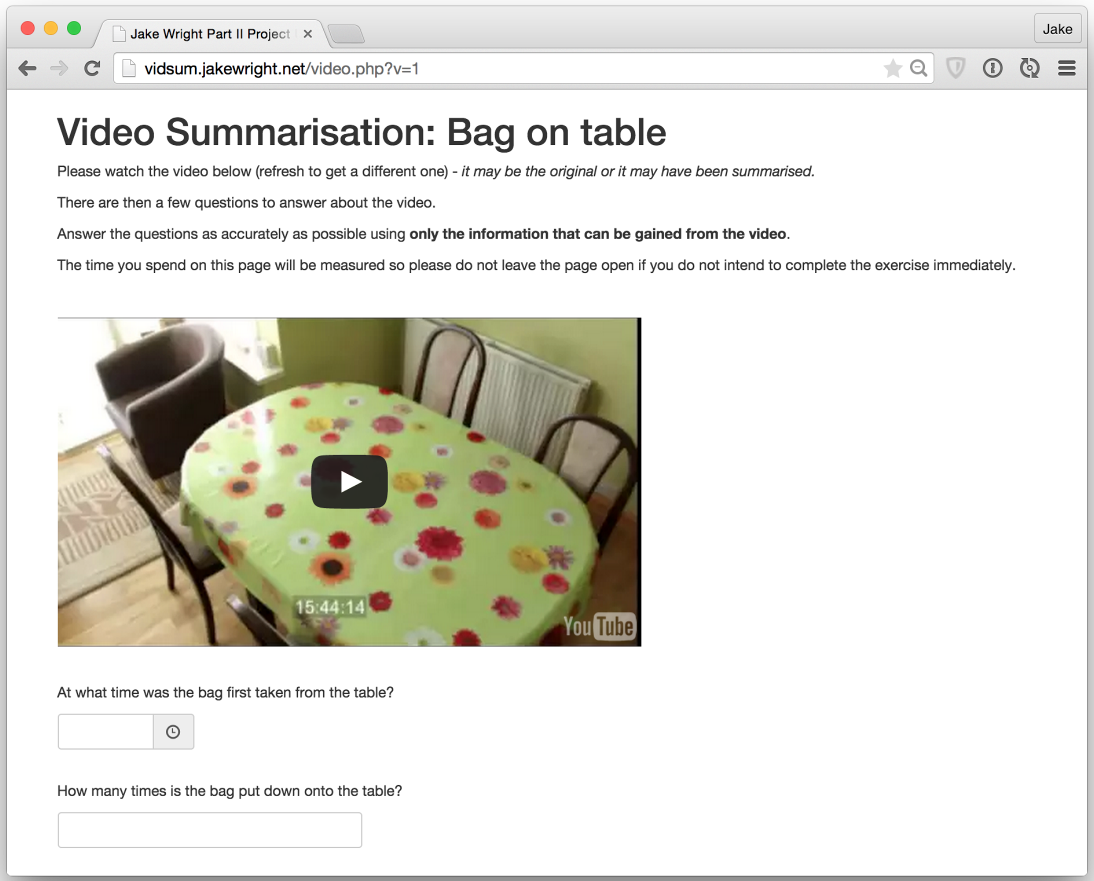


Figure 3.9: Entity-relationship diagram showing the database structure used for the back-end of the user study web-based platform.


Jake Wright Part II Project
Jake
vidsum.jakewright.net/video.php?v=1
15:44:14
YouTube

Video Summarisation: Bag on table

Please watch the video below (refresh to get a different one) - *it may be the original or it may have been summarised.*

There are then a few questions to answer about the video.

Answer the questions as accurately as possible using **only the information that can be gained from the video**.

The time you spend on this page will be measured so please do not leave the page open if you do not intend to complete the exercise immediately.

At what time was the bag first taken from the table?

How many times is the bag put down onto the table?

Figure 3.10: User interface of the web platform developed for the user study.

3.6 Summary

This section detailed the implementation of the program that performed the video summarisation and the web platform on which the user study was conducted.

- Section 3.1 gave a high-level overview of the structure of the program that was built.
- Section 3.2 described how continuous testing was integrated into the development workflow including unit testing, integration testing and regression testing.
- The input and analysis parts of the video summarisation program were described in Section 3.3. UML Class diagrams showed the relationship between the modules. A UML activity diagram showed the process of applying image modifiers to the frames and then each processor was explained.
- The part of the program that handles output was detailed in Section 3.4.
- Finally, details about the web platform, through which the user study was conducted, were given in Section 3.5. Details include input sanitisation and an entity relationship diagram showing the normalised database.

Chapter 4

Evaluation

The purpose of the project was to compare the usefulness of the colour-based video summarisation algorithm with the optical flow algorithm. A secondary goal was to explore the benefits of the speedup output write strategy over a simpler threshold-based method. To strengthen the conclusions that were made, the project was evaluated in two ways:

- each summary was compared to a manual summary of the same video, and
- a user study was carried out to evaluate the usefulness of the summaries.

The manual evaluation was used to ensure that the salient frames in the input video were being captured by the summarisation algorithms. However, this was very limited in terms of assessing the usefulness of the summaries, due to the subjective nature of the question. A user study was carried out to test the videos in scenarios that were designed to simulate real life situations involving CCTV footage.

The evaluation was broken down into the following three questions:

1. Is the speedup output write strategy better than the threshold write strategy?
2. Is the optical flow detection algorithm better than the colour-based method?
3. Are the summaries that were produced more useful than the original, un-summarised videos?

Recall from Chapter 1 that in the context of summarised videos, a useful summary was defined to be one that enabled a user to answer questions about a video correctly and in a shorter time than if the original video had been viewed instead.

The speedup write strategy was expected to perform better in the user study because more context and meaning would be preserved in the output video. This would lead to less confusion about which events were taking place thereby making it easer to answer the questions. It was expected that under the threshold method, despite the videos being shorter, participants would spend more time replaying regions of the video to understand the story.

There was no initial evidence to suggest that the optical flow detection strategy would outperform the colour-based strategy. This is what the project set out to determine. However, throughout development, testing found that the optical flow method was generally better at capturing the salient frames without picking up noise.

4.1 Videos Used in Evaluation

Three videos from the test corpus were chosen to be used for both the manual evaluation and the user study.

1. The bag video: a bag being placed on and removed from a table multiple times.
2. The desk video: a video of a desk being tidied, featuring significant lighting changes during the video.
3. The river video: a video of the River Cam with punts and pedestrians passing individually and occasionally two at a time.

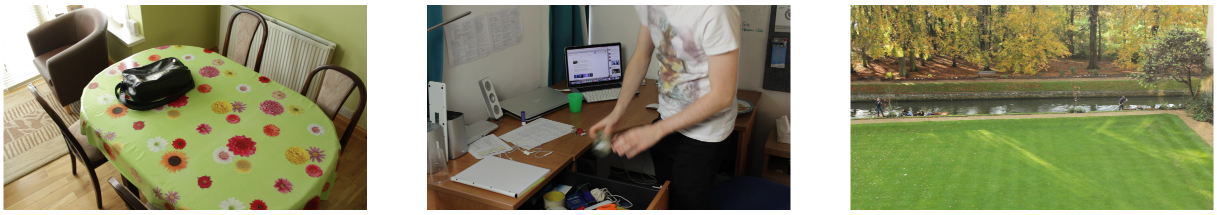


Figure 4.1: From left to right, representative frames from the bag video, the desk video, and the river video.

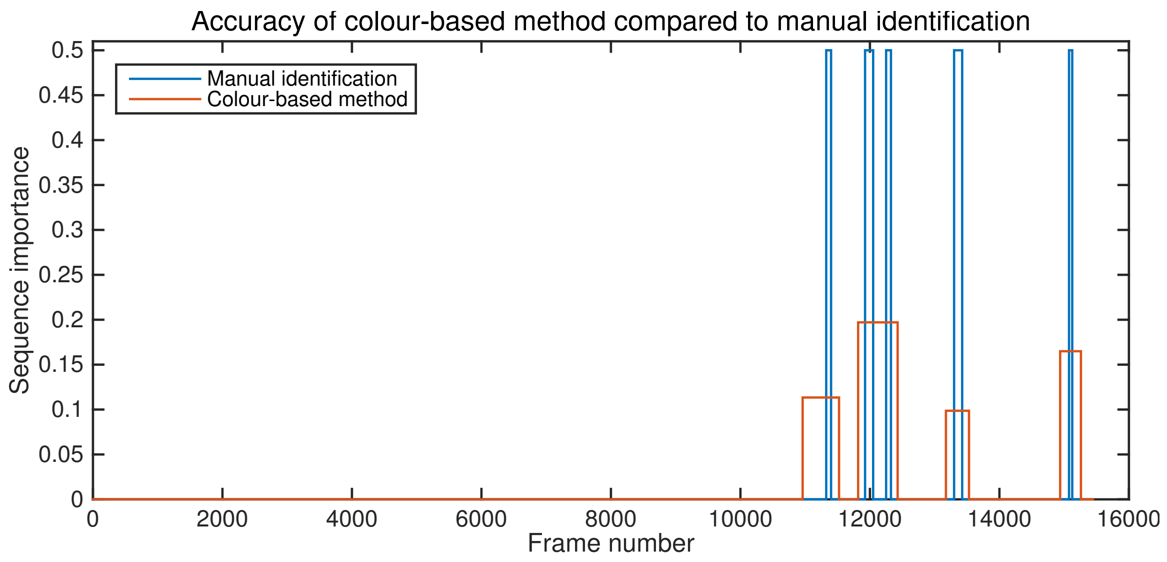
The bag video has very little background noise and very well defined events. The desk video was chosen to be more difficult to analyse, due to lighting changes and sections with low light. The river video was selected because of its large amount of noise from moving trees surrounding the river, which would be more challenging for the algorithms.

4.2 Manual Evaluation

Each of the videos from the previous section had their important sections manually identified. Each video was then analysed by the colour-based algorithm and the optical flow algorithm. Ideally, the important sections as identified by the algorithms would correlate with the standard that was defined manually.

The results can be seen in Figures 4.2, 4.3, and 4.4. The important sections according to the algorithms are overlayed on the manual identification. The manual study showed that the algorithms were performing adequately with the test videos.

The first video was the easiest for the algorithms to analyse. It featured large amounts of activity in the ‘important’ sections which contrasted with still footage everywhere else. The subsequence importance is plotted against the frame number to show how the algorithm output compares with the manual identification. Both algorithms performed very well with this video, correctly identifying each event without any false positives.



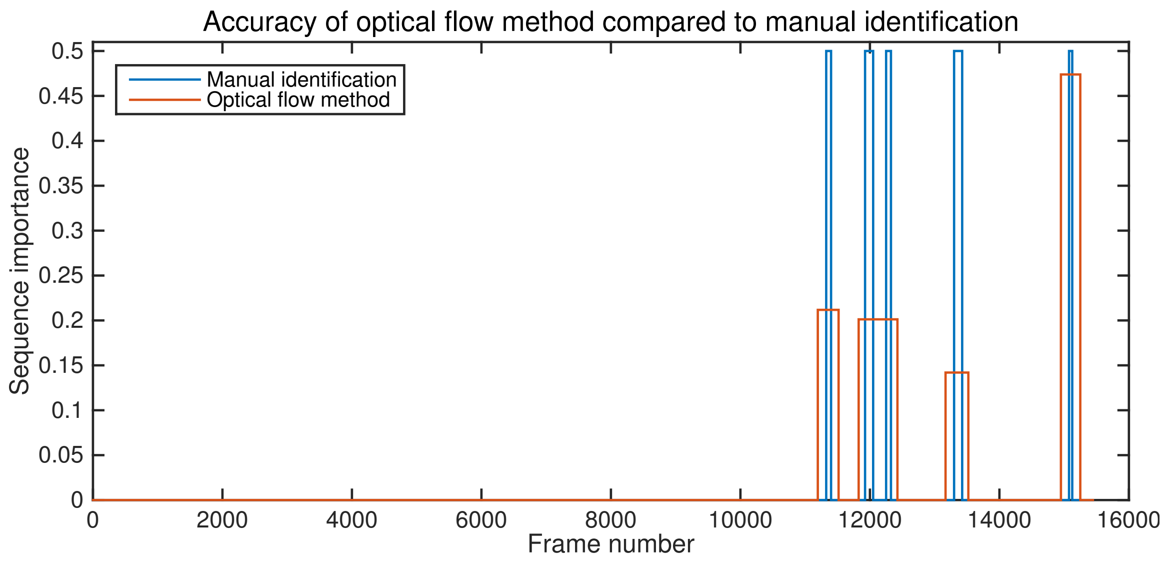
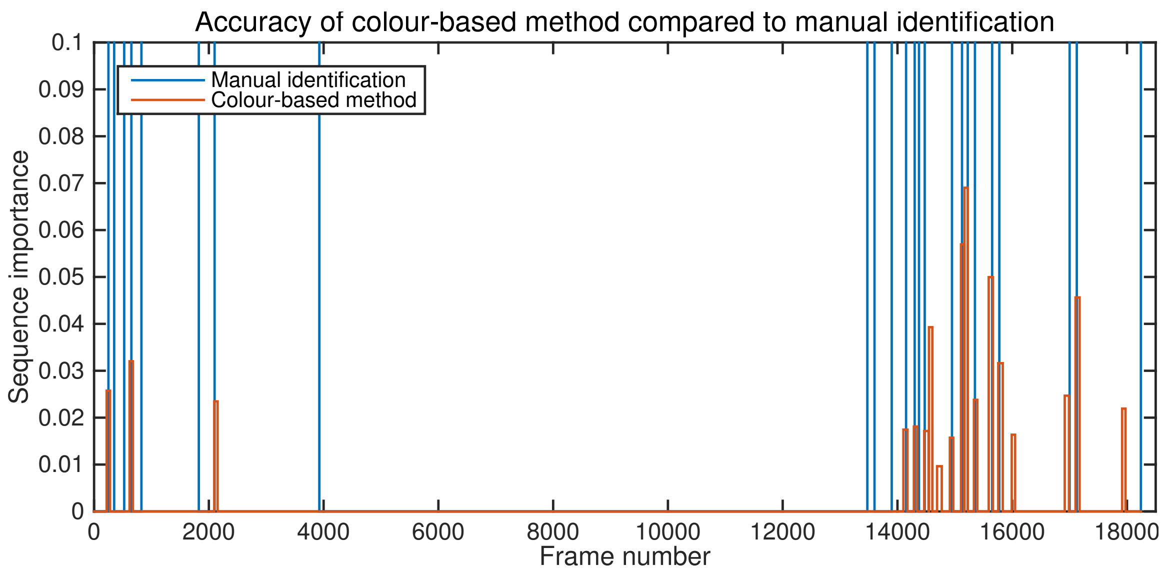


Figure 4.2: The performance of each algorithm on the bag video compared to the standard set by the manual event identification.

The second video had significantly more activity. The optical flow algorithm outperformed the colour-based algorithm, which, in this case, was too conservative and appeared to perform badly in the lower lighting conditions.

The third video was significantly more difficult due to large amounts of background noise from trees and only small amounts of motion from punts moving slowly past the camera. The algorithms were not expected to deal with this with a great amount of success. Nevertheless, both algorithms managed to capture most of the important events. The colour-based method did capture most things so this result is not significant. The optical



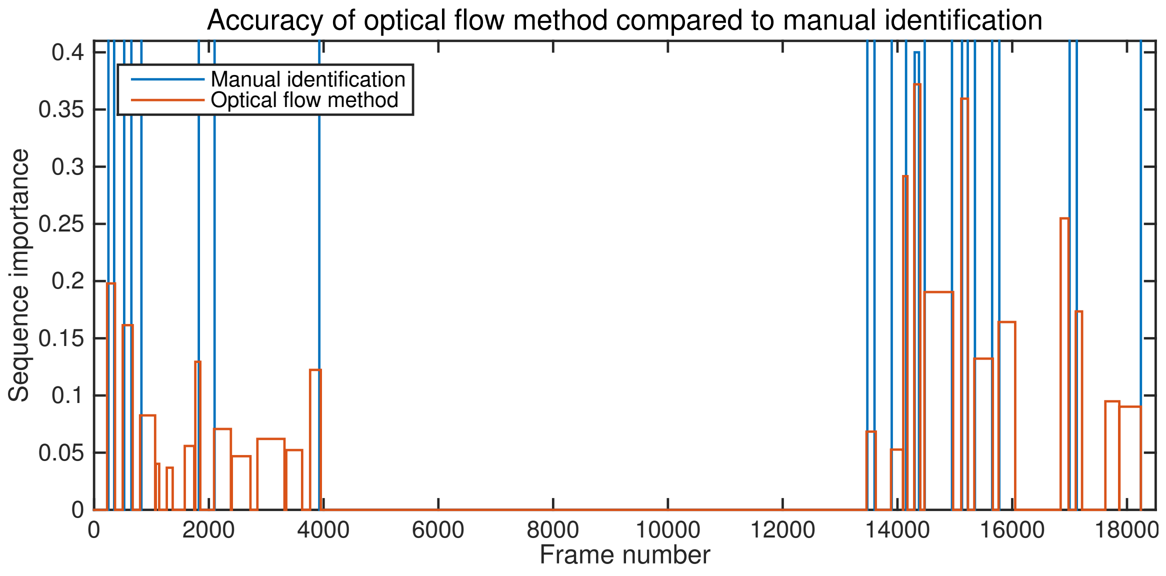
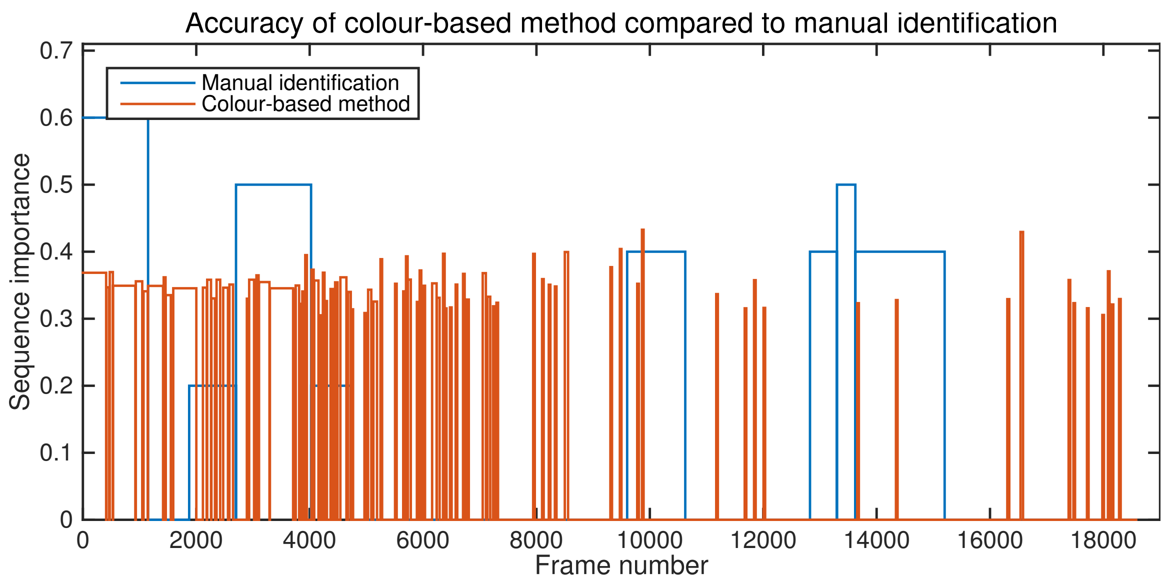


Figure 4.3: The performance of each algorithm on the desk video compared to the standard set by the manual event identification.

flow algorithm, on the other hand, was a lot more accurate, producing few false positives and only missing a single event.

4.3 User Study

Asking whether a summary of a video is useful is obviously subjective and would be a limiting factor to any user study. With this in mind, the user study was designed so that participants would comment on objective properties of the videos, giving quantitative answers that can be compared.



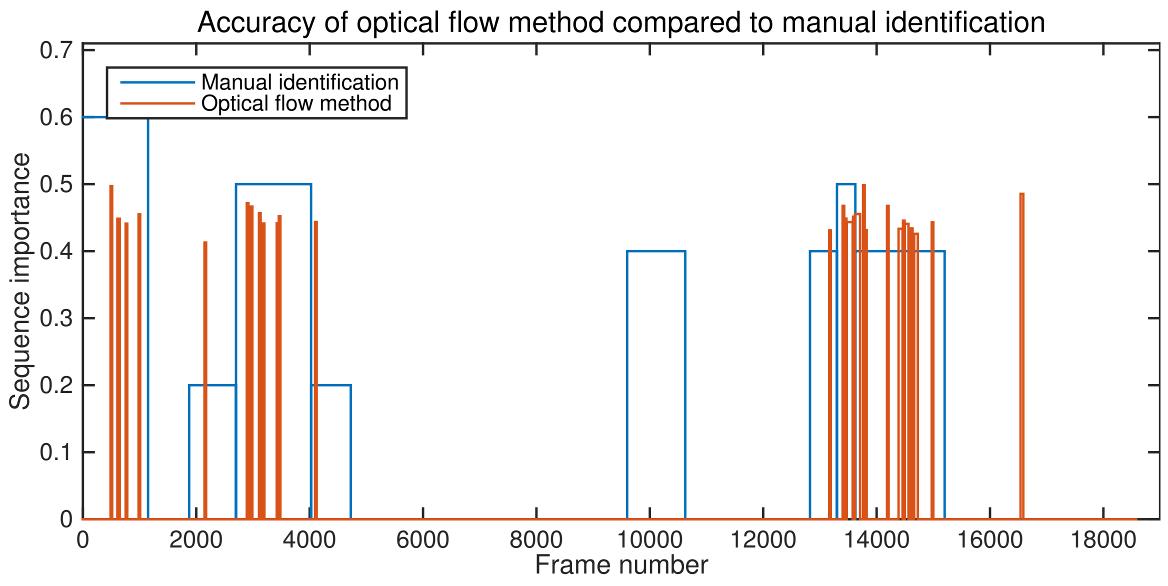


Figure 4.4: The performance of each algorithm on the river video compared to the standard set by the manual event identification.

4.3.1 Types of Video

The same three videos that were described above were used in the user study. To compare the colour-based method with the optical flow method, and the threshold output strategy with the speedup strategy, all four combinations of each video were generated, along with the original video. Table 4.1 shows all of the videos that were produced and the length of each video.

A participant would watch one of the videos and then answer some questions about the content of the video.

Summary Type	**Bag**	**Desk**	**River**
Full video	10:17	12:10	12:23
Colour / threshold	1:14	0:48	1:59
Optical flow / threshold	1:04	3:29	0:44
Colour / speedup	0:19	1:23	0:21
Optical Flow / Speed	0:06	0:32	0:14

Table 4.1: The lengths in minutes of each video included in the user study.

4.3.2 Types of Question

Subjective questions were avoided as they would lead to inconclusive results. Instead, questions with a restricted domain of answers were posed.

- Counting-based questions of the form "How many times did event X happen?".
- Time-based questions of the form "at what time did event X happen?". These videos had a timestamp embedded within them for participants to quote.
- True or false questions of the form "Did event X happen?".

The questions that were asked about each video are given in Appendix B.

Each participant was also given an optional comment box so that they could include any more detailed or higher-level thoughts about the video that they just watched and the questions that they answered. The purpose of this was to gain a general understanding of how people were completing the questions and how satisfied they were with the video and the survey process. For example, it could be used to explain that the whole video was not watched but instead the participant manually skipped through the video. This information helped when assessing the validity of the results and will be useful when designing future user studies.

4.3.3 Participant Recruitment

Participants for the user study were recruited using a crowdsourcing approach. A link to the website where users could take part was posted to Twitter and a video was posted to my YouTube channel. Crowdsourcing allowed a large amount of data to be collected over a short period of time and with minimal effort per participant.

The questions that participants were asked were not challenging and did not require any specialist knowledge except an understanding of English. Figure 4.5 shows that my YouTube subscribers are mostly from English speaking countries. It was not expected for age or gender to affect the results, but the demographics show a reasonable gender distribution nonetheless.

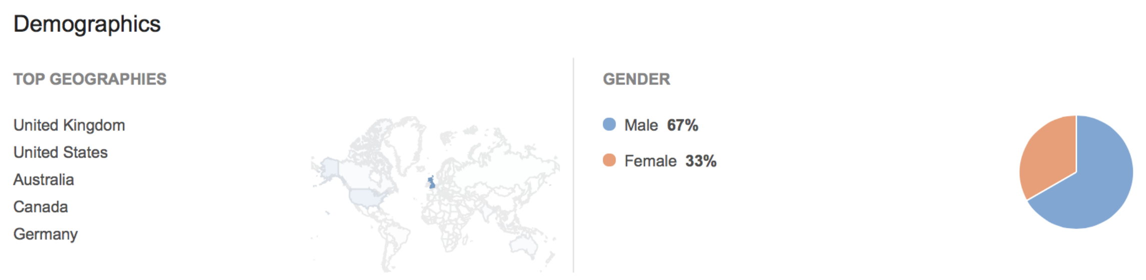


Figure 4.5: A snippet of my YouTube channel subscriber demographics, showing the distribution of top geographies and gender.

4.3.4 Ethics

Nobody was personally contacted and asked to complete the survey, instead, volunteers clicked the link themselves. Only the respondents' answers and the time spent on the page were recorded and only upon submission of the form. All questions were optional and participants were free to abandon the survey at any time. The purpose of the survey was made clear to participants both in the video and on the website itself. Participants were made aware of what data was being collected and how it was going to be used. Approval from the departmental ethics committee was granted before the user study began.

4.4 Results

First, the results from the user study were used to compare the output write strategies. The results were then used to compare the importance detection algorithms. Finally, the usefulness was compared to the results from the full-length videos.

4.4.1 Best Output Write Strategy

To determine the better output write strategy, the respondents' average response times were compared. The speedup method was expected to cause less confusion due to having fewer jump cuts in the video and therefore produce lower average response times. The results in Table 4.2 support this hypothesis, with the speedup method producing roughly a 30% improvement in response times when compared with the threshold method.

Video	Threshold/s	Speedup/s	Improvement
Bag	112	76.9	31.2%
Desk	193	139	28.2%
River	149	101	32.0%

Table 4.2: The average time taken by a respondent to watch a video and answer questions about its content. The percentage improvement in response times for a video presented using the speedup write strategy compared to the threshold strategy is shown.

Of course, improved response times are only useful if the respondents were still correctly answering the questions. Table 4.3 summarises the accuracy with which participants answered the questions, comparing the write strategies. For time-based questions, the root-mean-squared error (RMSE) is given.

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{Y_1} - Y_i)^2} \tag{4.4.1}$$

where $\hat{Y}$ is the vector of results from the user study, Y is the vector of correct answers for the question, and n is the number of responses.

For the counting and boolean-based questions, the percentage of incorrect answers out of the total number of responses is shown.

For the bag video, the results are very close, with the threshold strategy marginally taking the lead. For the other two videos, the speedup strategy is a more clear winner. The river video had poor results in the manual evaluation. Consequently, participants struggled to correctly answer the questions, with 98% of users answering one question incorrectly under the threshold write strategy. However, the speedup strategy clearly helped people to understand the story of the video and make sense of the events that were happening.

Interestingly, in almost all cases, the summaries out-performed the full-length video. This is probably due to users manually skipping through the longer videos and missing important events. When investigating the average time taken to complete the exercises for the full-length videos, it was found that users would usually finish in a time 40% faster than the video's actual length. This confirmed that users were skipping through the video and essentially performing a manual on-the-fly summary. Of course, however, we see that users performed better with the algorithmic summaries both in terms of speed and accuracy.

4.4.2 Best Importance Detection Algorithm

Again, the average response times were compared to find out if users could complete the exercise more quickly when the video had been summarised using the optical flow method. Table 4.4 shows that this was true for two of the videos, however users were generally

Video	Question	Question Type	Statistic	Full video	Threshold	Speedup
Bag	1	Time-based	RMSE	2.21	**2.12**	2.74
	2	Counting	% incorrect	5.56	**3.21**	4.26
	3	True or false	% incorrect	5.09	5.05	**2.84**
Desk	1	Time-based	RMSE	**0.351**	4.73	1.75
	2	Counting	% incorrect	1.49	1.01	**0**
	3	Time-based	RMSE	**0.671**	1.32	1.31
River	1	Counting	% incorrect	30.2	98.0	**23.9**
	2	Counting	% incorrect	20.8	30.0	**13.0**
	3	Counting	% incorrect	13.2	12.0	**10.9**
	4	Counting	% incorrect	18.9	**9.0**	13.0

Table 4.3: The user study results comparing the output write strategies' effects on the accuracy with which respondents answered questions about the videos. The accuracy for time-based questions is given as the root-mean-squared error, while counting and boolean questions' results are given as the percentage of incorrect answers. For each question, the best result is highlighted in bold.

Video	**Colour-based/s**	**Optical Flow/s**	**Improvement**
Bag	105	89.6	14.6%
Desk	150	181	-20.7%
River	144	110	323.2%

Table 4.4: The average time taken by a respondent to watch a video and answer questions about its content. The percentage improvement in response times for a video summarised with the optical flow algorithm over the colour-based algorithm is shown.

slower for the third video. To gain a more complete picture, the accuracy of the responses was analysed. Table 4.5 summarises the results.

There was no clear better algorithm, and most likely the colour-based method is better in some cases while in others, optical flow would produce better results. It is worth noting that questions 3 and 4 of the river video assessed the respondent's overall sense of what took place in the video, suggesting that with optical flow, users were more confident about which events had taken place and which had not.

4.4.3 Usefulness of the Summaries

So far, the results have shown that the speedup output write strategy is more successful than the threshold method, and optical flow, in most of the test cases, out-performed the colour-based algorithm. It was investigated whether a video summarised with the

Video	Question	Question Type	Statistic	Full video	Colour-based	Optical Flow
Bag	1	Time-based	RMSE	**2.21**	2.28	2.52
	2	Counting	% incorrect	5.56	**3.59**	3.66
	3	True or false	% incorrect	5.09	4.62	**3.66**
Desk	1	Time-based	RMSE	**0.351**	1.75	5.09
	2	Counting	% incorrect	1.49	1.11	**0**
	3	Time-based	RMSE	0.671	**0.476**	1.72
River	1	Counting	% incorrect	**30.2**	64.4	60.8
	2	Counting	% incorrect	20.8	**17.8**	25.5
	3	Counting	% incorrect	13.2	14.4	**8.82**
	4	Counting	% incorrect	18.9	14.4	**7.84**

Table 4.5: The user study results comparing the importance detection algorithms' effects on the accuracy with which respondents answered questions about the videos. The accuracy for time-based questions is given as the root mean squared error, while counting and boolean questions' results are given as the percentage of incorrect answers. For each question, the best result is highlighted in bold.

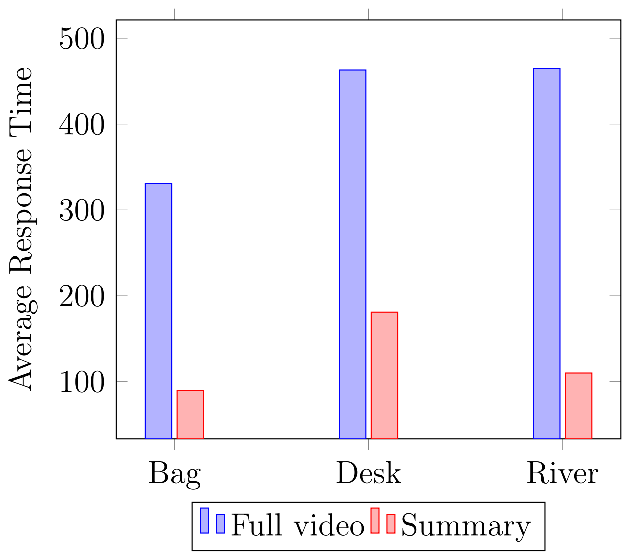
combination of optical flow and the speedup strategy is more useful than the original un-summarised video.

Figure 4.6 shows the improved response times of this summary method and the accuracy with which users answered the questions in comparison with the original un-summarised videos. The response times are drastically shorter and in most cases, the accuracy is higher. In the remaining cases, there was only a small decrease in accuracy—the mean of each set of responses was always centred on the true value, which suggest that users would be able to correctly answer the questions if the task was critical.

4.5 Threats to Validity

Some problems with the user study were identified that could affect the validity of the results. Firstly, it was suspected and then confirmed that participants did not watch the full videos before attempting the questions. Although this makes the comparisons between the summary methods less fair, it makes the overall study more realistic (people can and will skip through videos in real tasks), which can strengthen the conclusions regarding usefulness.

The respondents were estimated to be mostly situated in English speaking countries, suggesting that there would be no language barrier when answering the questions. This was not the case for all participants, with one notable response questioning whether a



Video	Question	Question Type	Full video	Summary
Bag	1	Time	**2.21**	3.71
	2	Count	5.56	**4.17**
	3	Boolean	5.09	**4.17**
Desk	1	Time	**0.351**	1.40
	2	Count	1.49	**0**
	3	Time	**0.671**	1.70
River	1	Count	30.2	**22.0**
	2	Count	20.8	**16.0**
	3	Count	13.2	**8.0**
	4	Count	18.9	**10.0**

Figure 4.6: Left: the improved response times when a video is summarised using the optical flow importance detection algorithm and presented using the speedup output write strategy. Right: The accuracy of respondents' answers with this summary type compared to the full un-summarised video.

punt was a 'small ship' and a pedestrian a 'large ship'! Resulting erroneous data was assumed not to be a problem for two reasons:

1. 'Impossible' answers were not aggregated. That is, answers that fell beyond the possible range given the constraints of the video were discarded.
2. The number of responses was large (1123 responses) so errors from lack of understanding would not have impacted the results significantly.

Finally, the reliability of the results collected via crowdsourcing was unclear, particularly since no reward was given to participants. Nowak and Rüger [18] showed that the results from crowdsourcing are of comparable quality to those from experts. Furthermore, the fact that domain checks were done on the collected data ensured that any malicious attempts to sabotage the results were excluded.

4.6 Summary

This chapter presented the results of a manual evaluation of the algorithms' performance and the results of the user study that was conducted to assess the usefulness of each importance detection strategy and each output write strategy.

- The videos that were selected to be used in the evaluation are described in Section 4.1.
- Section 4.2 gives the results of the manual evaluation. It showed that the algorithms were performing as expected with the test videos, capturing at least 75% of the important events.
- The details of the user study are given in Section 4.3. The types of summary that were included are described, as are the types of questions that were asked. Details regarding participant recruitment and the ethics of the study are given.
- The results, given in Section 4.4, showed that the optical flow method outperformed the colour-based method in most cases, and the speedup output write strategy was more useful than the threshold strategy. Furthermore, it was shown that the summaries were more useful to users than the original full-length videos.
- Finally, Section 4.5 discusses some potential threats to the validity of the results and the actions taken to mitigate them.

Chapter 5

Conclusions

This project set out to compare the usefulness of two video summarisation techniques: a method based on colour change and a method based on optical flow. It also compared two methods of presenting the output: a threshold based method and a *speedup* method. A video summarisation tool was built using C++ and summaries produced for three test videos. A user study was carried out alongside a manual evaluation to assess the usefulness of the summaries.

5.1 Achievements

The project's implementation was successful because all of the *must* requirements set out in Section 2.2 were met. Requirement M1 stated that summary sequences should be shorter than the input sequence. This was the case for every video used for evaluation. Table 4.1 shows that every summary was more than 70% shorter than the corresponding original video.

The graphs in Section 4.2 show that all videos retained at least 75% of their important events according to the standard set by the manual summary, thus achieving requirement M2.

Chronological consistency was preserved in the outputs by merging the subsequences as described in Section 3.3.4. This achieved requirement M3. Finally, both importance detection algorithms and both output write strategies were implemented, meeting requirements M4 and M5.

The videos that were recorded for the evaluation were recorded in Full HD to allow the algorithms to be tested without being limited by video resolution. This met requirement M6. For the sake of performance, the resolution of each video was scaled down during development.

The running time of the program when producing a summary obviously depends on the processing power of the machine. On the machine used for development and testing, the summarises were produced faster than the input video would take to play in its entirety thereby meeting requirement S1 of summarising in real-time or better.

The development of the web platform on which to conduct the user study was very successful, allowing over 1000 responses to be submitted without any known technical problems thus meeting requirement S2.

As described in Chapter 3, the project was built with a high level of modularity and extensibility. The generalisation of the algorithms to *processors* meets requirement S3 by making it possible to implement and compare more algorithms. The ability to add image modifiers to the video reader made cropping possible, meeting requirement C1, however this was not actually used during evaluation.

5.2 Future Work

The extensibility of the final program will make it easy to continue research in the area of video summarisation. The combination of the two implemented importance detection strategies can be tested without needing any further development. New summarisation techniques can be implemented as additional processors and compared against the existing algorithms.

A user interface, as proposed by requirement W1 could be built to enable easier summarising of videos and make the system accessible to non-experts.

Future work could utilise extra information associated with videos such as auditory channels. As mentioned in Chapter 1, peaks in the audio waveform can be correlated with important events. Speech recognition could also be employed to further understand and interpret the scene.

5.3 Final Remarks

With the benefit of hindsight, a local threshold would have been a better way to mark whether frames were important or not. Using a global threshold meant that anomalous frame importances negatively effected the entire summary unless the importance threshold was manually refined.

Despite this drawback, the optical flow method usually out-performed the colour-based method for summarising videos, though the optimal algorithm was video dependant. A strong conclusion was made regarding the effectiveness of the two output write strategies, with the speedup method being at least as effective, and usually more so, in every tested case. Overall, a summarised video was found to be consistently more useful than the original full-length video when produced using optical flow and presented by speeding up the less significant parts.

Bibliography

[1] Edoardo Ardizzone and Marco La Cascia. Video indexing using optical flow field. In *ICIP (3)*, pages 831–834. IEEE, 1996.

[2] Michael J. Black and P. Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Comput. Vis. Image Underst.*, 63(1):75–104, January 1996.

[3] Kevin Brennan et al. *A Guide to the Business Analysis Body of Knowledge.* International Institute of Business Analysis, 2009.

[4] Andrew Burton and John Radford. *Thinking in Perspective: Critical Essays in the Study of Thought Processes.* Methuen, 1978.

[5] H.D. Cheng, X.H. Jiang, Y. Sun, and Jingli Wang. Color image segmentation: advances and prospects. *Pattern Recognition*, 34(12):2259 – 2281, 2001.

[6] Vikas Choudhary and Anil Kumar Tiwari. Surveillance video synopsis. In *ICVGIP*, pages 207–212. IEEE, 2008.

[7] E. F. Codd. Recent investigations in relational data base systems. In *IFIP Congress*, pages 1017–1021, 1974.

[8] Juanita Ellis, Charles Pursell, and Joy Rahman. *Voice, Video, and Data Network Convergence: Architecture and Design, From VoIP to Wireless.* Academic Press, September 2003.

[9] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Proceedings of the 13th Scandinavian Conference on Image Analysis*, LNCS 2749, pages 363–370, Gothenburg, Sweden, June-July 2003.

[10] David J. Fleet and Yair Weiss. Optical flow estimation, 2005.

[11] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *Computer*, 28(9):23–32, September 1995.

[12] James Jerome Gibson. *The Perception of the Visual World.* Houghton Mifflin, 1950.

[13] Wei Jiang, Courtenay Cotton, and A.C. Loui. Automatic consumer video summarization by audio and visual analysis. In *Multimedia and Expo (ICME), 2011 IEEE International Conference on*, pages 1–6, July 2011.

[14] Zaynab El Khattabi, Youness Tabii, and Abdelhamid Benkaddour. Video summarization: Techniques and applications. *International Journal of Computer, Control, Quantum and Information Engineering*, 9(4):11 – 16, 2015.

[15] Arthur G. Money and Harry W. Agius. Video summarisation: A conceptual framework and survey of the state of the art. *J. Visual Communication and Image Representation*, 19(2):121–143, 2008.

[16] Jeho Nam and Ahmed H. Tewfik. Dynamic video summarization and visualization. In *Proceedings of the Seventh ACM International Conference on Multimedia (Part 2)*, MULTIMEDIA '99, pages 53–56, New York, NY, USA, 1999. ACM.

[17] Clive Norris. A review of the increased use of CCTV and video-surveillance for crime prevention purposes in europe. *Briefing Paper for Civil Liberties, Justice and Home Affairs Committee (LIBE), European Parliament: Brussels*, April 2009.

[18] Stefanie Nowak and Stefan Rüger. How reliable are annotations via crowdsourcing: A study about inter-annotator agreement for multi-label image annotation. In *Proceedings of the International Conference on Multimedia Information Retrieval*, MIR '10, pages 557–566, New York, NY, USA, 2010. ACM.

[19] Ofir Pele and Michael Werman. The quadratic-chi histogram distance family. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision - ECCV 2010*, volume 6312 of *Lecture Notes in Computer Science*, pages 749–762. Springer Berlin Heidelberg, 2010.

[20] Bernt Schiele and James L. Crowley. Object recognition using multidimensional receptive field histograms. *European Conference on Computer Vision*, 1:610–619, April 1996.

[21] Linda G. Shapiro and George C. Stockman. *Computer Vision*. Prentice Hall, 2001.

[22] Brian Sims. UK CCTV: Why we need more regulation. `http://www.ifsecglobal.com/uk-cctv-why-we-need-more-regulation/`, 2013.

[23] G.W.A. Snedecor and W.G.A. Cochran. *Statistical Methods*. Iowa State University Press, 1967.

[24] Michael J. Swain and Dana H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, November 1991.

Appendix A

User Study Results Aggregation Script

```
// Include the database configuration
require("config.php");

// Connect to the database
$mysqli = new mysqli(DB_HOST, DB_USER, DB_PASSWORD, DB_NAME);

$query = "SELECT Response.ResponseID, SummaryType.SummaryTypeID,
Video.VideoID, Video.VideoTitle, Response.TimeTaken, Response.Comments,
Question.QuestionID, Question.QuestionText, Question.QuestionTypeID,
QuestionType.QuestionTypeName, ResponseAnswer.Answer
FROM ResponseAnswer
LEFT JOIN Response ON ResponseAnswer.ResponseID=Response.ResponseID
LEFT JOIN VideoVersion
ON Response.VideoVersionID = VideoVersion.VideoVersionID
LEFT JOIN Question ON ResponseAnswer.QuestionID = Question.QuestionID
LEFT JOIN Video ON VideoVersion.VideoID = Video.VideoID
LEFT JOIN SummaryType
ON VideoVersion.SummaryTypeID = SummaryType.SummaryTypeID
LEFT JOIN QuestionType
ON Question.QuestionTypeID = QuestionType.QuestionTypeID;";

// Execute the query
$result = $mysqli->query($query)

// Put the results into an array called response
while ($row = $result->fetch_assoc())
{
  // Get the question type
  $questionTypeID = $row['QuestionTypeID'];
  $questionTypeName = $row['QuestionTypeName'];

  // Get the Video
  $videoID = $row['VideoID'];
  $videoTitle = $row['VideoTitle'];
```

```
// Get the Question
$questionID = $row['QuestionID'];
$questionText = $row['QuestionText'];

// Get the summary type
$summaryTypeID = $row['SummaryTypeID'];

// Get the time taken
$timeTaken = $row['TimeTaken'];

// Transform the answers if necessary
switch ($questionTypeName) {
  case 'Time':
    // Iterpret the date and output as an integer
    $answer = date("h", strtotime($answer)) * 60
        + date("i", strtotime($answer));
    break;
  case 'Boolean':
    $answer = ($answer == "Yes" ? 1 : 0);
    break;
}

// Ignore out of bound results
switch ($questionID) {
  case 1:
    if ($answer < 0 || $answer > 10) continue 2;
    else break;
  case 2:
    if ($answer < 0 || $answer > 10) continue 2;
    else break;
  case 4:
    if ($answer < 0 || $answer > 10) continue 2;
    else break;
  case 5:
    if ($answer < 180 || $answer > 240) continue 2;
    else break;
  case 6:
    if ($answer < 0 || $answer > 6) continue 2;
    else break;
  case 8:
    if ($answer < 684 || $answer > 697) continue 2;
    else break;
  case 9:
    if ($answer < 0 || $answer > 5) continue 2;
    else break;
  case 10:
    if ($answer < 684 || $answer > 697) continue 2;
    else break;
}

// Add the answer to the array
$answers[$videoID][$questionID][$summaryTypeID][] = $answer;
```

```
  // Add the video to the video array
  $videos[$videoID] = $videoTitle;

  // Add the question to a question array
  $questions[$videoID][$questionID] = [
          "QuestionText" => $questionText,
          "QuestionTypeName" => $questionTypeName
        ];

  // Add to the respondent array
  $respondents[$videoID][$summaryTypeID][] = $timeTaken;
}

// Build a nested array of response answers
foreach ($videos as $videoID => $videoTitle)
{
  $questionsForVideo = [];
  foreach ($questions[$videoID] as $questionID => $question)
  {
    $questionsForVideo[$questionID] = [
          "QuestionInfo" => $question,
          "Answers" => $answers[$videoID][$questionID]
        ];
  }

  $responseAnswers[$videoID] = [
        "VideoTitle" => $videoTitle,
        "Questions" => $questionsForVideo
      ];
}

// Returns the percentage of correct answers
function percentageIncorrect($array, $correctAnswer)
{
  for($i=0, $last=count($array), $incorrect=0; $i<$last;$i++)
    if ($array[$i]!=$correctAnswer) $incorrect++;
  return round(100 * $incorrect / (count($array)), 3);
}

// Returns the root-squred-mean error
function RMSE(array $a, $actual) {
    $n = count($a);
    $carry = 0.0;
    foreach ($a as $val) {
        $d = ((double) $val) - $actual;
        $carry += $d * $d;
    };
    return round (sqrt($carry / $n), 3);
}

// Returns the average of the input array
function averageArray($array)
```

```
{
  return array_sum($array) / count($array);
}

// Returns the percentage improvement of x2 over x1
function percentageImprovement($x1, $x2)
{
  return round(100 * (1 - ($x2 / $x1)), 3);
}
?>
```

Appendix B

User Study Questions

B.1 Bag Video

1. At what time was the bag first taken from the table?
2. How many times is the bag put down onto the table?
3. Were any other items placed on the table?

B.2 Desk Video

1. At what time was the green cup removed from the desk?
2. How many people were involved in tidying the desk?
3. At what time was a sheet of paper placed into a green ring binder?

B.3 River Video

1. How many punts go past?
2. How many pedestrians go past?
3. Was there a dog visible at any point in time?
4. What is the maximum number of punts visible at any one time?

Appendix C

Project Proposal

Computer Science Project Proposal
Video Summarisation

Jake Wright, Queens' College
Originator: Jake Wright

May 12, 2015

Introduction, The Problem To Be Addressed

Due to advancements in video recording and storage technologies, there is an increasing amount of raw, unedited video footage, particularly from sources such as CCTV cameras and dashboard cameras in vehicles. Editing all of this video or finding particular events of interest is tedious and time consuming, especially when dealing with surveillance footage that is continuously recording.

During my project, I will create a system that will, when given an input video, automatically produce a shorter video that summarises the original one. This will allow the uneventful video footage to be ignored, saving time when reviewing the video and potentially saving storage space if the summarisation is sufficient for archival purposes. My project will focus on video footage where the camera remains stationary as this is the most common scenario for surveillance footage.

I will compare and contrast at least two different methods of summarisation. Firstly, I will implement an approach that analyses and compares the colour histograms of frames. Video segments will be regarded as important if the difference in colour between the neighbouring segment exceeds a specified threshold.

Secondly, I will explore approaches based on motion. With this method, video segments with the most motion could be included in the summary, for example. Further methods of analysis will be investigated as possible extensions to the project. Various combinations of the algorithms will also be tried and evaluated based on their performance both in terms of time and accuracy.

Starting Point

My project will utilise the OpenCV software library as it already provides efficient implementations of useful computer vision algorithms that cover many areas, for example, motion tracking and histogram computation. This means I will not need to spend extensive amounts of time building systems to perform mathematical analysis of video frames. OpenCV has a C++ interface so, being a language I have some prior experience with, this the most suitable language for my project. There is existing work in the field of video summarisation, allowing me to research various approaches to the problem [1].

Resources Required

I will use my own machine for the project and write the code using software that I already have. If this should fail, I will be able to continue my work on an MCS machine. All files associated with the project will be backed up using an hourly, incremental backup system. I will also use github for version control. The video footage necessary for the project will be captured manually, as planned in the timeline. I have existing experience in this area and a range of cameras that I can use. If this should fail, I have access to further equipment within college. Similarly to the project files, any video files that are used in the project will be backed up.

Work to be done

The project can be broken down into the following sections:

1. Collect sample video footage that can be used to test and evaluate the summarisation process. The video will be captured manually using cameras that I already have. Video from various environments will be required to fully test the performance of the algorithms. I plan to collect sample footage from locations including my college bedroom, a view of the river from a window and the DTG corridor in the computer lab.
2. Work with OpenCV to implement a working method of processing and outputting video files.
3. Implement procedures to analyse properties of the video such as colour histograms and optical flow between frames. Based on the success of these methods, further algorithms may be implemented such as analysing shape-based features. Many options are presented by Hu et al. [2].
4. Create an algorithm to determine the importance of each segment of video and combine those that are considered most important. The task of combining the most important segments of video is also non-trivial and must be investigated further. Ideally, the resulting video will be smooth to watch and the progression of time will be visible to the user. Transitions or interpolation between frames of non-contiguous sections or speeding up of unimportant sections are possible ways of achieving this goal.
5. Evaluate the success of the project by conducting a user study to compare the summarised video with the original video. The project can also be evaluated by comparing the results to a manual summarisation and finally by testing the algorithmic performance.

Success Criterion for the Main Result

The most simple criterion for the success of my project would be to see whether or not it was capable of producing a shorter version of an original video. This, however, would not be a particularly useful metric in any real world application. For this reason, the success of my project will be based on how well the algorithms perform when given a video to summarise. The goal is not necessarily to build a tool that achieves video summarisation but to compare the algorithms' performance.

A user study will be performed during which participants will be given a chance to watch the original, unedited video, followed by the shorter, summarised version. While watching each one, they will be asked specific, quantitative questions about the video. These could include questions like "At what time was the bag removed from the table?" or "How many times did person X enter the scene". The user's ability to correctly answer the questions and the time it takes them to do so will be used to measure the successfulness of the algorithms.

Furthermore, the effectiveness of the summarisation algorithms will be measured by carrying out the described user study on videos with varying degrees of difficulty in the environments. "Simple" videos would be ones that are mostly static with areas of importance differing significantly from the uneventful sections. A more difficult video for the program to analyse would have lots of noise, for example, moving trees in the background.

The user study will be carried out using two methods. Firstly, I will invite a small number of participants to view the two videos on my machine, while I supervise the process. This will give me reliable but limited results. To gain a higher number of results, and thus aid in statistical analysis, I will invite remote users to complete the same survey after reviewing two videos, but they will do so on their own machines, unsupervised. Of course, this creates the risk that they will not honestly answer the questions and so conclusions based upon these results will be made with caution.

To allow users to complete the evaulation on their own machines, I will create a web interface to allow the users to firstly watch the videos, and then answer questions. I have lots of experience in web development and online video. This simple interface will be hosted online and not take a significant amount of time to complete.

Finally, if there is insufficient time for the user study, or I cannot find enough participants, the project can be evaluated also by comparing the results of the automatic summarisation with a "manual" summarisation. This will involve a person identifying the important events in a video and then finding how closely the automatic summary matches the expected outcome.

Possible Extensions

A possible extension to the project would be to create an easy-to-use user interface and use this to add extra functionality to the system. For example, it could alllow particular areas of the video to be ignored during analysis. This could be useful if, for instance, there is a constantly changing background like cars in a street, but the user is not interested in that section of the video. They could select this portion and have it not considered when summarising the video.

As mentioned previously, further extensions would be to implement and compare more algorithms that can summarise the video.

Timetable: Workplan and Milestones to be achieved.

1. **Michaelmas weeks 2-4** Learn how to use OpenCV and research previous work in the area of video summarisation [1, 2]. The goal with OpenCV is to be able to read and process video files. *Deadline: Wednesday 5th November.*

2. **Michaelmas week 5** Collect test footage to use throughout the development and evaluation of the project. As discussed previously, this footage will be recorded manually in a variety of locations, including Queens' college and the computer lab. I will seek appropriate permission before conducting any filming that will involve others. *Deadline: Wednesday 12th November.*

3. **Michaelmas week 6** Experiment with creating smooth video by comparing the results of cutting out segments, transitioning between segments, and speeding up sections of the video. Areas of interest at this stage will be defined manually so that code can be written to output the summary. *Deadline: Wednesday 19th November.*

4. **Michaelmas weeks 7-8** Start implementation of project. A colour-based method will be implemented first, followed by a contrasting motion based method [2]. *Deadline: Wednesday 3rd December.*

5. **Michaelmas vacation** Finish the implementation of the two main algorithms and then compare the performance of each, and explore combinations of each technique. I will then decide, by comparing the results with a manual summary, which option is the most effective and use this for the user study. The online review system will also be created towards the end of the vacation. *Deadline: Wednesday 7th January.*

6. **Lent weeks 0-2** Write progress report and begin evaluation by inviting the first participants to review the videos. Videos collected in week 5 will be summarised using the system completed over the vacation. *Deadline: Wednesday 28th January.*

7. **Lent weeks 3-5** Continue user evaluation of completed project. Since the evaluation will involve the user watching at least one long, un-summarised video, I will need to allow plenty of time for the study because participants cannot be expected to do this immediately. *Deadline: Wednesday 18th February.*

8. **Lent weeks 6-8** Complete and collect feedback from evaluation. The results will be analysed and conclusions regarding the algorithms' performance will be drawn. *Deadline: Wednesday 11th March.*

9. **Easter vacation** Write dissertation. This will be started early so that plenty of time is left for exam revision. *Deadline: Wednesday 15th April.*

10. **Easter term weeks 0-2** Proof read dissertation and submission. *Deadline: Wednesday 6th May.*

Bibliography

[1] ZHAO, B., XING, E. P. (2014) Quasi Real-Time Summarization for Consumer Videos `https://www.cs.cmu.edu/~epxing/papers/2014/Zhao_Xing_cvpr14a.pdf`

[2] HU, W., et al (2011) A Survey on Visual Content-Based Video Indexing and Retrieval *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews* 41 (6) `http://www.dcs.bbk.ac.uk/~sjmaybank/survey%20video%20indexing.pdf`