MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

"Composition of Highlight Videos by Modeling and Rating Events from Sport Broadcasts and its Metadata"

verfasst von / submitted by

Christoph Draschkowitz BSc

angestrebter akademischer Grad / in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur (Dipl.-Ing.)

Wien 2018 Vienna, 2018

Studienkennzahl lt. Studienblatt / degree program code as it appears on the student records sheet:	A 066 935
Studienrichtung lt. Studienblatt / degree program as it appears on the student record sheet	Masterstudium Medieninformatik
Betreut von / Supervisor:	Univ. Prof. Dr. Wolfgang Klas

Declaration of Authorship

Acknowledgments

Thanks to all supporters. Fail Forward. It’s simple, but it’s not easy.

Abstract

Motivation of this work is the fact that the creation of highlight videos nowadays takes a lot of manpower and is time consuming. There is a rising interest in global sports and more and more sports get spread around the world. Big sport teams nowadays sell their rights to more than a 100 countries. Due to video cameras and technical equipment getting cheaper and better, even rather small sport events can be recorded and very often these are also getting broadcasted, for example on online channels. The demand for those broadcasts might be small, whereas a short summary of events might very often be something more people would want to watch. This master thesis uses a regular TV broadcast and a text file, exported from a live ticker about the game, as inputs and aims to produce a sport highlight video without the full assistance of experts in the progress. However, an expert and/or user could adapt the model on the fly, by giving inputs. Thus the overall thematic area of this work is on one hand the retrieval of multimedia features. On the other hand it is also about creating and adapting a model (for events) using various methods that include any type of data analysis. Results confirm that, by the help of timestamped play by play data and multimedia techniques, a very decent highlight video can be created. While the question remains if a fully automatically generated highlight video can consistently perform better than a manually created video, it could be proven that such a framework can be extremely helpful in the creation process, for example by choosing events beforehand, by detecting event boundaries and many more.

Keywords: Sports Highlight Composition, Automatic, Summarization, Video Metadata, Audio Features, Video Features, Event Rating, OpenCV, Python, Icehockey

Zusammenfassung

Die Motivation dieser Arbeit basiert auf dem Fakt, dass das Kreieren von Highlight Videos auch heutzutage noch zeitintensiv ist. Es existiert eine steigende Nachfrage nach Sportkonsumation und viele Sportarten verteilen sich mittlerweile über den gesamten Globus. Große Sport Teams verkaufen ihre Rechte an mehr als 100 Länder weltweit. Da Videokameras und technisches Equipment immer besser und billiger verfügbar sind, werden sogar Randsportarten wahrgenommen und sehr oft werden Spiele davon zum Beispiel in Online Kanälen live gezeigt. Während die Nachfrage danach oft eher gering ist, wäre eine Kurzzusammenfassung der Videos etwas, das viele Menschen ansprechen könnte. Diese Masterarbeit benutzt einen regulären TV Broadcast und ein Text File, das von einem Live Ticker über ein Eishockeymatch exportiert wird, als Input und produziert daraus ein Sport Highlight Video, ohne dazu auf die Hilfe von Experten zurückzugreifen. Ein User könnte jedoch über ein Interface Einfluss nehmen. Daher ist diese Arbeit thematisch im Bereich "Multimedia Retrieval" angesiedelt. Es geht jedoch auch darum Modelle zu erstellen und zu adaptieren, die sowohl Micro Events, als auch Regeln zur Komposition des Videos beschreiben. Die Resultate zeigen, dass mithilfe von "play by play" Daten mit Zeit Informationen, dem dazugehörigen Video und Multimedia Techniken, qualitativ gute Highlight Videos erstellt werden konnten. Die Frage ob vollautomatisch kreierte Videos dauerhaft besser als manuell erstellte Videos sein können, bleibt natürlich bestehen. Allerdings konnte bewiesen werden, dass ein solches Framework extrem hilfreich in einem solchen Prozess sein kann. Zum Beispiel um im vornherein interessante Events zu erkennen und Abgrenzungen zu schaffen, aber auch durch viele andere Möglichkeiten.

Keywords: Sports Highlight Composition, Automatic, Summarization, Video Metadata, Audio Features, Video Features, Event Rating, OpenCV, Python, Icehockey

Composition of Highlight Videos by Modeling and Rating Events from Sport Broadcasts and its Metadata

Christoph Draschkowitz

July 24, 2018

Table of Contents

1 Introduction

1.1 Motivation

Motivation of this work is the fact that the creation of decent highlight videos nowadays takes a lot of manpower and is time consuming. There is a rising interest in global sports and more and more sports get spread around the world. Big sport teams nowadays sell their rights to more than a 100 countries.

Due to video cameras and technical equipment getting cheaper and better, even rather small sport events can be recorded and very often these are also getting broadcasted in example on online channels. The demand for those broadcasts might be small, whereas a short summary of events might very often be something more people would want to watch.

The problem could be somehow related to the term "big data", used in many other domains. There is an immense amount of data, that is already existing, but it is very costly to make sense or to even make use of it.

While manually created videos can be expensive considering the amount of time that needs to be invested by experts, automatic video summarizations would be a time saving alternative, depending on the desired complexity and background of the resulting video. Cutting a highlight video in the "traditional" way, is more work than expected:

Usually the cutter, that finally composes the video and one other video expert, already watch the full game broadcast, making notes about the better scenes, about goal scorers, assistants and of course the match time of these events. One of them already cuts out the scenes from the full video without taking care about correct cuts and event boundaries. Sometimes this part is done by someone from the broadcasting crew during intermission.

If that is the case however, it is more difficult to gather the complete set of scenes, as their main task is to cut the broadcast. After that, the commentator of the final highlight video and the cutter choose the final events for the video and the cutter starts working on it.

If one looks closer at those steps, one can easily see that some of them are predestinated to being automated. So the most simplistic possible solution can be a framework that is able to provide an end user with a set of scenes that most likely represent all the important scenes of an sport event of any sport, without the claim of completeness and thereby assist an expert by taking away time consuming uncreative work.

If it would be able to construct a perfect computational program that can do the job, it would most likely still lack a human's ability of creativity. So the question was if it is possible to find ways to break the steps, that seem to be creative in this case, down to a combination of features and observations, to address the same problem in a way that a computer can solve it.

IBM was trying something like that in a very ambitious project of automatically creating a movie trailer [14]. At the end of the day, no computer can ever replace a human brain, but it can surely assist it to achieve greater things.

This master thesis uses a regular TV broadcast and a text file, exported from a live ticker about the game, as inputs and aims to produce a sport highlight video without the full assistance of experts in the progress. However, an expert and/or user could adapt the model on the fly, by giving inputs through a graphical interface.

Thus the overall thematic area of this work is on one hand the retrieval of multimedia features like zero crossing rate or boundary detection, to name popular ones for audio or shot cut, or replay detection and motion analysis for video features. On the other hand it is also about creating and adapting a model (for events) using various methods like data analysis and a rule set for the composition of the video. There are various (open source) helpful libraries, like OpenCV [1] or FFMPEG available, that are used for that purpose, depending on what works best.

The framework and its algorithms and methods are specialized and tested for the sport of ice hockey, because I play it myself and can therefore provide a lot of insights. It goes without saying that the provided videos and meta data files are also from that sport. Ice hockey specific rules and habits are going to be part of a composition model, that is created and used in this work. However, by changing these rules to other sports, the framework becomes generic.

[1] http://www.opencv.org/

1.2 Problem Statement

The initial starting points and inputs are regular TV broadcast videos of ice hockey matches from a professional league and a semi structured text file. Both the video and the data file are provided by "Sportradar" and are in this work referred to as input video and metadata text file. The meta data file is a "XML" file containing detailed play by play event information about the corresponding ice hockey game.

This work, in order to succeed in fully automatically creating an ice hockey highlight video, requires to convert the meta data file, the video stream and the audio stream, to a representation, where a self created rule model can be applied on the data. That rule set model has to take care of the final selection and composition of events. It has to consider for example, the importance of events on one hand, but also to choose events that are spread over the whole length of the game.

Audio, image, as well as motion features, that can characterize different event types, and therefore make them comparable, have to be found and implemented.

The metadata text file is used to identify event types and approximate event timestamps. This file is automatically extracted from live ticker data feeds and provided as a "XML" file. Once the events are defined, they have to get a score assigned, that finally builds the base for the composition. The final composition does also require the detection of the correct clip boundaries and a way to concatenate clips correctly (Figure 9).

Furthermore certain rules, that a well composed video consists of, have to be followed in order to choose the best events, shot types, fades and more.

For users, a possibility to specify the input files has to be implemented. At the end of the frameworks lifecycle the results shall be presented to the user.

To acquire data from the video, it is necessary to extract images, audio samples and video metadata from encoded video files and to make use of special methods and algorithms to extract features on the images and audio samples. Then the data can be applied to the rule model to create rankings and apply the events to the composition rule set. Figure 2 provides a quick overview of the compositional structure of the framework.

The final framework should allow a user to specify a broadcast ice hockey video and its corresponding metadata file and then fully automatically provide a finished

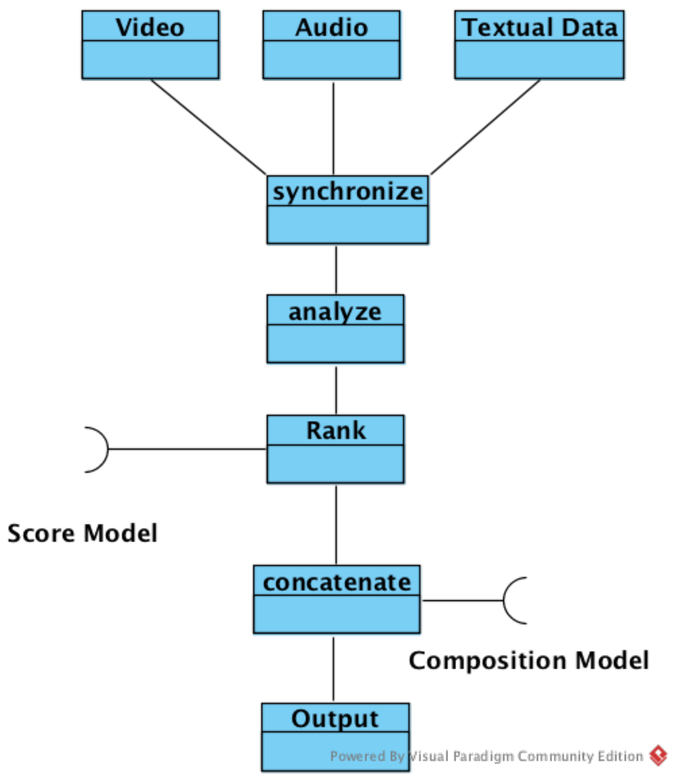


Fig. 1. Simplified overview of components

highlight video without further assistance. This requires all components to provide data in a way that it can be integrated into others. An unordered list of tasks that need to be addressed, can be seen in Figure 3.

The diagram describes the sub functions that the framework has to take care of, in order to successfully provide the service.

Firstly all kind of usable features are extracted from the broadcast video, as well as the metadata text file. These features will be taken from video, audio and of course the metadata, by using multimedia retrieval techniques like histogram comparison, optical flow, background subtraction, or Fourier transformations. Secondly a model that defines events and puts them in relationship to the taken features has to be created. It should then be possible to create a ranking of highlight related events by combining and weighting all features.

The last step is to compose a highlight video. In order to do so it is clearly necessary to detect event boundaries from the original broadcast. This again can

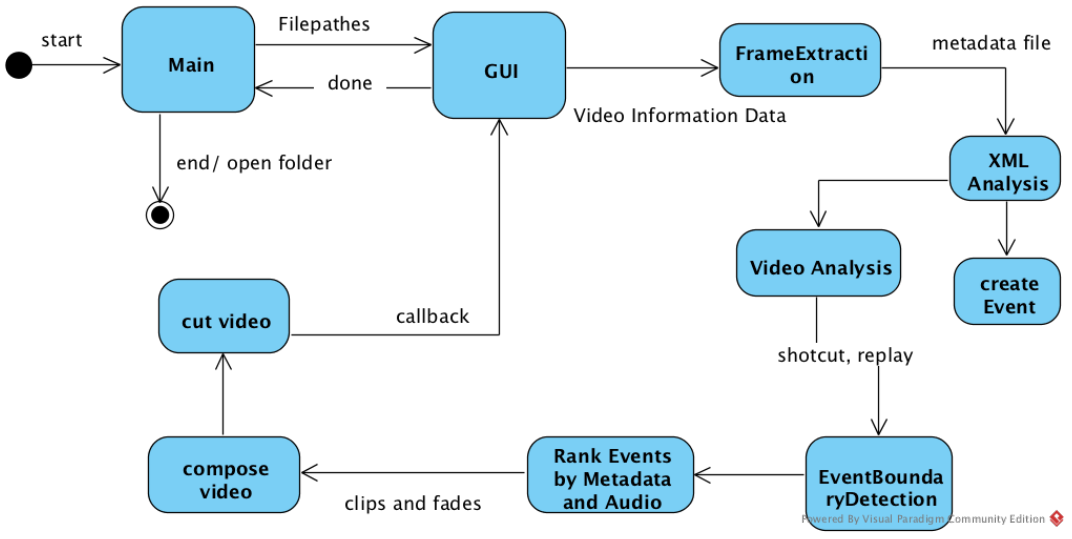


Fig. 2. Simplified Activity Diagram of the challenged tasks

be done by similar techniques as mentioned above. These last two steps could be generically parameterized by adjusting the ranking, or changing the composition to the special needs of a given user.

Additionally real time background information could be used to enhance the quality of the composition. A final model that does not only rely on the event rankings but does also consider the typical construction of a highlight summary is then established and used to create an automatically, well composed highlight video. The state of the art of such a framework is, that there are lots of video analysis and editing software available, but none of them are automatically cutting, or rating game events, as described in this approach.

Since the video is a broadcast, it already includes replays, commercials, and all other effects. The metadata text file contains timestamps of all common events during the game, like shots, saves, penalties, start and end of periods, goals and many others. Note that these metadata files are not created for this specific reason. They have already been created before as part of an online live ticker feed and are usually used to keep betting sites up to date at real time. That, on the other hand makes reading it out possibly more difficult, since it is optimized for different purposes.

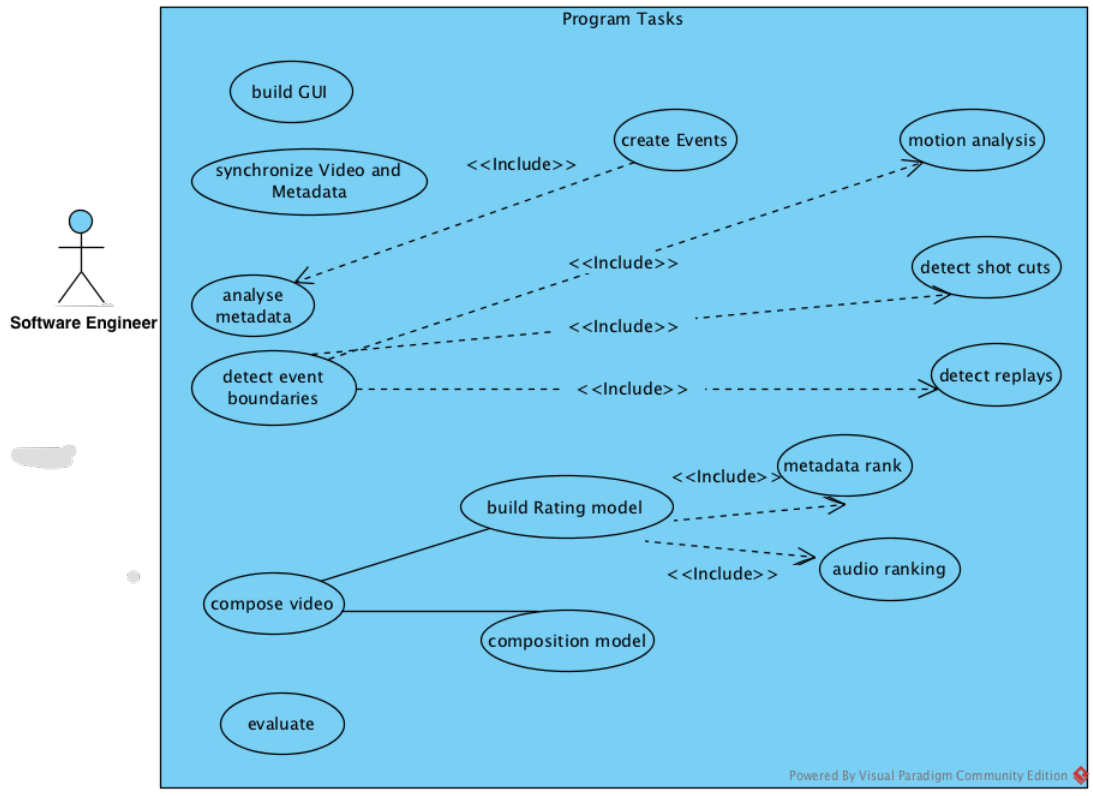


Fig. 3. All sub tasks, that are part of the framework

In order to rank the importance of plays and other events in a game, it will be necessary to create a model that defines all of the in-game events included in the meta data file. Therefore it has to be questioned what an actual in-game event is and how it can be described with certain characteristics. What kind of characteristics can be found (within multimedia retrieval features)? Here is a closer look at a simple in-game event:

A goal is scored. Metadata shows, that the home team scored it. We can expect the crowd to react louder than they would when the away team scores. So a solution would be to analyze the audio stream and make assumptions for the micro event in the model.

Questions that had to be solved for this work were among others: Can the created characteristics be used to completely describe the events, that are given in the metadata? Is it furthermore possible to make use of all micro events given and still create dominant criteria for each? What will happen if the metadata is not sufficient? The assignment of features onto micro events in case of missing data could then also rely on sport specific knowledge as well as data analysis and might eventually also be addressed by machine learning.

The next step does aim at finding micro event boundaries. Can we reliably detect event boundaries so that the composed video looks like being manually cut? There are enough of research results available on that topic. The challenge is to apply the methods on a system that is specialized for ice hockey broadcasts. Then the collected events have to be ranked. The focus here is the following question: How can one tell that an event is more important than others? Is it considered important when the crowd reacts enthusiastically? Also the flow / composition of the TV broadcast, may provide some indication of what has happened, and how important this may be, for example indicated by replays.

One way to extend that system over a knowledge based approach could be to create a generic configuration scheme, which can be used to weight the micro events by a certain criteria or even a combination of criteria, that is partly configurable by a users preferences and partly by the system administrators preferences. Those possibilities can be provided by simply moving the feature weights a bit, in order to steer the event ranking in the users favor, but could also be done by changing the video's composition, for example if an editor needs to create a special video for fan TV stations.

1.3 Goal of the Thesis

This thesis aims at the composition of highlight videos by modeling and rating events from sport broadcasts and its metadata.

The minimal requirement of the work should be a framework, that can at least cut unusable material off a TV broadcast and provide an idea about the most important scenes. The ultimate goal is to create a summary video, that is composed of a series of the top ranked highlights from the TV broadcast. The video should have a clear timeline and proper event boundaries and transitions.

To get the final result, a model needs to be defined, in order to compose such a video. This work's goal is not to research for new implementations, or new algorithms for event boundary detection, feature extraction or similar, but to make use and adapt the existing approaches, since those are already well researched. The goal is to realize a solution for the event boundary detection, that suits the needs of the given application and provides enough capabilities to realize the overall goal of the thesis. The focus lies on the interaction of all parts of the framework to realize a prototypical system that covers all the phases needed to deliver interesting video highlight summaries to a user and to create score calculation and composition models that can map the events to a users importance.

While the systems architecture is built to be adaptable and generic, the overall intention of the system is not to be generic for sports. The system and the developed models are intended to be very specific to the sport of ice hockey and use specific rules and occurrences that are typical for that sport, although the concepts and the system design should be based on generic concepts that allow configuration and adaption of the system behavior and could therefore be widely adapted.

As a minimal functional service the user can retrieve a sequence of ranked (important) events of the game. The focus however is set on the creation and composition of a full highlight video that is comparable to a manually prepared one with correct event boundaries and shot cuts. The Framework is generally configurable in certain parts by a user.

The Use Case Diagram in Figure 4 describes all possible usages of the project, that are part of this thesis:

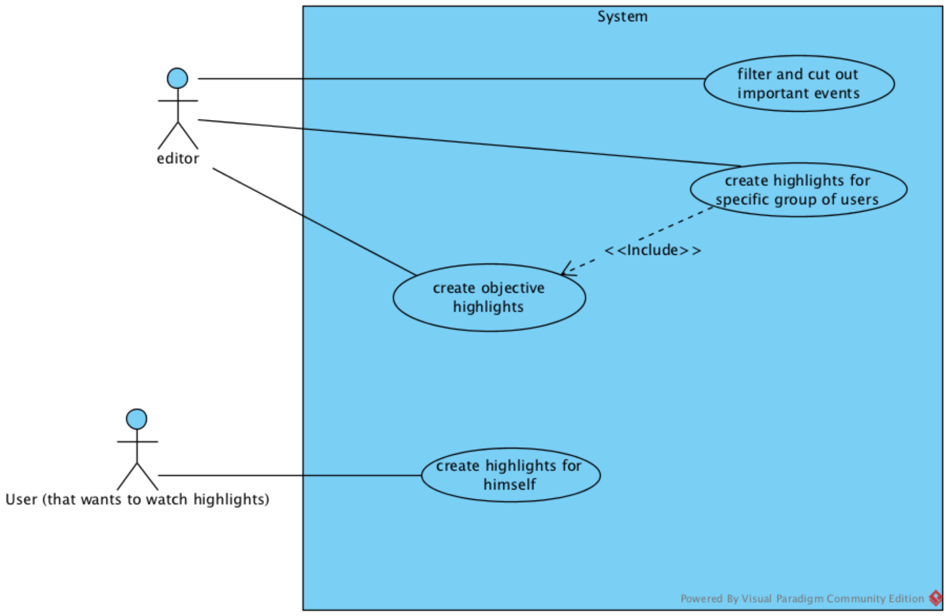

System
editor
filter and cut out important events
create highlights for specific group of users
<<Include>>
create objective highlights
User (that wants to watch highlights)
create highlights for himself
Powered By Visual Paradigm Community Edition

Fig. 4. Use Cases describing possible usages by different actors

Finally a simple User Interface should be available for a user to select the TV broadcast, the corresponding event information text file, as well as any additional information. A user could be a fan that is using the framework or an editor who is creating a video for users.

Lastly a brief experimental evaluation should provide some insight in technical and semantic quality. It needs to be checked if all technical requirements are fulfilled. Are the shot cuts, transitions and event boundaries in the final video set correctly? Are all highlights included? Is it possible for a user to verify which team scored, when they scored and what the score is? Some requirements can be checked manually. A small expert team from the video domain, working for Sportradar, will check the semantic quality of the highlights and rate the resulted highlight video. That evaluation is not intended to be a full quantitative research, but should clarify if the resulted highlight video is useable in practice as another way to watch a condensed version of a hockey game, or if it is maybe even comparable to a manual composed video. In addition the domain experts will give their subjective feedback as part of a qualitative analysis also.

1.4 Outline

The rest of this work will cover the following parts:

This part is followed by a section covering the state of the art and related works (2) as well as a broad background section 3 which provides a detailed look into the underlying principles of audio, video, XML and also ice hockey.

Next up is a system overview that describes shortly what tasks the framework is performing from start to end (4).

After that, the complete architecture and design of the program is shown (5), then the workflow of the framework is described in detail.

The following sections describe the actual implementation and the used algorithms, methods and concepts in detail. Thematically they are divided into the following:

- graphical user interface
- time synchronization
- meta data analysis
- rule sets
- shot cut and replay detection
- audio analysis
- video analysis
- automatic video cutting

Finally the result section (7.1) discusses the final result and the evaluation that was done. It is followed by a critical reflection in the conclusion and an outlook to possible future work and limitations (7).

At the very end the Appendix (A) provides a link to the resulting videos, an index of programs used and additional material.

2 Related Work

In the following, several papers that cover essential parts of this framework are mentioned. The section gives insights into work concerning the overall thematic and is followed by more detailed information on the various topics that are part of this thesis.

2.1 Overall thematic

There exist a lot of papers that cover certain parts of this work. However only a few of those combine and cover most of the essential parts all together. The most similar approach of these rare ones is probably [6]. They automatically synchronize a basketball game from the NBA with additional information that they found in online sources and use multimedia features to generate a ranking of the best baskets of the game. [57] implements video compositions of soccer games by using for example logo and replay detection as well as shot type classification. They specifically focus on replay generation and insertion and camera view selection.

There have also been works from bigger companies that implement such a system, that is similar to the one in this work, and make use of it in big sports events. "IBM" did it at the Golf Masters in Augusta in 2017 [1]. The system was specifically trying to detect high crowd excitement and commentary excitement, besides also creating a full framework to rank highlight scenes by a very sport specific ruleset. They had a similar approach at the Wimbledon tournament. Both of them did also connect their ranking models with additional metadata they collected. The difference to this work is that they did not yet use machine vision technologies as the goal was to only detect the events.

To produce highlights in different sports without taking metadata into account [60] consider video and audio features. They differentiate between fast paced and slow paced sequences and combine the output by analyzing volume and pitch of the audio signal.

A very interesting approach that is not related to the world of sports, was the attempt to fully automatically create a cognitive movie trailer from scratch [14]. This is extremely ambitious, as it attempts to complete a clearly creative task without human assistance. [55] was another project trying to do the same, utilizing all of the multimedia types, that were also used in this work, by analyzing audio and video, and by fetching metadata from movie databases.

An overview of state of the art techniques and methods in video summarization is given in [2]. It is very helpful if different methods and its advantages and disadvantages need to be compared (see Figure 5).

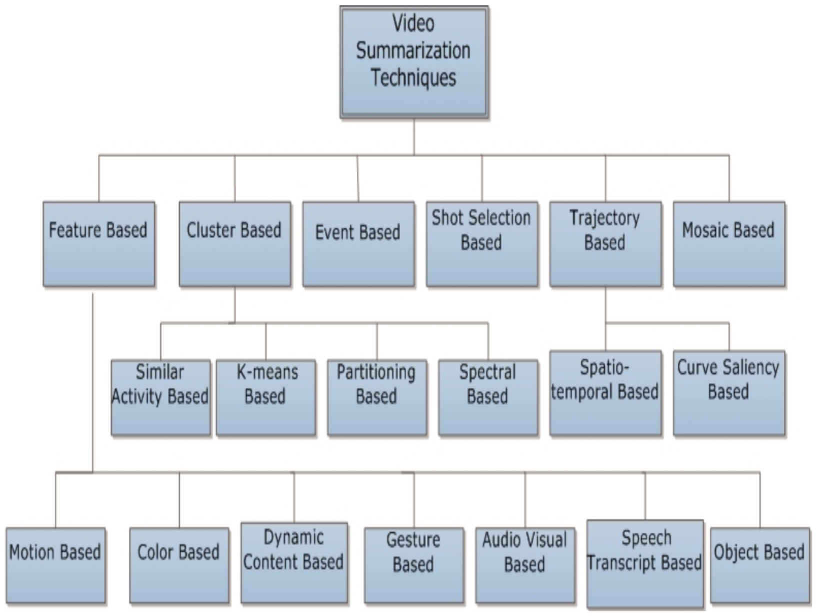


Fig. 5. Different methods for video summarization. Proposed by [2]

2.2 Live feed information

[62] and [61] capture live web casting text from the world wide web and align it with the broadcast. The downside compared to this work is that the web cast data does not provide as much information. They do not use audio features either. [62] also include domain specific knowledge and detect shot cuts as well as the start of the game.

Working with soccer and basketball broadcasts [64] try to detect semantic events in the videos without using calculation intensive techniques from audio and video features. They do use online webcasts, but do not have an accurate timestamp on the metadata text to synchronize with the video automatically. Instead they use shot cut detection to approximate the timestamp.

2.3 Audio Feature Extraction

Audio wise [26], [60], [45], [58] and [4] are proper examples of works, that use audio information in order to detect events.

[26] claim to be able to detect highlight events in a sport generic framework, only by using audio analysis. They point out that the energy of a signal is however not sufficient to differentiate important and unimportant voice. Their implementation uses a technique called "Piecewise Gaussian Modeling".

Pretty similar audio features as in this work, are proposed in [60]. Amongst others those are energy, zero crossings and Mel Frequency Coefficients.

[45] create highlight videos from baseball games. As the sport is rather slow, no live feed or metadata information is needed. Instead they rely on domain specific knowledge as well as generic sport knowledge in combination with lots of audio features including pitch, energy and MFCCs. They specifically conquer the problem of a noisy environment. Another example for highlight creation is introduced in [4], depending only on audio features. They work with rugby broadcasts and train classifiers to identify crowd noise, excited speech, as well as the referee's whistle.

2.4 Image and Video Feature Extraction

Using shot type detection of the broadcast to gather information about an event is the method proposed in [5]. It analyzes a sequence of shot types to gain knowledge if a certain event has happened. This is possible with domain specific knowledge about the cutting habits in certain sport broadcasts.

Methods with histogram comparison, which is used in this work, but also many other more sophisticated methods to detect shot boundaries are explained and compared in [33].

[32] and [25] set their focus on the detection of replay scenes. They show that logo detection and comparison can be a very robust and secure method of finding replay segments.

Adaptive user highlights are proposed by [18] by using slow motion methods and line detections.They do also consider domain specific rules that can be expected in given broadcast videos.

Complete highlight videos are created in [31] by converting a semantically annotated full video with the help of supervised learning from a training set and motion analysis. They propose an event model that consists of semantic relations, temporal relationships among activities and local motion.

[67] use video analysis and illustrate the importance of domain specific knowledge for each sport. They analyze and learn from the structure of specific sport broadcasts. In tennis, for example, they detected the ground lines and could therefore determine when a serve was about to happen, because sport specific broadcast structure usually shows that event from a special camera angle above the court.

Other algorithms for object tracking that could potentially be used in this work, are "Camshift" as described for example in [8] and "FAST", short for "Features from Accelerated Segment Test" [44]. It is one of the newest algorithms in that area and it is several times faster than other detectors. On the downside it is not robust to high levels of noise and it depends on a threshold.

[15] describes the usage and some advantages of the "Bhattacharyya" distance method over for example intersection. The "Bhattacharyya" distance is in this work used for histogram comparison. Other usages of the Bhattacharyya metric are shown in [24] and [52].

The calculation of the optical flow by using the "Lucas-Kanade" method is described in [41]. A very similar motion technique method is explained in [7] and [9]. The implementation of this algorithm in OpenCV does use "Shi-Tomasi" corner detection to detect possible objects edges [50]. It is an improvement to Harris and Stevens [27], or Moravec algorithm [39].

2.5 Composition

Finally [38] discuss shot and film cutting techniques in theory and practice. The book however, is not specialzied on sports. It is built around the basics of film making.

An overview of helpful features for the classification of sports is given in [17]. It claims that cut rates do specifically correlate with excitement in sport videos.

[66], [40] and [5] describe different solutions of weighting the implemented features. [66] weight audio and visual features separately using different weights. They then use user relevance feedback to adapt the ranking to the needs of a special user.

[40] create a video summary of soccer games based on sensor information. They capture for example heart rate and exact the positions of every player on the pitch. A small group of experts can then query for special events like scenes inside the penalty box. This method can provide immense knowledge if well written queries are used. However, without further textual event information, or audio and video features, there is still a limitation.

Many experiments use machine learning to approach the challenge of choosing events and composing the video. [63] is one of those. As most of those approaches are supervised, labeled data is needed however.

A combination of web live feed information and video analysis is proposed in [62], but more importantly, they finally also discuss scenarios of how to deploy the proposed solution on various devices.

3 Background

This section provides an overview of basic knowledge that is needed in order to fully understand the implemented algorithms and used methods in this work.

3.1 Image Colorspaces and Histograms

Generally there are two main ways of color representation: Additive and Subtractive. In the additive representation you start with black color, which actually represents no color at all and then mix in other colors to eventually get the desired one. In subtractive color spaces the starting point is the color white which is the composition of all colors. Then you filter out color parts in order to design a specific color.

As one can probably guess, subtractive coloring is used for example when a printer writes on paper, where as additive coloring is used on computer monitors [34]. It dates back to the 19 th century that the Young-Helmholtz-Theory was created. Helmholtz and Young found out that it is possible to mix any color from three primary colors and that the human must have receptors in the eye for all three of them. They did also already believe that those receptors do respond to red, green and violet [47] [19].

Nowadays the standard for computer monitors is to create colors in the "RGB" color space. As mentioned, it is additiv and it is mixing up the colors red, green and blue, in an attempt to reproduce the human color perception. Each pixel on the computer does have three values. One for each of these color channels. The amount of colors possible to produce does then rely on the amount of values available in each channel. All combinations form a different color. However, the "RGB" color space is pretty simple and there are problems with human perception for example regarding light intensity.

In this work histograms are used to compare images with each other, which is one of the simplest, but very reliable methods [23]. Histograms show the frequency distribution of scaled characteristics. In our case we put each color in one bin and then count the occurrences of that color. We can assume that big changes in color over the whole picture only happen when a shot cut is done. Because if objects are moving in the image, no matter if the camera moves or the actual object, the color frequencies will most likely stay almost the same.

In order to be able to put the pixel values into the right bins, it is better to change the color space, because in RGB color space each color is a combination of the three values which gives millions of combinations. This is difficult to allocate into a small set of histogram bins. That is why we change into the HSL color space. This is short for hue, saturation and lightness. Hence it has only one channel for the actual color (hue) and can therefore easily assign the values to the right color bin of the histogram. The other two values only specify the color further. This color space is actually better aligned to human perception than the RGB space [54], [35].

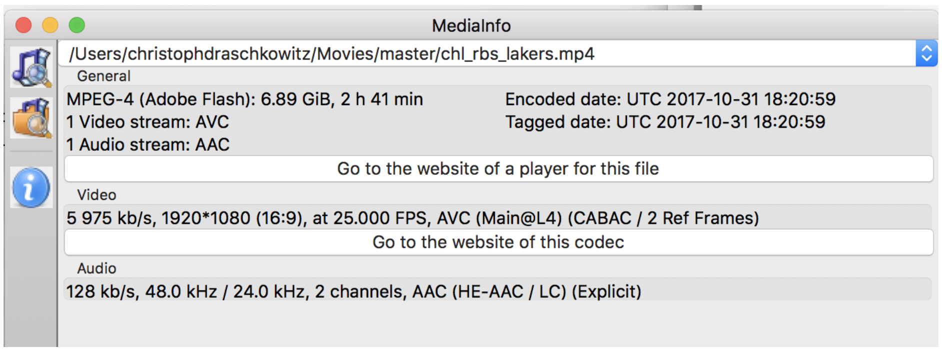
The last step here is to compare the histograms by some measure. OpenCV offers four different metrics to do that: Correlation, Chi-Square, Intersection and Bhattacharyya distance. For various reasons explained in [10] and [11], all of them are of course better than summing up the "bin by bin" differences.

The Intersection method takes the sum of the minimums of the pictures, which results in higher numbers on good matches. Intersecting histograms still has a disadvantage as it does not consider distances between the bins, which gets really important with ordinal data for example. However, that is not the case here. Another disadvantage is that it is not bounded, so that its values have no finite range, which could get a little bit problematic here. In this work Bhattacharyya distance is used because it worked best in the tests. [15] describes the usage and some advantages of that method over for example histogram intersection. The Bhattacharyya distance does also return zero for a perfect match. The higher the number gets, the more different the images are. It is calculated from the Bhattacharyya coefficient which is a measure of the amount of overlap between two statistical samples. Other usages are shown in [24] and [52].

3.2 Video

Let us start this section with the basics of human visual perception. In order for the human eye to create the illusion of an evenly moving motion sequence, it takes twenty four pictures in a second. Another parameter that plays an important role is the brightness. However, to completely prevent the human eye from perceiving flickering when images change, the frequency has to be as high as fifty Hertz [46]. That is why the common television methods PAL and NTSC use fifty or more half pictures in a second. Each half picture is shown over every second row. The PAL method is used all over Europe as well as Australia and parts of Asia, South America and Africa, where as NTSC is mostly used in North America. Technically PAL was developed to correct coloring errors in NTSC, but those two standards do also differ in frequency. PAL uses twenty five pictures and fifty half pictures and NTSC thirty pictures and sixty half pictures. Most of the cinema movies on the other hand, are created by twenty four pictures only [21]. The videos provided for this work consist of twenty five pictures in a second.

The video is provided in the MP4 container format which is a MPEG standard specification with only one official filename extension ".mp4". Nevertheless it can contain any kind of media starting from audio and video, to subtitles, still images and of course metadata. The actual video metadata can be seen in Figure 6. It is extracted by the software "mediainfo", which also provides a tool that is implemented and used within the framework.


MediaInfo
Go to the website of a player for this file
Go to the website of this codec

```
/Users/christophdraschkowitz/Movies/master/chl_rbs_lakers.mp4
General
MPEG-4 (Adobe Flash): 6.89 GiB, 2 h 41 min          Encoded date: UTC 2017-10-31 18:20:59
1 Video stream: AVC                                 Tagged date: UTC 2017-10-31 18:20:59
1 Audio stream: AAC
Video
5 975 kb/s, 1920*1080 (16:9), at 25.000 FPS, AVC (Main@L4) (CABAC / 2 Ref Frames)
Audio
128 kb/s, 48.0 kHz / 24.0 kHz, 2 channels, AAC (HE-AAC / LC) (Explicit)
```

Fig. 6. Media Info of the video input file. Captured with the program MediaInfo

As one can see the actual video compression technique used here is AVC. The full name is H.264/MPEG-4 AVC and it is a very commonly used standard for video compression using features like motion compensation. Motion compensation methods are used for video compression by trying to predict motions in the movie to be able to only save very few images entirely and construct the rest by parts of all other images [53]. The goal of this standard was to create a video compression standard that provides decent quality at lower bit rates without increasing complexity.

In this work motion analysis is done in order to find out the general direction of play in the game at a specific moment (which is the beginning of an event). At this stage the event start boundary is defined. The scientific term used for that is optical flow. More scientifically spoken it can be described as the

> .. estimation of velocity of image structures from one frame to another frame in time sequence of 2-D image. ...the 2D motion results from the projection of moving 3D objects on the image plane. The 2D projection is called apparent motion or optical flow. [41]

In order to get an idea of the direction of play, it is obviously necessary to find data clouds that can be tracked in the video. This is of course closely related to object tracking. However, in this case it is obviously neither important what kind of object is tracked, nor is it important that it is an actual object, as long as it is possible to track its movement reliably.

The basic idea to detect good features for tracking is to split an image into small windows. The next step can be described as having a piece of a puzzle and finding the right spot for it. If one of these windows contains just one object it is hard to locate its correct position in the full image. If it contains edges that divide the window it is easier to find, but still there are many other places where this edge could belong to [51]. So the easiest way to find a correct spot is a piece that contains a corner. Up to that we can expect large image variations in the neighborhood of corners, in all directions. That leaves us searching for corners to detect, because these parts of the image can be easier found in the following pictures. These pieces of corners will then be searched for around the area they were found in the last image. If this is done picture by picture, it is possible to track an object and even estimate an objects "flow" [27] [28].

The OpenCV library that is used in this work, provides an algorithm that can be used for all those steps. Optical flow is usually often used for video stabilization and compression. The estimation of the flow in this algorithm is done using the "Lucas-Kanade" method. This method solves the basic optical flow equations by the least squares criterion. [41], where the least squares criterion searches for a function that is closest to all points of data.

The implementation of this algorithm in OpenCV does then use "Shi-Tomasi" corner detection to identify possible objects edges [50].

3.3 Audio

The acoustic pressure of a sound wave can be measured in decibel (dB). However, a human person differs sound waves also by its frequencies (high and low voices). Furthermore, the loudness for a human ear is neither linear to the acoustic pressure of the sound wave, nor is the acoustic perceived pitch linear to the frequency.

Phon is a unit that measures loudness, while trying to compensate the effects of the frequency of that tone. "Sone" is a linear scale that describes how loud a noise is perceived by the human ear. If a sound is measured as one Sone, then a sound that is subjectively twice as loud, has two Sone [42].

The Mel Scale on the other hand is a unit describing the perceived tone pitch. Again it is linear to the perceived pitch but compared with frequency it is almost logarithmic (see Figure 7).

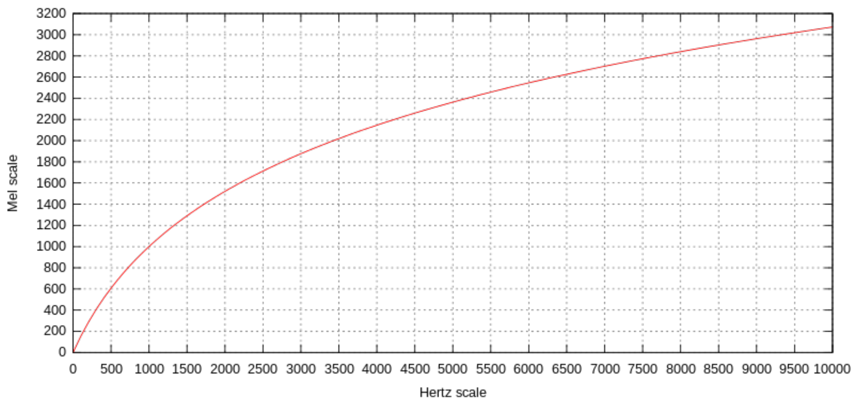


Fig. 7. Picture Author: Krishna Vedala; GFDL License: Approximation of the Mel Scale. Just like with Sone and Phon, the relation of human perception and Mel is not linear

These principles are important to keep in mind in order to understand why it is not sufficient to analyze an audio sample only by its loudness or frequency.

For that reason in this work several different audio features are used. Some of them consider those non linearities, while others do not. The methods used in this work that do not sufficiently consider this non linearities in their basic implementations, are energy and zero crossing rate.

As the energy measures the amplitude of the signal, it is a measure for loudness, whereas zero crossing rate is a measurement for frequency as it counts all events, whenever the signal crosses the zero value of the amplitude.

It is especially used in speech detection, since speech has a low amount of zero crossings while crowd noise has a lot of zero crossings [49], [3]. Zero Crossings are usually implemented as a rate value over some specific time, or as a total count. In this work it does not matter which is chosen, because the audio samples do always have the same length.

A feature called "Mel Frequency Coefficients" does however take the before mentioned principles into account. This is reached by performing several steps on the audio signal:

It starts with a very well known technique in audio analysis. By the help of Fourier Transformation the audio signal gets decomposed from time to frequency domain. In this representation, frequencies, that are used more than others, can be detected. The Fourier Transformation can be used, because as Fourier Analysis states, general functions can be approximated by sums of simpler trigonometric functions.

After applying the Fourier Transformation, further steps map the powers of the resulting spectrum onto the Mel scale. Then the logs of the powers at each Mel frequency are calculated and the discrete cosine transform is performed on them. Finally the Mel Frequency Cepstral Coefficients, that collectively make up the MFC, are the amplitudes of the resulting spectrum [36], [65].

The videos provided for this work are, as already mentioned, provided in the MPEG 4 container format. The audio is compressed with "AAC", an MPEG4 standard, which stands short for advanced audio coding and is shown in detail in Figure 6. It is a lossy audio compression method and it is the standard for audio on youtube and many other well known platforms. In order to realize the compression, the algorithm follows two different strategies:

Firstly it finds and eliminates all redundancies found in the audio signal. Secondly signal components, that are perceptually irrelevant to the human ear, are eliminated. This follows from the fact that humans can only hear sound in frequencies between approximately twenty and twenty thousand Hertz, where only kids can hear up to that maximum. Therefore one step is to eliminate all sounds outside a certain frequency range.

3.4 Markup Language

The metadata text file, that is used for this work, can be described as "Descriptive Markup Language", just like "Latex", that this document is written in.

The goal of those languages is to annotate text inside a document, while still keeping it distinguishable from the actual text. Text is therefore encapsulated inside "tags". The advantage of the mentioned languages is that they are standardized. Hence, all structures are well defined. A main advantage of structuring such text files is that it keeps the file human and machine readable. The meta data files in this work are actually XML files. Still they do not contain a version declaration, which was optionally allowed until version 1.1. Being a standardized language also makes it easy to automatically read out machine readable content as done in this work. However a specialized XML schema with well documented and structured tags for the creation of the metadata file would be desirable. That way the error rate of the program while reading out the text file could be minimized and maintenance of both the software, that exports the meta data file, as well as the software that uses it, would be much easy.

3.5 Python

Python [2] is an interpreted high level programming language that was first released in 1991. Interpreters are available in all common operating systems and its reference implementation "CPython" is open source software. It is known for good code readability, for example through forced indentation, and supports object orientated, as well as functional and procedural programming. It has a built

[2] http://www.python.org/

in memory management and a widespread, well documented standard library. Up to that it offers immense support and a variety of free external libraries.

No compilation is necessary to run the code. This of course makes it, together with dynamic typing, a slower language, than for example Java or C++. Besides that dynamic typing leads to more runtime errors.

On the other side of the medal it also has various advantages and was chosen for this project for following reasons:

It is well established as a scientific programming language, also used by people that are non programming experts, for example due to the "Numpy" library, that brings "Matlab" like behaviors and of course the easy readability and dynamic typing. Therefore an incredible high number of well documented and well supported open source libraries are available. It does run on most platforms and no compilation is needed. Memory management is included. Since performance is not a higher priority for this work, controllable memory management, that other programming languages like "C++"support, is not necessarily needed. All together Python can be considered a great programming language for scientific prototyping. A professional implementation would on the other hand be better off with other languages like "C++".

3.6 Libraries

3.6.1 Computer Vision by OpenCV Computer Vision describes tasks that are solved by computers, but can usually be done by the human eye. In this work this includes shot cut detection, background detection, motion analysis and object tracking. [51] and [23] provide an overview of existing techniques and methods.

The library used in this work is OpenCV [3], the short term for "Open Computer Vision" and as those words would suggest, it is an open source library for computer vision tasks. It is a de facto standard library in its domain, was first published in 2006 and originally written in C.

Nowadays it offers wrappers for Java and Python as well. To this day there is still further development in its algorithms. Provided functions stretch from easy image retrieval tasks like returning pixel values, to heavy tasks like basic machine learning algorithms and face detection [43].

[3] http://www.opencv.org/

3.6.2 FFMPEG FFMPEG is the name of the whole free software project that produces software for handling multimedia data and was started in 2000. The name is inspired by the MPEG group with the double "f" for fast forward. FFMPEG itself is constructed as a command line tool that can convert audio and video formats. However many other well known applications make use of FFMPEGs libraries and codecs (VLC Player, Google Chrome, Youtube). It does compile and run in all major environments and architectures. It can be used to

> decode, encode, transcode, mux, demux, stream, filter and play [4]

almost any format. The "moviepy" library does for example also use part of FFMPEGs implementations.

3.6.3 MoviePy Moviepy [5] is a Python library for video editing. It can handle almost any format. It uses parts of "ffmpeg" and also depends on other Python libraries like "Imageio" and "Numpy".

For this work it is essential for extracting and concatenating video clips, as well as some basic manipulation, like text overlays and simple analysis. It is also powerful enough to be helpful for some tasks done in the audio analysis of this work.

3.6.4 LibRosa LibRosa [6] is an audio and music processing library in Python and is exceptionally well documented.

It is used for most of the work done in the audio part of this thesis: It is used for all audio analyzations and to create audio fades at the beginning and at the end of each clip of the final video.

3.6.5 Tkinter Tk is a free and open-source graphical user interface toolkit that was already released in 1991. It was initially written with the high programming language "TCL", which is usually embedded in "C" applications. Created for Unix it was however designed to be extended and is available across all platforms nowadays.

[4] http://www.ffmpeg.org/
[5] https://zulko.github.io/moviepy//
[6] https://librosa.github.io/

It offers all widgets commonly used in developing desktop applications. Tkinter is the standard Python binding to the Tk toolkit and generally the de facto standard GUI in Python. It is short for Tk interface. Just like Tk, Tkinter is free software. Architecture and usage are described for example in [20].

3.7 Ice Hockey

This section provides a quick overview about the basic rules in ice hockey as well as a couple of more specific rules that are of significance for this work. In terms of rules, most professional hockey leagues follow the IIHF (International Ice Hockey Federation) rulebook [22]. A lot of professional leagues nowadays create their own rulebook, still following the IIHF rules, but specifying some of them differently.

As the name suggests, ice hockey is played on ice. All players wear skates, a well defined set of protective clothing and a stick. The goal is to get a circular hard rubber puck into the opponents net. It is allowed to move the puck with any part of the body. However to score a goal, only the stick can be used actively. The dimensions of the playing field vary a little bit, but are approximately 60 times 30 meters and it is surrounded by boards.

A full team consists of twenty players and two goalies, nonetheless only five players of each team face each other on the ice, with one additional player guarding the net. Goaltenders wear special equipment and shall not be attacked by field players. It is basically allowed to switch players at any given time with no regulation on the amount of changes. Extra players have a seat on a bench right behind the playing field. The objective is to put the puck into the opponents net. After a playing time of three times twenty minutes netto (time stops on intermissions), with intermission between the periods being as long as fifteen to seventeen minutes, the winner is the team that scored more goals. Depending on specific league rules, there might be an additional overtime and/or penalty shootout if there is no winner after sixty minutes.

After goals and certain other events the play gets whistled of. Each intermission is continued by a "faceoff", where one of the referees drops the puck. Most leagues specify four official referees that guide the game. If a penalty is awarded to one of the players, he has to sit out for at least 2 minutes (depending on the type of penalty) and can not be replaced by one of his teammates for the given time.

Anyhow, in special situations the referee can also reward a penalty shot to the fouled player, instead of sitting the offender out. If a goal is scored and the referee is not sure whether all rules were followed prior the goal, he can request a video review. He then gets to see the replay of the scene and decides if his calls stands or if a different call is appropriate.

A team is also allowed to play with a field player instead of the goalkeeper. That is usually used at the end of the game, by a team that is trailing a goal, to create a man advantage upon the field players. This is called an empty net situation since there is nobody left to guard the net.

Games analyzed in this work are commonly from the CHL (Champions Hockey League). Teams from all over Europe can qualify through their respective national championships for the following year, similar to the well known "UEFA Champions League" in soccer. It is played as a tournament: After a group stage, the best teams qualify for the knock out rounds. In this stage two teams face each other in two legs and the score gets aggregated. If the score after both games is even, the game gets to a ten minute sudden death four on four overtime, meaning that the team, that scores the next goal, wins and that there are only 4 players plus the goalie on the ice for each team. If however the overtime ends scoreless the game is decided in a penalty shootout [13].

4 Approach

This section provides a simple overview of the whole framework and the approach taken to conquer the problem. All parts mentioned, are further explained in detail in other sections coming up.

The initial testing during deployment was done with four different ice hockey broadcast videos (and the corresponding metadata files). All of them were recorded on the 31 th of October 2017 and part of the 2017/18 Champions Hockey League (CHL) season. The single matches were:

Brynas against Mannheim with a final score of 3:1, Bern against Munich with a score of 2:3, Salzburg against Växjö ended 2:1 and Brno against Zug, which ended 4:3. The first mentioned team is always the home team.

The set of videos was chosen in a way that as many game situations as possible were covered. Obviously there are high scoring, as well as low scoring games included. Also the number of people watching the games at the rink and therefore their response to in-game situations differs a lot from game to game and is further discussed in the following sections.

Results were also tested with other videos of that CHL season and from the german national league (DEL), as it should not make a difference to the program, as long as the basic ice hockey rules [22] are followed, the corresponding metadata file is provided and information in it is correct.

In order to conquer the described problem, various tasks have to be solved. A simplistic, basic work flow, that describes the relation between the most important tasks of the framework and the models and external resources they need to have access on, can be seen in Figure 8.

To be able to read in the correct input files, a graphical user interface, written in "Tkinter", will be implemented, to let the user choose a broadcast video file and the corresponding meta data file. The meta data file is a semi structured XML file with all event information about the game and is further explained in the section metadata. In this early part of the framework, the event objects get created already. Those objects will later be adapted by adding extra information from further analysis. The events also get a first score calculated, depending on their type, their timestamps, or relative position to other events.

In the next stage only certain events get further analyzed. The audio information around those events gets extracted and several features calculated from it. A combination of all those features provides the base for the final event scores. This ranking depends on a different rule set, that is based on information from the metadata. Shots on goal in the last couple of minutes of a game are for example more important than somewhere else in the game.

Last step in the process to choose the best events for the final video, is to follow a composition rule set model. It will for example consider how many goals were scored and adapt the amount of other scenes on that information.

Now that the events are chosen, in the next steps, the perfect event boundaries to fit the events into the final video, get calculated. Therefore shot cut detection algorithm, replay detection and motion analysis are implemented. The composition model still has an impact here.This rule set could also consider shot types of a video or the target highlight video time, in order to for example shorten the time of a teams celebration, shown in the final video.

Once the event boundaries are known, the clips can get extracted. An audio fade in and fade out is laid over all of them. Video fades, that are manually defined before the framework gets started, are put between the events and finally the video gets exported as a "mp4." file. In an extra monitoring mode text overlays are written over each event video clip and show the criteria why they were chosen.

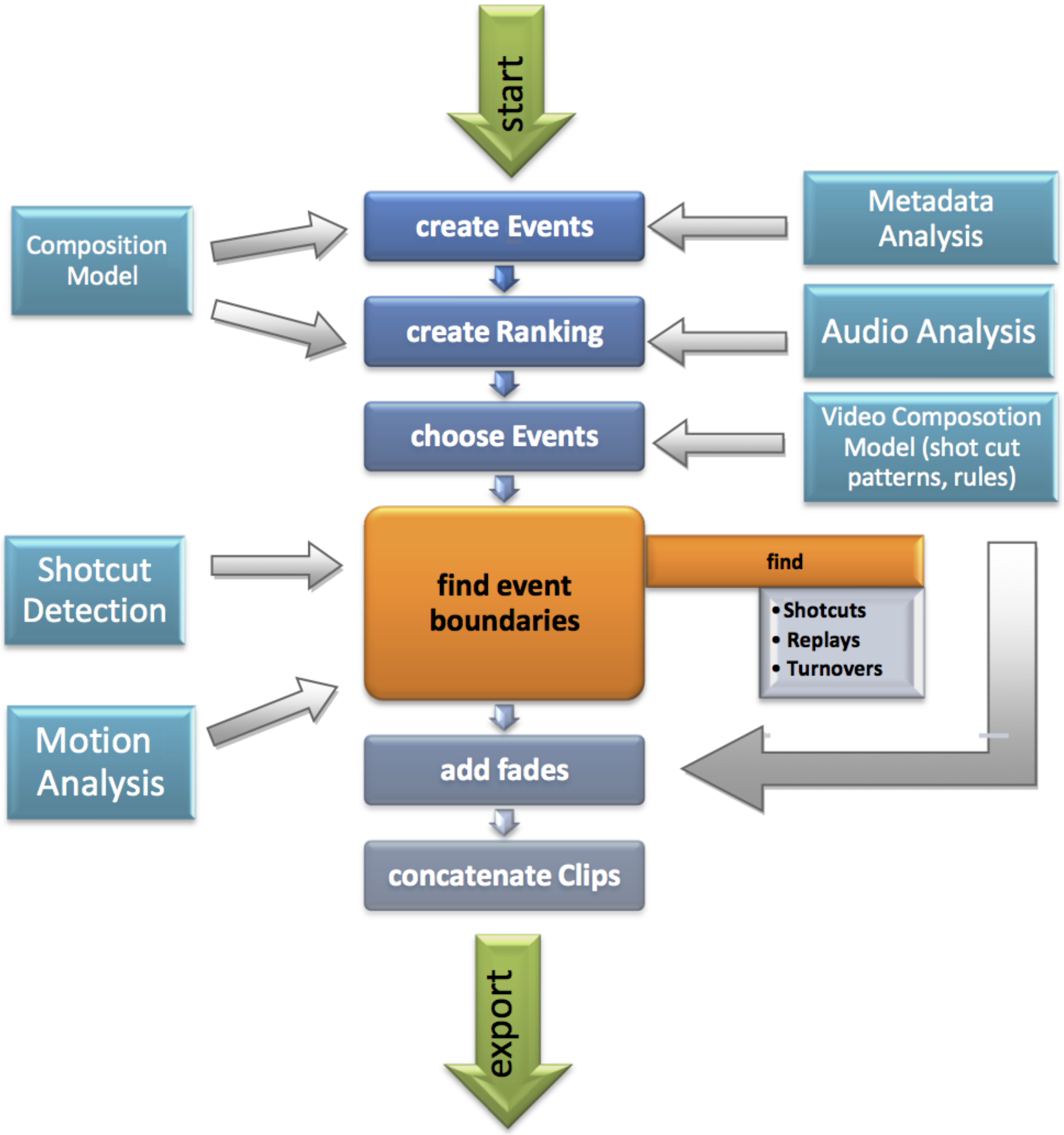


Fig. 8. Simplistic task flow showing the relation between the most important components and external resources and models

5 Overall Design, Workflow, and Implementation Strategies

This section covers the overall design of the framework, that was implemented to solve the problem. It provides detailed insights to the workflow of the framework and describes strategies and concepts that influenced the architecture of this works implementation.

For the implemented framework it was determined, that a non computer expert should be able to execute a program that produces the desired highlight video. Hence the program is supposed to be running all at once and not be divided in various executable parts.

Therefore the decision, to rely on "Python" as a programming language for the whole program, was made. The benefits are the easy usage and the widely available support for scientific libraries. It is considered perfect for scientific research as so many extra libraries are freely available. For example the fully applied wrapper for OpenCV comes in very handy, as it is the de-facto standard open source libraries in its particular domain and was originally only provided for C and C++.

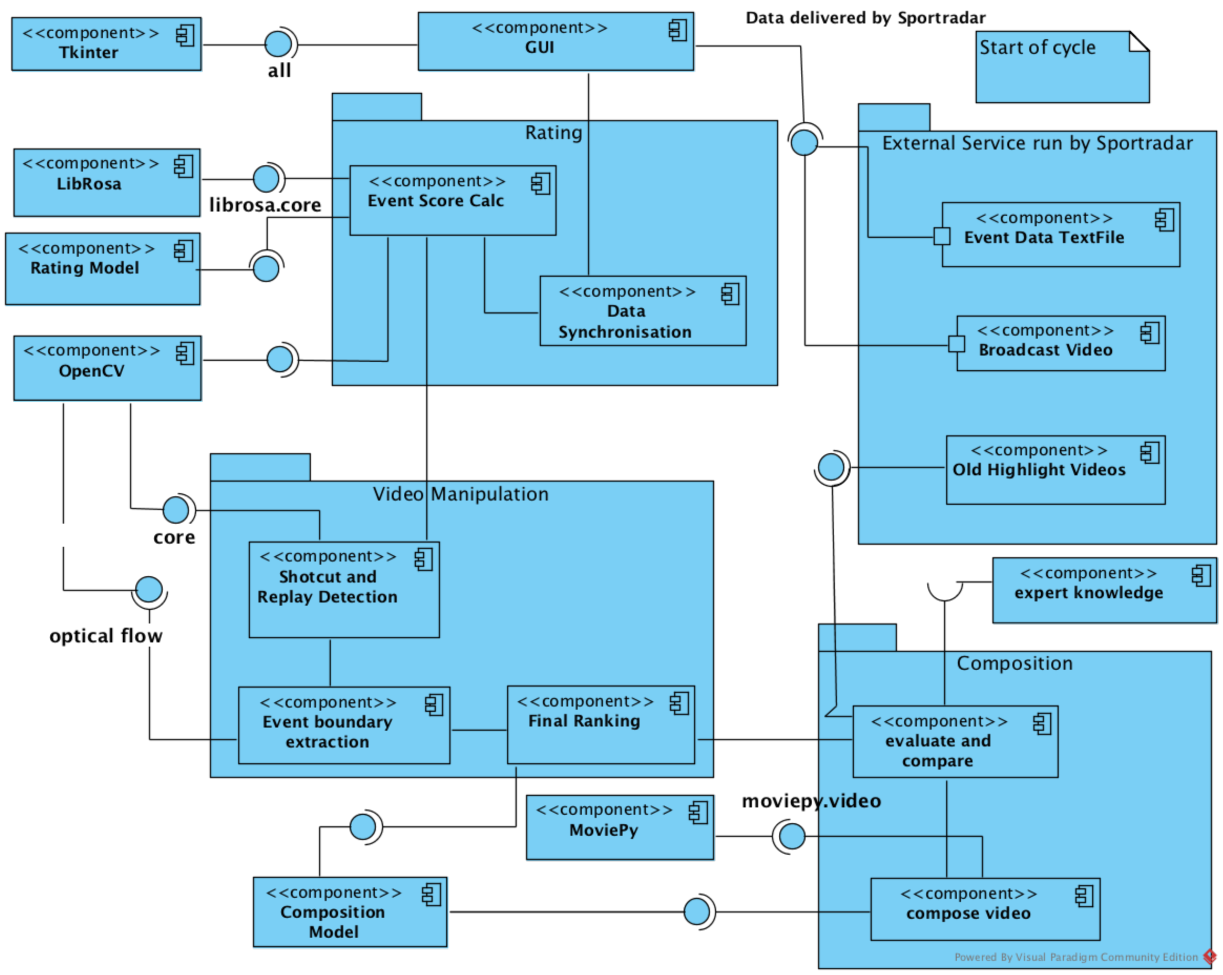
Furthermore it makes sense to provide the user a graphical possibility to specify the input files (which are the meta data file and the video file). Of course performance wise other programming languages would be more favorable, but as stated in previous chapters, the focus of this work does not include performance issues at a higher priority. Therefore the ability to implement the whole framework as one standalone program has a higher value for this work. However, as the broadcasted videos are composed of 25 frames a second and full match videos having a duration of over two and a half hours, of course there is the need to use resources responsibly, which again has a direct impact on the systems workflow.

The component diagram shown in Figure 9 provides a break down of the programs components and additionally used libraries.

The package on the upper right is all external and already provided. It provides video broadcasts of the hockey games and the corresponding event metadata. Additional metadata can later be fetched from online sources.

The rest of the framework is split up into three major components that handle video and metadata synchronization, as well as boundary detection, the rating process and the composition process.

A more detailed description of the tasks of each python module can be found in the appendix (A).



<<component>> Tkinter
all
<<component>> GUI
Data delivered by Sportradar
Start of cycle
Rating
<<component>> LibRosa
librosa.core
<<component>> Event Score Calc
<<component>> Rating Model
<<component>> Data Synchronisation
<<component>> OpenCV
External Service run by Sportradar
<<component>> Event Data TextFile
<<component>> Broadcast Video
<<component>> Old Highlight Videos
core
Video Manipulation
<<component>> Shotcut and Replay Detection
optical flow
<<component>> Event boundary extraction
<<component>> Final Ranking
<<component>> expert knowledge
Composition
<<component>> evaluate and compare
<<component>> MoviePy
moviepy.video
<<component>> Composition Model
<<component>> compose video
Powered By Visual Paradigm Community Edition

Fig. 9. Component Diagram of the framework. The package on the upper right is external and not part of the implementation of this work

In the rest of this section, the workflow that was already simplistically described in Figure 2, is analyzed more detailed.

At the beginning of the programs execution a graphical user interface gets called for the user to be able to choose the video as well as the metadata file that he wants to get analyzed.

After that, the OpenCV library for computer visual tasks is used for the first time. It checks if the video file is valid and furthermore extracts basic video data like frame rates and image dimensions. Another part of the program checks the event metadata file for validity. Then it extracts the events and creates objects out of it.

The sequence diagram in Figure 11 draws a simple picture of used classes and their sequence of execution.

Depending on the exact structure and information in the XML file, the structure of the event classes created from that file differs by a small margin.

Next the program starts to handle the synchronization of the video and metadata file. The metadata file has an Unix timestamp included. Some of the original broadcast videos provided for this work where encoded as a ".mxf" file and did also have a timestamp of its creation in its metadata. It did fit with the start time of the metadata file and the start time of the actual hockey game, so it can be assumed that the video file was created live at the respective game. However, once the video is ported to a more commonly used filetype, this information gets lost. Other broadcasts already came as handy ".mp4" format files, that are also compressed decently.

If video or metadata do not provide correct timestamps, synchronizing them gets incredibly time consuming, since it would have to happen through video analysis algorithms. [6] uses such methods for that matter. Also other visual hints would be a possibility. This issue is covered in section 6.2 Time Synchronisation more detailed.

In order to save computing time the next step is to analyze the collected events, to be able to extract as few video and audio parts as possibly needed from the video file, since that is more time consuming. To do that, event types are queried and the program awards scores to each of them. These scores are at that stage only depending on metadata information. Most important is the event type in combination with other event information like the match time and the score line when the event occurs.

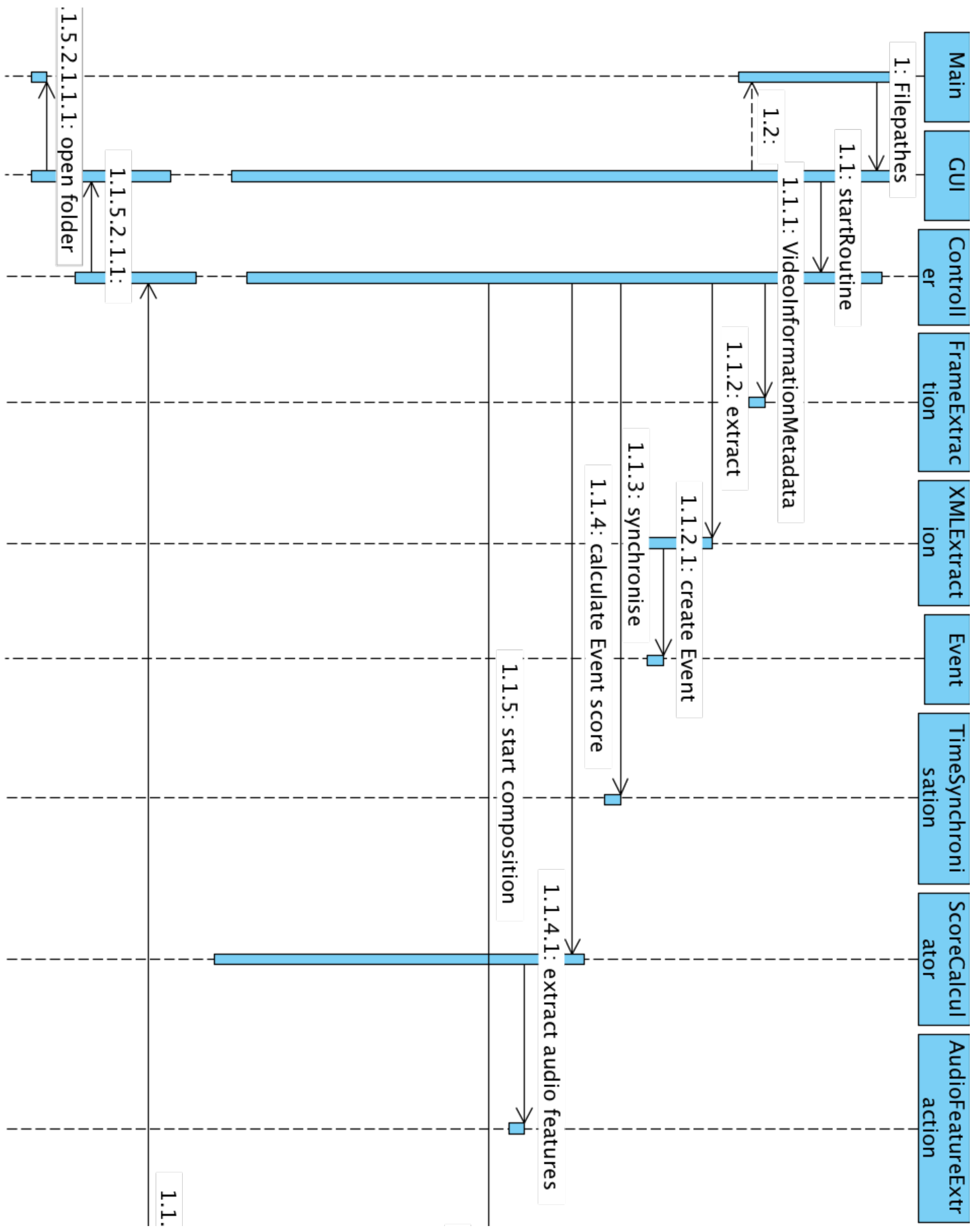


Fig. 10. Detailed Sequence Diagram of the workflow of the framework that creates video summaries from ice hockey games

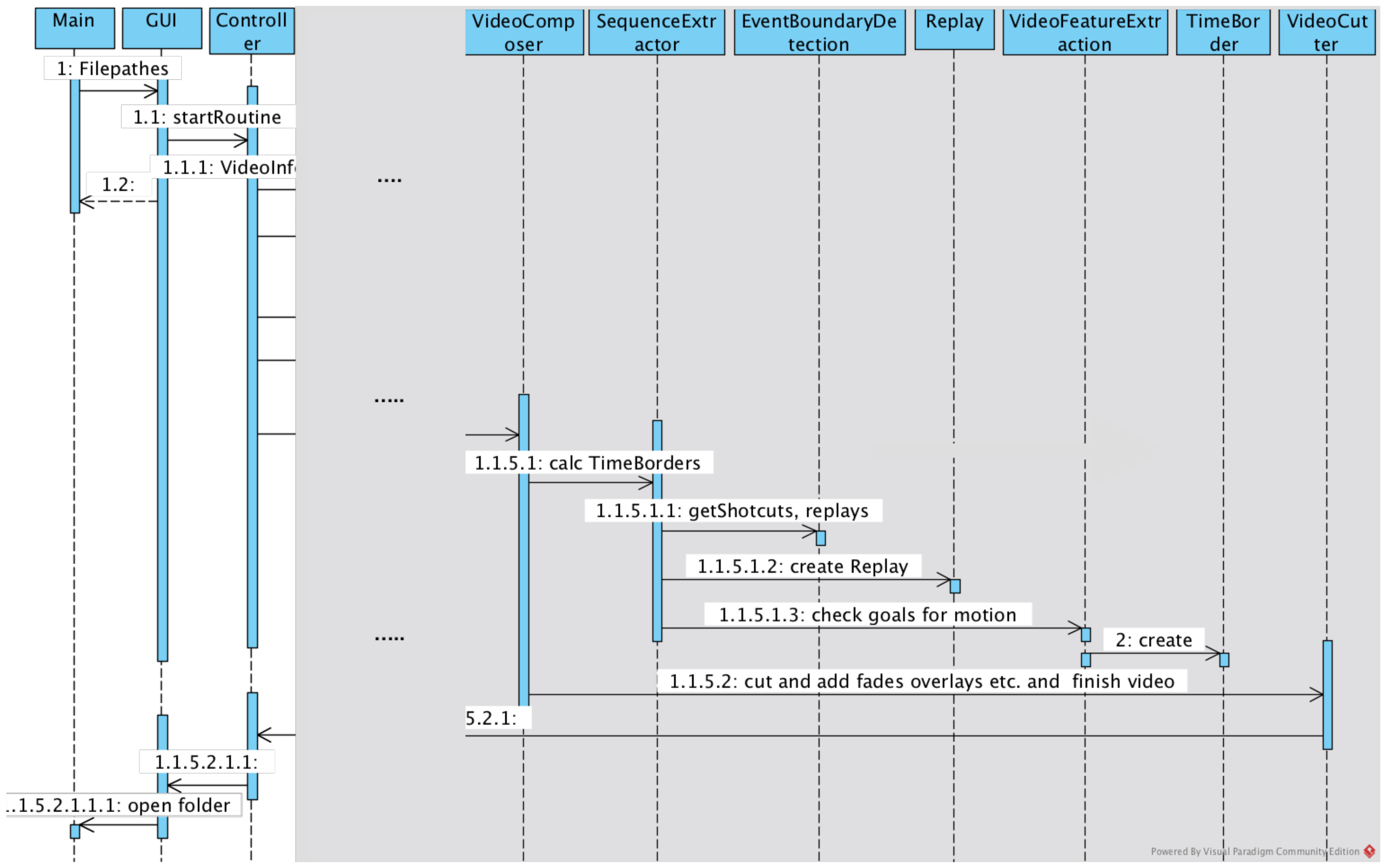


Fig. 11. Detailed Sequence Diagram of the workflow of the whole framework

A normal penalty, for example, that gets called during the game, is usually not worth showing in a highlight video. If the other team, however scores on the resulting power play, it is. For the given XML files it was necessary to loop and search for certain events when others occurred. A "Betstop: possible goal" info for example, would indicate the exact time the goal was scored, but is not yet confirmed. One of the next couple of events would confirm or revise the goal. The score however, has to be given to the first event in order to proceed with the correct timestamp. Of course goals get the highest score here, but also uncertain events, that need to be validated by other techniques, receive a decent score in these steps. All events that do not receive a score will not play a part in the rest of the workflow and are therefore eliminated.

In the next step, certain events like a shot on goal, need to be further processed. An audio analysis of that part of the video determines wether the score moves up or down by a margin. Other events get processed by querying extra data from online sources.

Although not yet included, video features could be used in that step too. As video feature extraction is extremely time consuming, it has to be used carefully. As there is usually fifty to eighty shots on goal in an average hockey game, it is hardly possible to analyze each shot on goal by video features, if the program is supposed to finish in an acceptable time. Now all scores are set.

Depending on time restrictions of the final highlight video, an algorithm chooses the events for the final video. Before those get extracted, there has to be determined how long each event will be shown. This does not only depend on the event score, but does also consider shot cuts, replays and motion analysis:

Before program start, the standard cut time for each event type is preassigned. People would want to see the whole origin of a goal for instance, where as a video review, where the referees watch the replay on tv screens, does not need any special consideration and should be shorter. This gets preassigned by the type of event. For the composition of the final video it is important to follow some basic rules in order to make the video look good for the human eye and in order to not confuse the user.

One of these rules is for example, that there shall not exist extremely short shots to avoid confusion. In order to take care of that, the program looks at the mentioned preassigned time borders of each event and watches for shot cuts near them. If one is detected the time border gets adapted to match that shot cut.

At some events, like goals, it is however not easy to define a generic border, since people want to see most part of the attack, that leads to that goal. This work solves this exact problem by tracking motion parameters before each goal. Once a change towards the net, where the goal is going to be scored at, appears, the new attack starts and the motion tracking algorithm notices it.

With the addition of the event's replays, the final event clip time is then completed. Again, depending on the type of event, a certain amount of replays gets added to the event time borders. There are multiple ways to detect replays within a video. In many videos, including the ones used for this work, replays where always introduced by a very short logo fade and thus could be detected by rather simple background comparison techniques.

The very last step is the actual extraction of the finally used events and their recomposition to the finished product: There are video fades provided at the beginning and at the end of the video, as well as between each period. In chronological order the video clips do now get concatenated together with the video fades in between. Before execution, each video clip does also get an audio fade at the begin and at the end.

In an extra monitoring mode text overlays are written over each event video clip and show the criteria why they were chosen.

6 Concepts and Implementation

The following sections build the main part of this thesis. They describe all major components of the framework in detail. Each section is semantically divided in two parts. The first part is explaining the concepts of the component and the second part is looking shortly into important implementation details.

6.1 Graphical User Interface

The graphical user interface for this work is implemented in "Tkinter", the probably most common used library for user interfaces in Python. It is easy to use and well supported and as the goal of this work was never to implement a special type of GUI, it was a perfect fit, because it does not require a largely extended specialized knowledge, if the goal is to create a basic GUI.

Within the systems architecture, the graphical user interface behaves like a controlling instance. It gets called by the "Main" class of the framework. Then it collects user input, sends it to the other components of the program and waits for the response, to finally show the final product to the user and finish the program with a callback to the "Main" class.

The implementation of the Graphical User Interface is a component that has a more sophisticated design by itself. The sequences of that component can be seen in the activity diagram of Figure 12.

At the beginning, a "fileopen-box" pops up and waits for the user to select a ".txt" file containing the metadata, as well as the corresponding video file. Of course the first window only allows text files, while the later requires ".mp4" or ".mxf" files (Figure 13).

Once the user completes these steps, a "Proceed" button appears. When that gets clicked, the "fileopenbox" disappears and the main program starts running. If the two files are not already specified, the program asks the user to complete these steps before proceeding (Figure 14).

Another option can be changed by a toggle button. Once pressed by the user, it switches from "on" to "off" and vice versa and lets the user activate the monitoring mode. As this whole framework is easily extendible, any other user inputs necessary for program extensions, can take place here before proceeding. A user could for

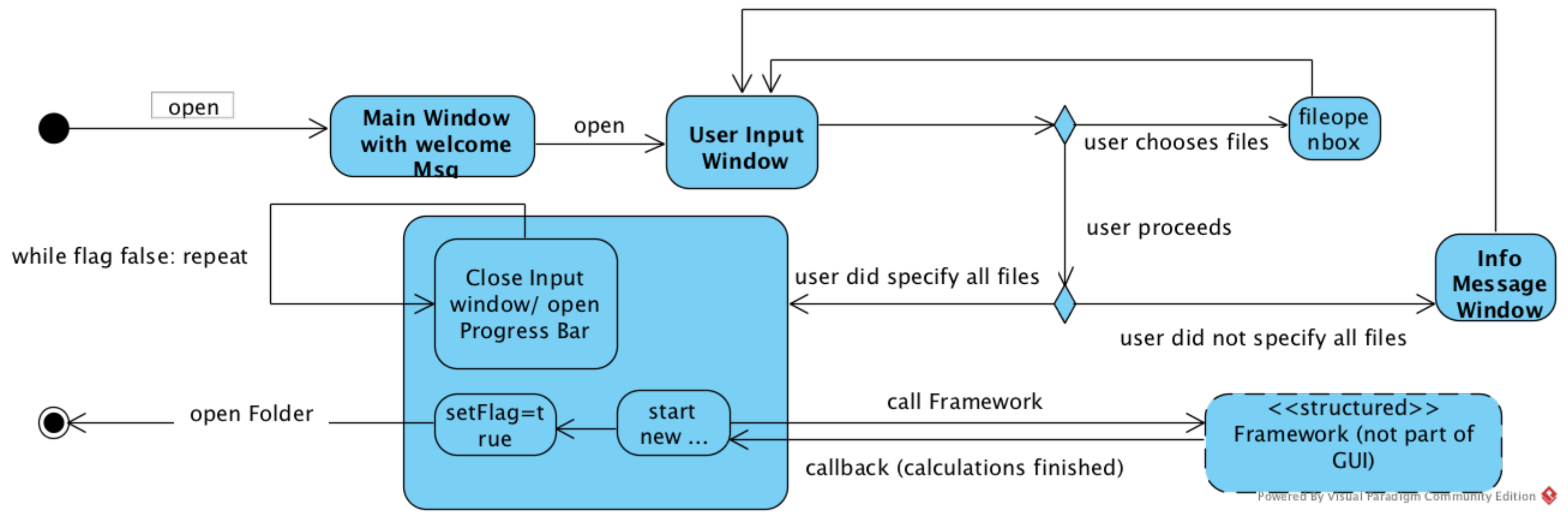

open
Main Window with welcome Msg
open
User Input Window
user chooses files
fileopenbox
user proceeds
user did specify all files
user did not specify all files
Info Message Window
while flag false: repeat
Close Input window/ open Progress Bar
open Folder
setFlag=true
start new ...
call Framework
callback (calculations finished)
<<structured>> Framework (not part of GUI)
Powered By Visual Paradigm Community Edition

Fig. 12. Activity diagram from processes of the Graphical User Interface

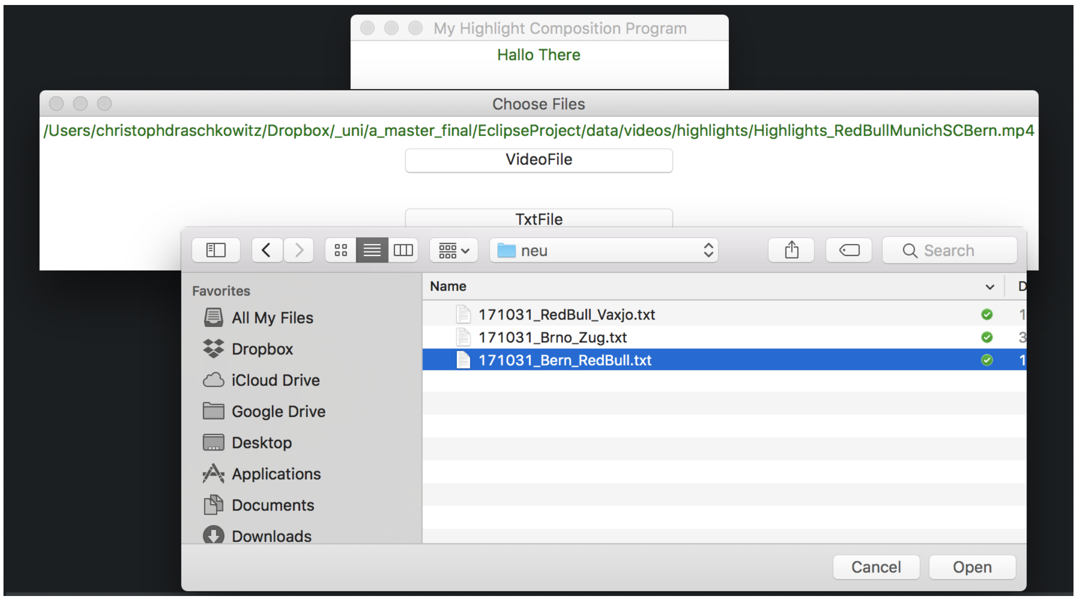


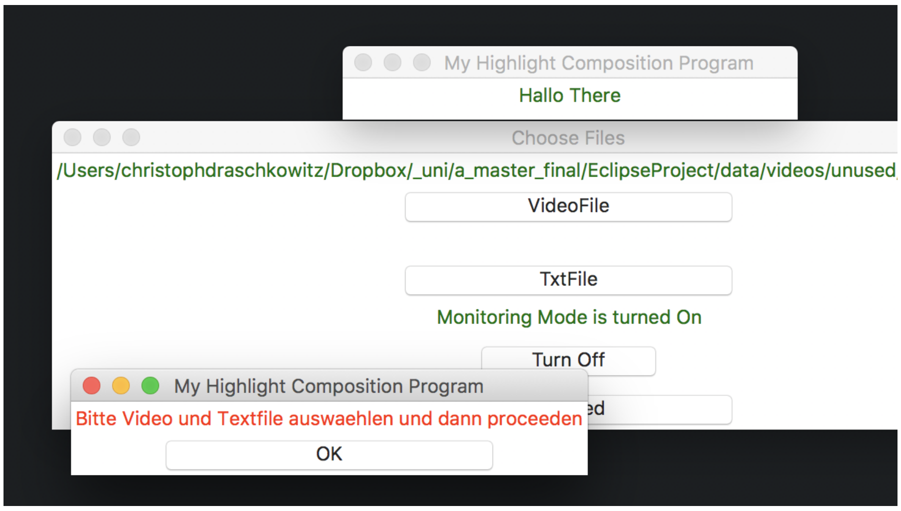
Fig. 13. GUI. Program asks user to choose files

example declare his preferences regarding favorite teams or players. Once the user proceeds, a moving "progressbar" shows up and asks the user to wait while the video is processed (Figure 15).

In order to do this in "Tkinter", it was necessary to start another thread when the user presses the "Proceed" button. The "goflag" gets turned, the input window destroyed (newWindow) and the main window shows the progress bar, while the extra thread calls the controller of the framework and waits for the callback (Listing 1).

Listing 1. GUI mainloop when user proceeds

```

        if (self.goflag==False):
            self.goflag=True
            threading.Thread(target=real).start()

          # destroy file openbox and alter text root message box
          newWindow.destroy()
          # set window to foreground;
          self.root.lift()
          self.mainLabel.config(text='Please wait... Your video is 
              getting processed')
```


My Highlight Composition Program

Hallo There

Choose Files

/Users/christophdraschkowitz/Dropbox/_uni/a_master_final/EclipseProject/data/videos/unused

VideoFile

TxtFile

Monitoring Mode is turned On

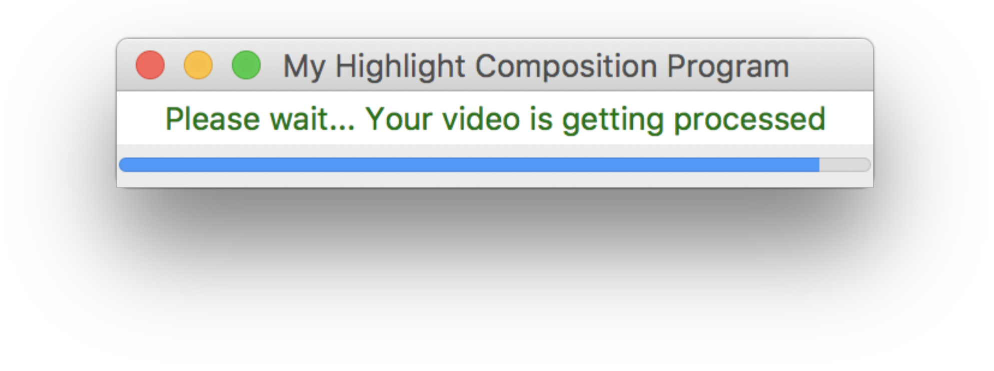
Turn Off

My Highlight Composition Program

Bitte Video und Textfile auswaehlen und dann proceeden

OK

Fig. 14. GUI step 2. User did not specify all Files.


My Highlight Composition Program

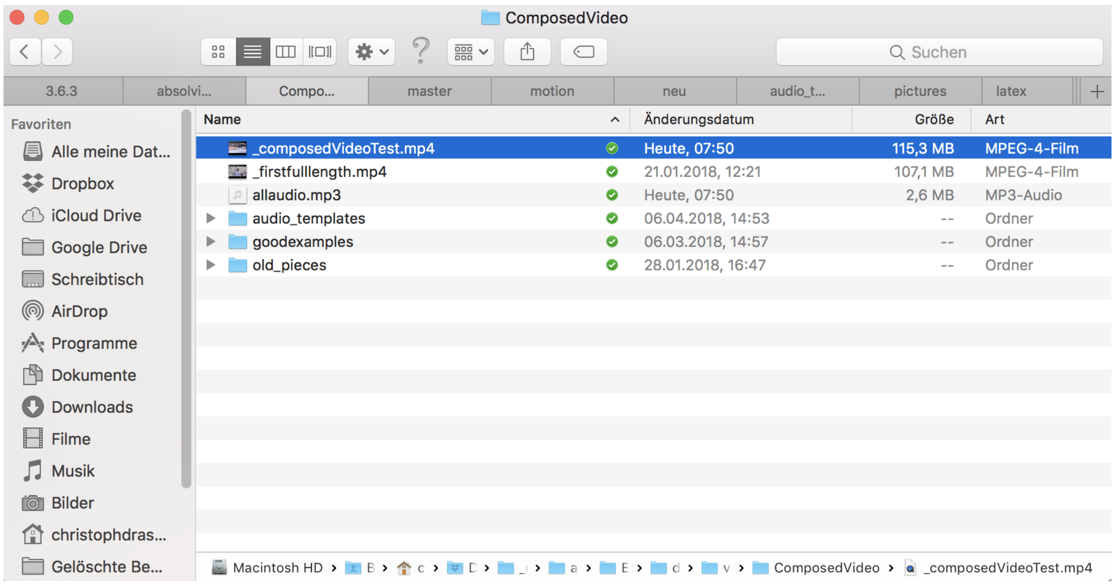
Please wait... Your video is getting processed

Fig. 15. Programm running, User is waiting for it to finish

```
                self.mainLabel = tk.Label(self.root, fg="dark green", text=
                    'Please wait... Your video is getting processed')
                self.mainLabel.pack()
                ft = tk.ttk.Frame()
                progressbar = ttk.Progressbar(self.root, orient='
                    horizontal', mode='determinate')
                progressbar.pack(expand=True, fill=tk.BOTH, side=tk.BOTTOM)
                progressbar.start(1) #    = 1 second

        return hist


```

```

            # thread calls controller
        def real():
            Controller.startRoutine(filename, filename2, self.toggle)
            print ("returned_to_gui")
            self.closeGui()
```

Once the program finishes, the window closes and the folder containing the final video pops up and is shown to the user (Figure 16). At this point another extension is possible. The program could provide the raw event clips that the program suggests to use and let the user choose which of these he wants to be considered in the final video.

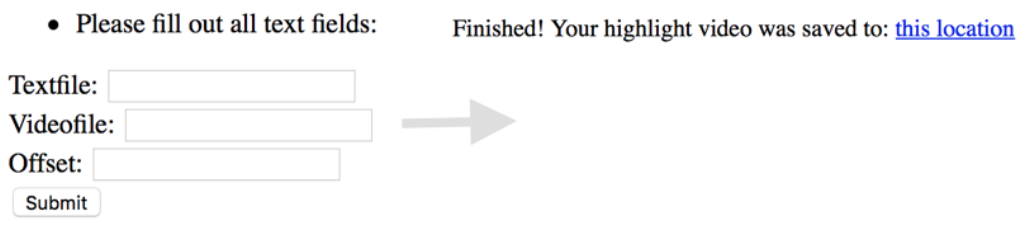


Name	Änderungsdatum	Größe	Art
_composedVideoTest.mp4	Heute, 07:50	115,3 MB	MPEG-4-Film
_firstfulllength.mp4	21.01.2018, 12:21	107,1 MB	MPEG-4-Film
allaudio.mp3	Heute, 07:50	2,6 MB	MP3-Audio
audio_templates	06.04.2018, 14:53	--	Ordner
goodexamples	06.03.2018, 14:57	--	Ordner
old_pieces	28.01.2018, 16:47	--	Ordner

Fig. 16. Folder containing the finished file pops up

The latest version of the framework was also adapted in a way that it is possible to run it on a cloud service. Flask, a common python framework for the deployment of applications on web-servers is used for that. In order to be able to run the program from any device, the GUI is slightly changed and realized in HTML, part of a Flask RESTful Web Service. A user is then able to provide input over a HTML form in the web browser. This way the workload can be shifted to the cloud service and the composed video is finally provided to the user via hyperlink (17).

To be able to handle an arbitrary number of videos, a new textfield to specify the time offset from the video broadcast file, is implemented as well.


- Please fill out all text fields:

Textfile:
Videofile:
Offset:

Finished! Your highlight video was saved to: this location

Fig. 17. Index Site and Response of the REST service deployed on a cloud service

6.2 Time Synchronisation

In order to work with the provided meta data, the text file has to be synchronized with the video. As every second matters for algorithms to work successfully (for example when analyzing the sound at a specific point of an event), synchronization has to be on point. The offset is always related to the actual event. For the live feeds of the ice hockey broadcasts those offsets are pretty low and also consistent throughout event types. Goals for example are usually right on point, and all other event types lack by no more than five seconds. For the evaluation phase however, the framework was adapted to be used with soccer broadcasts also. The meta data files of those broadcasts were not as accurate. Their events were sometimes timestamped 15 seconds late.

The provided metadata file, as well as the "mp.4" video file, do both provide information about their creation time. By comparing those two timestamps, the files can be synchronized. It is important to always check if both of them are provided in the same timezone.

However, a bigger problem arises if one of the following two cases appears:

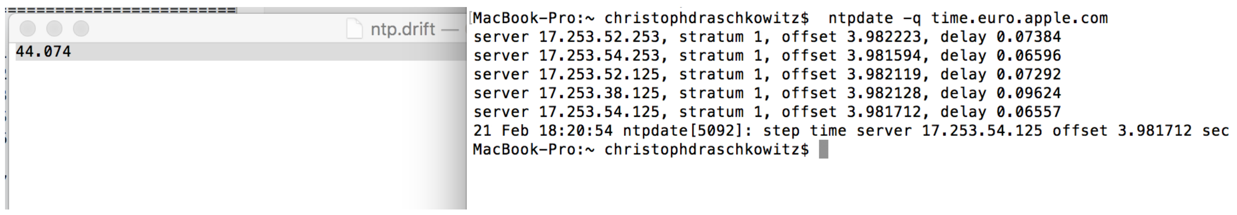
– When one of the files gets manipulated after creation:

In that case the timestamp of encoding could be adapted to the date of alternation and all other timestamps provided by the metadata file will not fit any longer, as the program will always calculate the time from manipulation. This problem can be solved by letting the user specify the time the video is already running when the game starts. After that, it is possible to only work with the meta data timestamp by calculating the time since the start of the game. This approach is actually used in a second project that was created for evaluation of the framework and is using soccer videos.

– When the two files were created on different devices:

Inevitably the synchronization will fail by a couple of seconds. The reason for that is pretty simple. Computers, nowadays usually synchronize their local time with the "Network Time Protokoll" (NTP) [37]. Every now and then, depending on the operating system used, the computer synchronizes its clock with an NTP server. Although there are a couple of mechanisms to minimize timing errors, like latency to the server, it is not possible to prevent the computer clock from drifting away from the reference clock (which is an atomic clock at the top level). Therefore the exact time can only be assured right after a computer synchronizes. With a computers clock drifting, both front and backwards, the time difference between two separate computers can easily reach a couple of seconds. Figure 32 shows the time delay on the computer used in this work.

To prevent this from happening, it would be possible to update the time of the devices more often, or to rely on external software that is able to synchronize two devices.



```
44.074
```

```
MacBook-Pro:~ christophdraschkowitz$  ntpdate -q time.euro.apple.com
server 17.253.52.253, stratum 1, offset 3.982223, delay 0.07384
server 17.253.54.253, stratum 1, offset 3.981594, delay 0.06596
server 17.253.52.125, stratum 1, offset 3.982119, delay 0.07292
server 17.253.38.125, stratum 1, offset 3.982128, delay 0.09624
server 17.253.54.125, stratum 1, offset 3.981712, delay 0.06557
21 Feb 18:20:54 ntpdate[5092]: step time server 17.253.54.125 offset 3.981712 sec
MacBook-Pro:~ christophdraschkowitz$
```

Fig. 18. Time delay on the local device, this framework was implemented on

Generally we can now separate the constant time offset (clock drift) and an inconstant offset (event). The constant offset is basically the clock difference between the devices that produced the video and the meta data file.

In this works implementation I chose a start point as an in-point for each video, which is not perfect, but is said to be a common approach by companies using those metadata feeds. A user has to check for how many seconds the broadcast video is already running when the game starts, and provide this information to the program.

In order to finally cut out the correct scenes from the video, the video time of the events needs to be calculated:

The method used for the ice hockey framework is using above mentioned method (media-info) to synchronize metadata and video file and finally the clock drift related offset (O_c) is added hardcoded to it, because this was more practical during the deployment phase:

$$E(metadata) + O_c + O_t = E(video) \tag{1}$$

where:

O_c is the constant offset between video and metadata
O_t is the type specific event offset
$E(< video/metadata >)$ is the timestamp of the event in video or meta data

In the adapted version for soccer games, that step is finally simplified, since clock drift is a known issue. The user has to get active anyway and provide the starting point:

$$(E_{start}(metadata) - O_{start}(video)) + E(metadata) = E(video)$$

(2)

With $O_{start}(video)$ being the time of the video at the start of the game, which is specified by the user.

Everything else in the coding in this part is straightforward. First the time of the encoding of the video is receiving by using the "mediainfo" library. Then the string gets converted to a date and is adapted to the UTC timezone. The same procedure is iterated over all event timestamps.

Other solutions would be possible, but exceptionally more complex. In [6] a way of reading out the scoreline (including the remaining playing time) from a broadcast video is proposed. [62] implements a finite state machine for visual analysis of the start of soccer games. Also background templates, coupled with motion analysis, would be possible. Because the shot angles at the start of a game are usually really similar, it would be possible to compare a template with the background as long as that shot is found for the first time and then have the game start as soon as motion happens.

6.3 Event Metadata

In this work there are two important metrics to decide whether an event is considered interesting or not. Getting information out of metadata must always be considered a priority, since it requires extremely low computational effort compared to other techniques like audio or even video analyzation. Hence analyzation of data must start with metadata analysis and any kind of information that is provided by the data must be used in order to work efficiently.

The event metadata in this work is provided by the company "Sportsradar". It is provided in the form of a valid ".xml" file that is automatically created by a live ticker at each game the company tracks. The live ticker is manually assisted by an employee. Right after the game the inputed data gets exported as the given XML file and is saved on the companies' servers. The most important thing to point out here is that the event metadata was in no way created or adapted to fill the needs of this work. Basically any other event information would be just fine as well. On the other hand, it makes it more difficult to read the file out correctly, since it is optimized for different purposes.

The metadata text files contain timestamps of all common events during the game, like shots, saves, penalties, start and end of periods, goals and many others. In this case the advantage is the amount of event data that gets captured. Not all of it was actually used, but combining events and ice hockey specific rules can create extra insights without heavy computing calculations.

On the downside, there is no official documentation available, and there is no guaranteed standard that could clarify if the data is correct and/or datatypes are correctly filled. Therefore this has to be taken care of within the implementation.

The big difference here, compared to other approaches mentioned ([61] [62]) is, that it is possible to constantly rely on the specific structure of the document with consistent information.

At the start of the file, basic information is provided, including a unix timestamp and the official names of the two competing teams. Each event has its unique id and a type that is captured in the "info" item as a string, as well as in the type section, as an integer. Furthermore there are items for the time. There is an item for the netto in-game time that is already over and one for the remaining netto in-game time in the current period. Aside from that there is another item for the number of period we are currently in, and of course the unix timestamp. Additionally another item specifies whether the event regards the home or away team.

Event by event is provided in that way. A whole lot of different event types are possible:

- Shot
    - Shot on Goal
    - Goal
- Face Off
- Time Running
- Time Stopped
- Empty Net
- Video Review
- Penalty Shot
- Suspension
- Two Man Advantage
- Powerplay
- Empty Net
- First Period
- Second Period
- Third Period
- Game Started

– Game Ended

As one can see there are also events presented, that help detecting situations where the game is in an intermission (Time stopped, start of periods). This makes filtering of important parts of the video a lot easier, and therefore helps working on performance issues. Besides the tags already mentioned and described above, there is also an item carrying the approximate position on the x and y axis included in some events.

The meta data feeds for ice hockey used in this thesis provide incredibly accurate timestamps. On the downside of it, sometimes events are triggered before they are confirmed (by the referee). Therefore in the implementation it has to be assured that the event is actually confirmed later.

Another possible downside can be seen in the suspension event data. There is unfortunately no information whatsoever on the type of penalty a player receives. Therefore fighting penalties and any other penalty that users usually want to have in their highlight videos need to be found differently, for example through additional online data, audio or video features.

Other available live feed sources are widely spread. In ice hockey there are for example statistics about the NHL available. Listing 2 shows a short excerpt from a freely request able NHL stats api.

Listing 2. Example Json metadata file from the NHL api

```



 {
"players" : [ {
  "player" : {
    "id" : 8471678,
    "fullName" : "Benoit Pouliot",
    "link" : "/api/v1/people/8471678"
  },
  "playerType" : "Blocker"
}, {
  "player" : {
    "id" : 8474578,
    "fullName" : "Erik Karlsson",
    "link" : "/api/v1/people/8474578"
```

```
      },
      "playerType" : "Shooter"
    } ],
    "result" : {
      "event" : "Blocked␣Shot",
      "eventCode" : "BUF201",
      "eventTypeId" : "BLOCKEDl\_SHOT",
      "description" : "Benoit␣Pouliot␣blocked␣shot␣from␣Erik␣Karlsson"
    },
    "about" : {
      "eventIdx" : 90,
      "eventId" : 201,
      "period" : 2,
      "periodType" : "REGULAR",
      "ordinalNum" : "2nd",
      "periodTime" : "00:15",
      "periodTimeRemaining" : "19:45",
      "dateTime" : "2017−12−13T01:00:48Z",
      "goals" : {
        "away" : 0,
        "home" : 0
      } }




    "result" : {
      "event" : "Penalty",
      "eventCode" : "BUF40",
      "eventTypeId" : "PENALTY",
      "description" : "Mike␣Hoffman␣Slashing␣against␣Josh␣Gorges",
      "secondaryType" : "Slashing",
      "penaltySeverity" : "Minor",
      "penaltyMinutes" : 2

     .
     .
```

This NHL metadata seems to be implemented as a Web API that is freely accessible but undocumented. As one can see, the data is pretty similar to the one used in this work from the Champions Hockey League. There are even a bit more events tracked (like the blocked shot in this example) and the id and other important information about the players participating, is immediately provided.

Also the type of penalty is already known and included, which is really valuable, since fighting penalties are usually a common part of highlight videos. A future consideration of this works implementation is to extract this information through multimedia features and/or gathered from other sources on the world wide web.

Basically any textual information is incredibly valuable, since it does only use very little computing power to navigate through those files. For all files it is important to use tags consistently, to not cause errors and to make automatic machine reading easier, with less code. For now, coding required to implement a couple of special cases, caused by the structure of the text files.

For the implementation of the concepts of this section it is important to remember that the metadata file is not adapted to fill the needs of this work. In our case, the file provides a lot of information, but still reading the file out correctly is rather difficult, since it is optimized for different purposes. Reading out the file in the first place is rather simple by the help of the ElementTree XML API, which is a Python build in module.

However, an "IF" loop needs to be started, as sometimes different attributes are available for the events, depending on the available metadata. Also catching errors gets more complex for that cause. If, for any reason, any attributes do not have a proper value in the meta data file, an empty string gets assigned to the objects attribute. Finally the event objects get created. Time borders and replays are realized as objects themselves and will lather on be used to store information about the events' boundary.

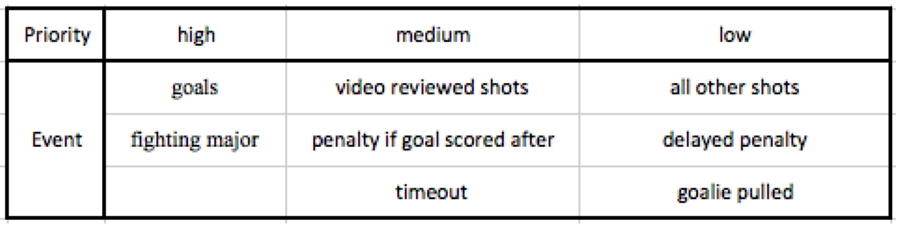
6.4 Event Score Calculation (Rule Set)

The event score calculation is started in a first step by analyzing the metadata text file.

From soccer games a work proposed in [57] analyzed 143 events that were important enough to show replays and discovered, that only four percent were neither attacking scenes nor penalties, but for example injuries. I therefore assume, although I am dealing with a different sport here, that goals and penalties are considered the most important events. This is also what I would have expected before research.

[61] defines three priority levels and adjusts all events to one of them to create a score.

The approach taken in this work was to assign scores between 0 and 100 to each event type. However in the final implementation it is actually sufficient to split them up in three basic priorities as well (see Figure 19):



Priority	high	medium	low
Event	goals	video reviewed shots	all other shots
	fighting major	penalty if goal scored after	delayed penalty
		timeout	goalie pulled

Fig. 19. The three priorities, that the events are sorted into, explained. They are sorted by event type

Events from priority 1 do not get further processed, as they are almost certainly part of the final highlight clip. Only when there are loads of goals (ten or more goals) it can be considered to leave out a goal for example.

Scenes from the second priority level are most likely part of the final video, again depending on the amount of scenes from priority one and of course the distribution of scenes. This is again explained in the section about the video composition and does not influence the event score.

Events from the last priority will receive further audio analysis in order to calculate exact scores for them. In the current implementation however, delayed penalties, timeouts and scenes, where the goalie gets pulled, are not considered for the final video. The reason for that is that the timestamps for those events are very inconsistent, which makes it extremely hard to detect event boundaries. A solution is of course, as done in a lot of research papers like [17], to analyze and classify the shot cuts around the event in order to find good event boundaries for the video, by adding knowledge about domain specific cutting habits. For this work this was considered too much extra effort for a rather small reward.

That leaves the shots on goal for further consideration. The audio analysis of these is explained in detail in section ?? Audio Extraction. In the final state of the work four different audio features were chosen to be combined to rank the priority 3 events. Besides weighting the four features separately, two types of combining were considered.

Firstly the product of all four. The advantage of this would have been that a small value in one of the feature would have a bigger impact on the score, than it would have when summing up the features, which is positive. The finally chosen method however, was different.

All home team and away team events were calculated separately, since fans are not spread evenly and that would hurt the ranking significantly. Then, feature after feature, the average score gets calculated and all events that do not exceed a certain percentage of that average, get a malus on its score in order to penalize very bad results in any feature category.

After that, all feature scores get weighted by feature and then just get summed up and form the final ranking in two lists. One for home team events and one for away team events.

Then, in the final stage of the ranking process, events get re-ranked depending on match score and match time. For example a shot on goal by the home in the last two minutes of match when the home team is trailing one goal, will certainly be on the highlight list.

For the implementation of the ruleset it was sufficient to simply write methods that follow the rule set instructions.

Listing 3 shows a part of the implementation of the rule that specifies that shots on goals are more important if the score is close and it is the last two minutes of the game.

Listing 3. Score calculation example

```

    #get match result
    for event in eventlist:
        if "ENDED" in str(event.info):
            matchscore=event.matchscore

    matchscore=matchscore.split(":")

    #get match time for each event
    for event in eventlist:

        temptime=event.mtime
        temptime=temptime.split(":")

        # mtime in seconds:
        if len(temptime)==2:    # 3. is der :
            mtime=int(temptime[0])*60+int (temptime[1])
        else:



                    if (mtime>=(58*60) and (eventlist[0].get_side()=="home" and
                        matchscore[0]==matchscore[1] or int (matchscore[0])
                        +1==int(matchscore[1])) ) or (mtime>=(58*60) and (
                        eventlist[0].get_side()=="away" and matchscore[0]==
                        matchscore[1] or int(matchscore[0])==int(matchscore[1])
                        +1)) and event.info is not "ENDED":
```

However, when the rule set grows bigger it would be recommended to, for example, save the rules in a structured form, or in an ontology and then generically read them into the program and react to them from the methods. That way it would also be easier to further extend or alter the rules, especially if the alteration and specification is done by a non computer expert.

As one can clearly see from the Listing, it makes sense to implement the rules more generic in a declarative form, if the rule set is big enough.

6.5 Audio Features

Many of the works mentioned in the state of the art section, that do use more methods than video and audio features, do however only measure the amplitude of the audio signal.

In this work there are two important metrics to decide whether an event is considered interesting or not. As already mentioned, getting information out of metadata is always considered a priority, since it requires extremely low computational effort compared to all other techniques and possibilities.

However, compared to the amount of events provided, only a small set can definitely and without further investigation be considered important. Therefore, in a next step, the idea is to analyze the sounds around events, that are possible candidates for the final video.

Other works have found good results with similar approaches: [45] used machine learning and found out that exciting speech will start up to 5 seconds after an event. [30] claims that...

> ... the observation that during important events, the sound is usually more clearly audible over a long period of time than is the case with less important events [30].

Unfortunately a problem came up when comparing metadata timestamps with the actual video: While the timestamps on goals and other fairly important events were exactly on point, on a couple of other events, like shots on goal, a delay of two to four seconds, depending on the video, could be seen. The only positive is, that it was fairly consistent throughout the video. That way it can be assured that it only happens within certain events (as mentioned, mostly shots on goal). Furthermore, the questionable events were always timestamped late.

It follows that the events happen two to four seconds before the timestamp. This of course makes it more difficult to compare the audio data as we have a window of two seconds of uncertainty.

However, the approach to choose the best events of that team will always be subjective and dependent of certain features that differ from person to person. The idea in this work is the following:

By analyzing the crowd behavior we might get an idea what most people, that were at the rink, think about the situation. This makes the decision more objective automatically. We are therefore going to analyze crowd noise in specific. Note that the proposed algorithm does also work when the video is provided with the voice of a commentator. As a TV commentator has the duty to be as objective as possible and should be a sports expert, the desired goal should still be fulfilled.

In order to identify usable audio features, suitable audio parts from the given videos were extracted. Sixteen parts from different events and situations of different amplitudes of loudness were chosen and it was tried to cover as many different noises as possible. All noises were extracted from the four initial match broadcast videos that were available at that time.

These parts included different voices like whistling and drumming in a nutshell, as well as the more practical composite voices, that form the background crowd noise during the game. For example fans cheering, or a fan club singing their slogans.

Furthermore goal situations of different loudness and excitement were included (for example very loud or quiet). In order to create audio features and weightings effectively, those clips included goal scoring scenes, and goal chances, both from home and away teams.

The length of the audio samples taken for testing and for the final implementation, was five seconds long, started approximately three seconds before the event and ended two seconds after.

In this work audio analysis is basically only used for shots on goal, but could without any adaptions be used for any other event too, for example for the importance of penalties.

One of the provided video broadcasts had the live commentary on the audio track too. Analyzing the commentators speech seemed a lot easier and had better success, than analyzing the crowd noise. However, it has to be mentioned that this might also depend on the commentator. In the analyzed videos he got really excited when good scenes happened and anyone could hear that. With other people commentating, this might be more difficult to analyze, even though the crowd noise is not always the same either. As a TV commentator is supposed to be objective, taking his or her voice as a parameter for excitement measurement seems legit and promising.

By analyzing the samples with Audacity [7] a couple of assumptions were made.

Figure 20 shows about two seconds before and after a goal scored by the home team. The exact moment of the shot is taken can be clearly seen. When the crowd starts cheering, a significant raise in the amplitude as well as in the frequency can be seen.

Since all goals scored are already marked in the metadata however, the events we are looking for in this sections are harder to detect.

Up to that, for two reasons it is not possible to only separate steady background noise and cheering.

[7] http://audacityteam.org/

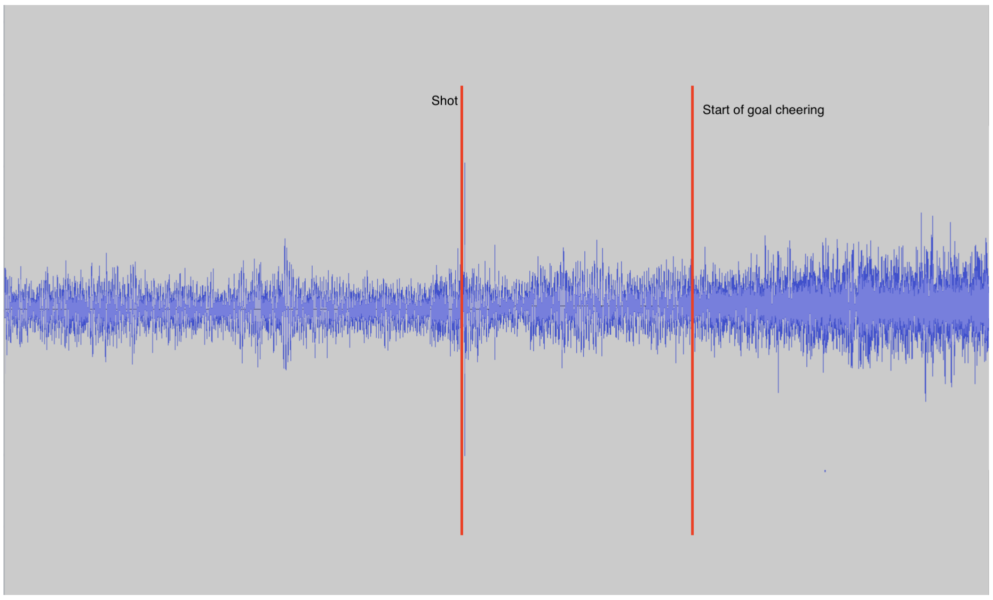


Fig. 20. Situation of a goal scored. 5 seconds sample. The red lines specify the time a shot is taken and the start of the crowd cheering.

At first, especially in European sports, fans tend to support and cheer for their team for long periods no matter if there is a special situation on the field or not (Figure 21).

Secondly, especially in those European, international ice hockey games that were analyzed in this work, there are usually very few away team fans at a rink, due to the fact that teams are spread all around Europe.

This of course would not be that significant if one would consider national wide games only. Those cases make it even harder to correctly rate events from that team. In this work therefore, events from home and events from away teams are analyzed separately, so they can be compared only to themselves. In this way there is no need to use any hard threshold to verify an event's importance.

At last there exist disruptive factors that could be interpreted as crowd cheering. For example whistling and drumming, which is very popular in hockey rinks, is loud and/or has a high frequency (Figure 22).

Therefore it was important to analyze those special patterns, to be able to filter them out. Testing showed, it is however very hard to filter out specific sounds, like drumming, because too often they appear together with other noises. [58]

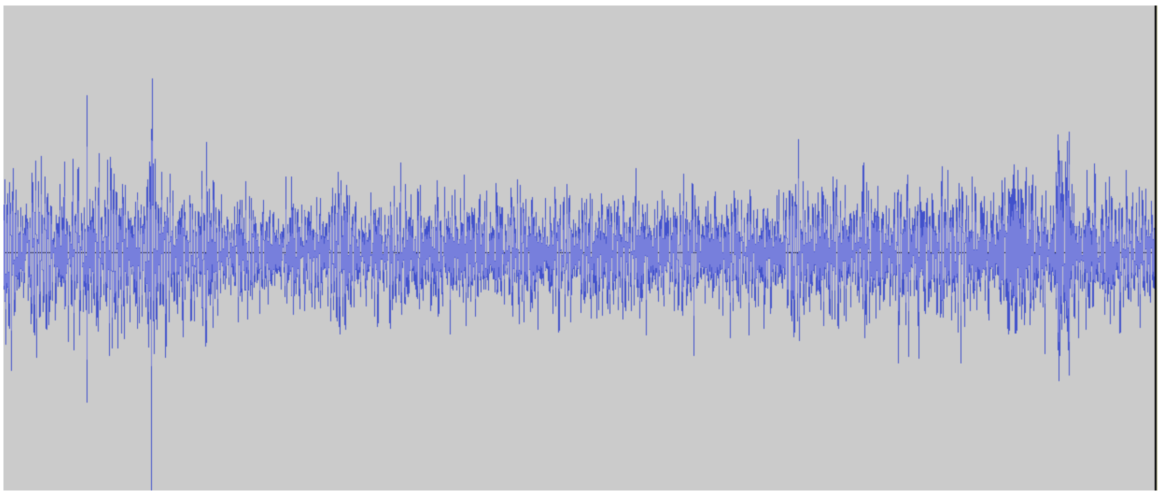


Fig. 21. Standard background crowd noise as an example for typical crowd noise in ice hockey games.

reported falling precision when crowd whistling was falsely interpreted. This is also mentioned and considered in [60]. [26] furthermore points out that ranking crowd excitement only by energy is not always sufficient.

[16] shows frequencies at which whistles are detected by using a Fourier Transformation or a high amount of zero crossings. For these reasons, in this work the audio analyzation was implemented by testing out a full range of audio features from both time and frequency spectrum and by creating new features inspired by test results. Specific sounds that one would consider important, were not specifically filtered out. This would be a different approach, that could also be successful.

For all features different setups were tried also. What is meant by that is, for example that the results of one feature calculated over the whole timespan of an event were always checked and afterwards that was compared to the same feature calculated over parts of the audio signal and than summed up or averaged the results of all of them.

But let us first define the criteria that are seen important in order to find good events: The algorithm should aim at rating events by crowd noise analyzation. Therefore maximization of the features for speech recognition was not considered. However features that are used for speech recognition can still be used.

The criteria used for this part are pretty simple. First of all it is assumed that louder is better for detecting excitement. That does not necessarily mean that

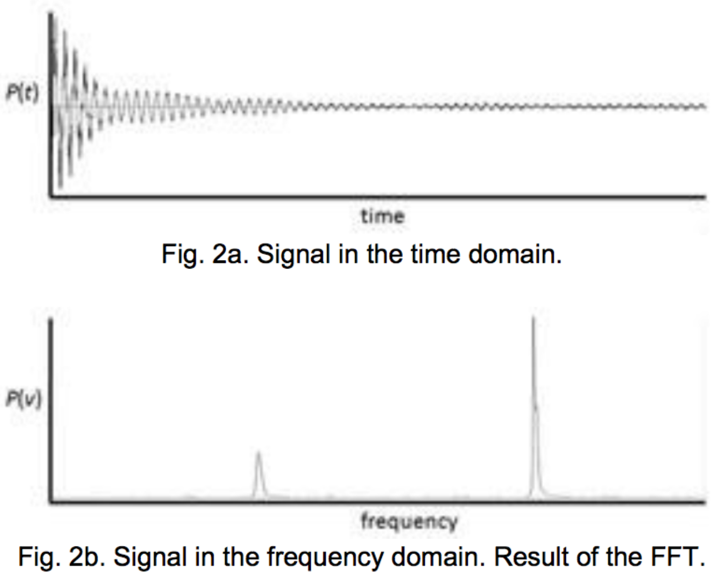


Fig. 22. A normal whistle explained by [16]

the sample must be loud over its whole sequence, as can be seen in the features afterwards, but generally spoken it is an assumption that should hold for all type of events. A feature that measures the difference of loudness within the same sample was also implemented, because research showed that, at important moments in a game very often the crowd gets quiet and immediately before the event raise its voice very high.

Secondly it is assumed that the pitch at important events is also higher than usual. The pitch is closely related to the frequency of a noise, only that it does measure human perception. The assumption from analyzing the test samples is that, while normal speech has a rather low frequency, excitement should be characterized by higher frequencies. This however could possibly not be true for all possibilities and might be a future issue to work on.

Human perception receives acoustic pressure as loudness and the frequency of voice as the pitch of an audio signal.

The first features that were tried for this work are covering those two parameters. Those features were the energy and zero crossings.

LibRosa has a built in function for energy that is implemented as the root mean square energy. The root mean square is defined as:

$$\sqrt{\frac{\sum_1^n x_1^2 + x_2^2 + x_3^3 + \ldots + x_n^2}{n}} \tag{3}$$

Accordingly bigger numbers get weighted higher. The audio energy is very closely related to the loudness of a voice. The loudness is how a human would hear a voice. Therefore a raise of energy does not raise the loudness proportional due to human perception. Especially in low regions, a small raise in energy leads to a duplication of loudness to a human ear. Hence energy is taken as one of the most important features in this section, but human perception will influence its implementation.

Zero crossings are simply the occurrences of when the signal crosses the zero value at amplitude level. Hence the higher the frequency of the voice, the higher the zero crossing rate. It is used especially in speech detection, since speech has a low amount of zero crossings, where else crowd noise has a lot of zero crossings. However, as we try to detect excitement in crowd noise, a higher level of crossings is what we are looking for. This also applies when a commentator is speaking, because in our case he speaks a lot louder than the background noise. Any crowd noise is therefore automatically filtered by other audio features like energy. Zero Crossings are usually implemented as a rate value over some specific time, or as a total count. In this work it does not matter which one is taken, because the samples do always have the same length.

However, the loudness for a human ear is not linear to the acoustic pressure of the sound wave, nor is the acoustic perceived pitch linear to the frequency. For this reason the unit scales Sone, Phon and Mel were created.

As described, one of the provided videos that this work was tested on, was including the commentators speech. By only using and partly combining these two features, it was possible to extraordinary exactly rate events by the beforehand explained criteria. By implementing Mel Frequency Coefficients as a feature it did even produce better results. Mel Frequency Coefficients take human perception by using the Mel scale into account.

The Mel Frequency Cepstral Coefficients have for some time been successfully used for speech recognition and do also find their place in more complex tasks like music classification or audio similarity measurement [36]. Creating them consists of several steps including Fourier Transformation in order to decompose the signal into its frequencies and converting the results, so it refers to the mel scale, which describes human perception of pitches. This process is more closely described in the background section. Almost every audio based highlight detection approach is using these coefficients [60] [26] [48].

Nevertheless the three other videos processed, did not have human speech on the audio and were therefore harder to judge. It was experimented with several other features that did not provide significant improvement (if any at all). Those were for example flatness, rolloff and also the delta of the mel frequency cepstral coefficients.

Spectral centroid, spectral rolloff, spectral flatness and others are explained in [48]. Note that only because they did not work in this working environment, does not mean they are not suitable for this task. They just did not perform the way they were expected to. However, by testing a combination of them in a machine learning environment, might consider them good features when combined properly. To run such an algorithm over all features would nevertheless require a much bigger amount of audio samples and for best performance, does probably require the data to be supervised. [60] for example used 1500 audio clips for the training set.

As mentioned, Figure 20 shows about two seconds before and after a goal scored in one of the matches without commentary.

When the goal is scored, the frequency and loudness of the audio signal raises by a big margin. Analyzing a lot of events underlined that this is common for events that are considered important for highlights. Therefore a new feature out of the existing ones, namely the delta of the energy feature, was created. Basically a high difference between the lowest and the highest point in the amplitude is considered good.

For the composition of the chosen features different weights were tried. At the current setup mel frequencies are weighted down by a little and the delta energy feature is weighted to be more important.

Furthermore events that contain a feature value that is well below the average of that feature, compared to all other events of the same type and same team, get a malus. This approach was chosen, because obviously bad events did most of the time have one feature at a really low level. However, sometimes better events did also suffer from it.

Most of the work regarding the audio features was of course research and testing for the right features and weights. The basic implementation of the audio features themselves is implemented by the "LibRosa" library. Most algorithms were adapted in order to fit more into the specific needs however.

In the following, the creation of the delta energy feature is described more precisely: In order to get rid of noise, the signal is split up into five even parts (because it is always five seconds long). For each part the root mean square energy gets calculated just like mentioned before. Then for all pieces the difference between a piece and the next piece gets added to the final sum (Listing 4). The delta of the energy performed significantly better for most of the situations and was hence considered for the final audio algorithm.

Listing 4. Code of self created delta energy feature calculation

```python




        splitter=int (len(audio)/5)
        deltalist= []
        for i in range (5):
            high_energy_count,high_energy_norm = rsmeEnergy(audio[i*splitter:(i+1)*splitter])
            deltalist.append(high_energy_count)


        # calc sum or max and add to delta sum
        a= []
        a.append(abs(deltalist[0]-deltalist[1]))
        a.append(abs(deltalist[1]-deltalist[2]))
        a.append(abs(deltalist[2]-deltalist[3]))
        a.append(abs(deltalist[3]-deltalist[4]))

        t=abs(deltalist[0]-deltalist[1])+abs(deltalist[1]-deltalist[2])+abs(deltalist[2]-deltalist[3])+abs(deltalist[3]-deltalist[4])
        energydeltasum2.append(t*10)
```

6.6 Video Composition (Rule Set)

The composition of videos is in general a very creative task and the quality of a videos composition is always highly subjective. However there are certain guidelines that, if done correct, almost promise a better result. For this work therefore a small set of rules was created, put together by the help of video cutting specialists, observations and relevant specialist literature like [38]:

Figure (23) shows the assumptions made and the solutions implemented in this work:

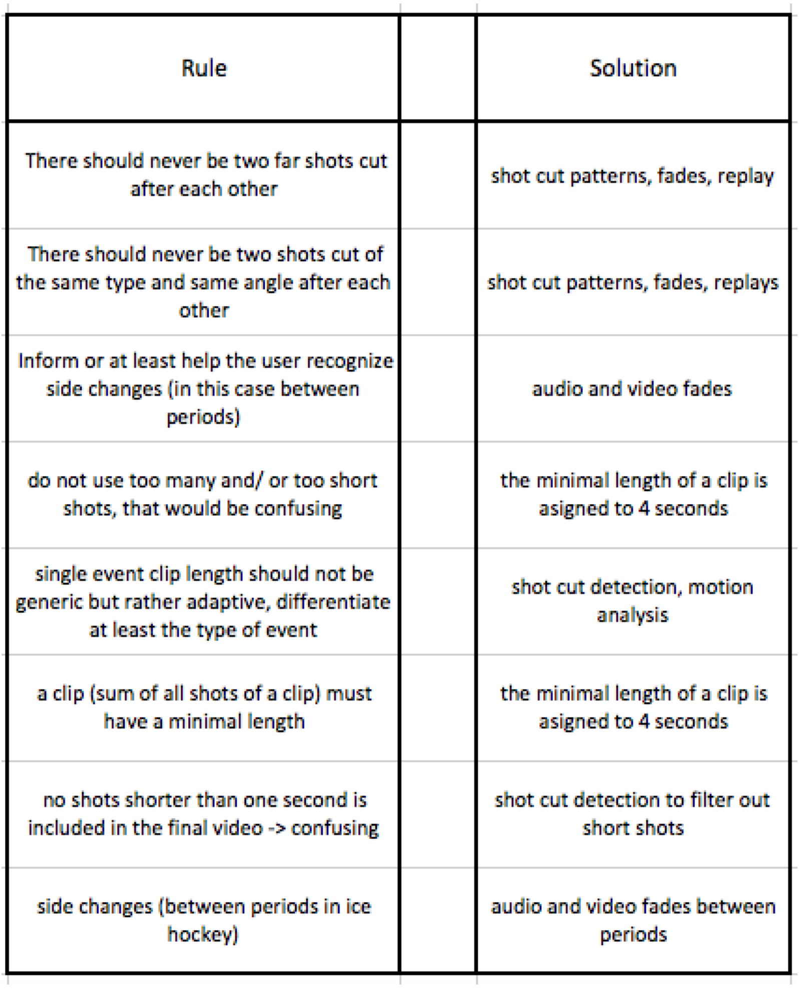
Within the same event, there is no need to manually watch if there are two shot cuts of the same type or too many cuts in a short time, because the input video is already provided as a broadcast. It can therefore be expected that those rules apply within events. For shot cuts between events several mechanism have been applied. If the period changes, a special video fade is applied just like described in [31] for example. Most of the time replays get composed after an event. These replays are mostly not from far view like the usual game sequence is. Up to that the logo fade, that is shown before a replay, in the scene, was left in the scene, just to make sure. Furthermore the shot cut detection component makes sure that short shots do not get applied in the final video when material is cut off. Motion Analysis takes care of the adaptive clip length while remembering that event clips must not be shorter than four seconds.

This next part discusses the possibility to take advantage of certain repeating shot cut patterns that can be used to make analyzing the broadcast input video easier.

To start really basic, let us get away from sports specifically. [38] describe a shot change as an instrument to focus

> ...the audience's attention on specific areas of the scene [38].

Hence it does also make sense to change the shot after important events, but that does not necessarily mean that those events are the only reason to change the shot of course.



Rule		Solution
There should never be two far shots cut after each other		shot cut patterns, fades, replay
There should never be two shots cut of the same type and same angle after each other		shot cut patterns, fades, replays
Inform or at least help the user recognize side changes (in this case between periods)		audio and video fades
do not use too many and/ or too short shots, that would be confusing		the minimal length of a clip is asigned to 4 seconds
single event clip length should not be generic but rather adaptive, differentiate at least the type of event		shot cut detection, motion analysis
a clip (sum of all shots of a clip) must have a minimal length		the minimal length of a clip is asigned to 4 seconds
no shots shorter than one second is included in the final video -> confusing		shot cut detection to filter out short shots
side changes (between periods in ice hockey)		audio and video fades between periods

Fig. 23. Composition Rules and Solutions

In this case most of the game is shown from the main camera (Figure 26). In case of important events, one can therefore expect a switch to a closer shot type.

[5] uses shot type detection of the broadcast to gather information about an event (Figure 24).

And as described in [18] and [67], structure analysis of shot type compositions can help to find event boundaries reliably. Those examples show that it makes sense to observe the structure of shot types.

The work proposed in [18] empirically define the following rules for what is happening right after goals:

– There is a break of at least 30 to 120 seconds before the gameplay goes on.

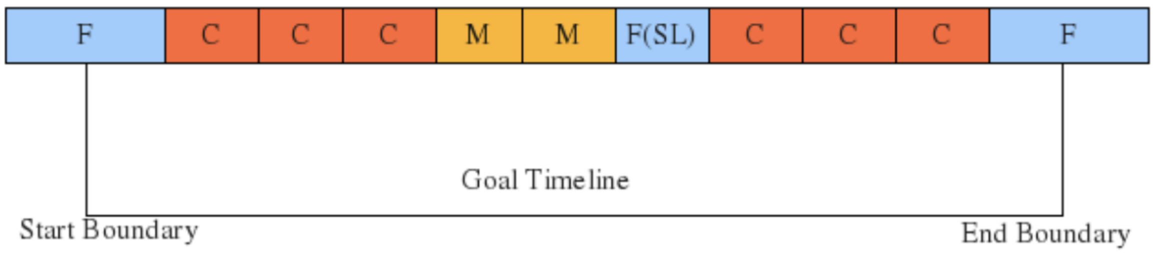


Fig. 24. Typical shot type sequence in a soccer game. Figure from [5]

- At least one close up shot is shown.
- After close up shots, replays are following.

Observed in [61] is the fact that the launch of replays in broadcasts differs from sport to sport. In soccer games for example, replays are shown during the game, where as in basketball replays are shown in the next break of the game. As ice hockey games do also have a lot of in game breaks (apart from breaks between periods), it can be assumed that ice hockey broadcasts behave similar to those in basketball regarding this.

In this work the exact same rules and numbers were not applied, but it shows that repeating rules can be used to make any shot cut, replay or goal detection a lot easier. In the implementation self created and empirically defined assumptions were made as the basis of those mentioned above. Those are part of the algorithms of the following sections. After certain domain specific events for example, one can wait for special shot types or replays and does not have to analyze the whole video sequence.

Very helpful is the fact that all bigger sport leagues world wide provide a certain script of what exactly should happen before a game, called a "run down". They, for example, define that one minute before the start of the game, the official logo has to be shown at full screen for two seconds. This could be used in this work for a better time synchronization of video metadata and local computer. An example run down of the Austrian soccer league can be seen at [8].

[8] http://www.bundesliga.at/?proxy=redaktion/OEFBL/Bestimmungen/BL_Spielbetriebsrichtlinien_Anhang-1_Run-Down_ab-01072016.pdf

For the implementation of this section the same argumentation as in the section about the event calculation ruleset applies: Simple methods that follow the rule set instructions were used and a generic way of saving and reading the rules would be desirable for a better adaptable framework. As already mentioned, for a more complex rule set, a good approach could also be to use ontologies, in order to create and extend the ruleset [56].

6.7 Shot Cut Detection

There exist quite some really good algorithms to find shot cuts [33]. In this work however, it is assumed, that only hard cuts have to be found, (except for the case where a replay is played, but more on that in the next section). That means that shot cuts are not split by video fades (that slowly move into the screen) and the shot cuts will therefore be easier to find.

As our only drawback is the limitation of computation power, the goal is to use a method, that delivers a fine tradeoff, on one side, being good enough to find all shot cuts, but at the same time, not needing a whole lot of calculations to achieve that. Data mining algorithms that analyze certain image features, are therefore not considered and also functions that make, create and use texture descriptors and key point descriptors, are too heavy for this cause.

Now the basic idea is to track the similarity of the stream of images. Once a low similarity is found, it can be assumed that a shot cut was detected. Doing research on that subject, the by far most mentioned method is the comparison of image histograms, with the drawback being, that totally different pictures could end up having the same histogram, because the composition of colors is the same. The classic example here is that a beach and a banana will eventually look the same on a histogram.

Another method followed on the way of finding a simplistic approach, was trying to detect a shot cut by comparing each pixel to the same pixel of the next image. Of course some optimizations like only reading every second column et cetera were used. The idea here was that, given twenty five pictures a second, the changes in the picture are minimalistic and can therefore even be tracked by a extremely simplistic method like that. This approach did work as well as comparing histograms. Nonetheless the possibility for optimization, by for example only reading in every second picture, was not as good as with comparing histograms. That is why the decision to use a comparison of histograms was made.

Methods with histograms and other more sophisticated methods are explained and compared in [33].

For the implementation of the shot cut detection OpenCV was chosen, because it provides helpful functions for the most of the mentioned techniques. Now several steps are necessary to compute the histogram of the single images and compare them to each other.

For best practice the actual image gets converted in HSL color space. This promises better results as perfectly explained by [12]:

> It is important therefore that the features of interest can be distinguished in the color dimensions used. Because the R, G, and B components of an object's color in a digital image are all correlated with the amount of light hitting the object and therefore with each other. Image descriptions in terms of those components make object discrimination difficult. Descriptions in terms of hue/lightness/chroma or hue/lightness/saturation are often more relevant [12].

However in researches it can also be read that sometimes this step gets skipped in order to save computing time, as it requires a complete transformation of an image on each comparison (only if the second picture can be saved to use as input for the next picture, as implemented in this work, otherwise it is even two). Instead of RGB, where each pixel saves a value for red, green and blue, in HSV color space a pixel is described by hue, saturation and a value. So the actual color is only saved in one channel.

By the help of the "calcHist" function from OpenCV histograms are then getting created. The next step is to normalize the histograms, to make sure we have the same range of values in both histograms when compared later. (Listing 5).

Listing 5. Histogram creation with OpenCV is fast and easy

```

def createHistogramm(frame):
    hsv = cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
    hist = cv2.calcHist([hsv], [0, 1, 2], None, (8, 8, 8), [0, 180, 0, 256,
        0, 256])
    hist=cv2.normalize(hist, hist)

    return hist
```

Finally the normalized HSL histograms gets compared with the associated "compareHist" function and receive a numerical metric as a result. As stated in the documentation, the function does not work well with very dense histograms. That however, should not be too big of an issue in this case. In order to save computing time, only every second row gets read in those last steps regarding the histograms.

The OpenCV function provides four different metrics to compare the data: Correlation, Chi-Square, Intersection and Bhattacharyya. Correlation returns 1 for a perfect match and 0 for the opposite. Chi-square, a well known statistical measure, that delivers zero for the good match and higher numbers (over one) for worse results. The Intersection method takes the sum of the minimums of the pictures, which results in higher numbers on good matches. The Bhattacharyya coefficient splits the sample in partitions and then performs some more complex operations (than the other methods). It is widely used in image processing and feature extraction. [15] describes usage and same advantages of that method over for example intersection. The Bhattacharyya coefficient does also return zero for a perfect match. The higher the number gets, the more different the images are. Other usages are shown in [24] and [52].

While Chi Square needs the fewest and Bhattacharyya the most extensive calculations, in practice of this work there was not much time difference between any of them. The Bhattacharyya method was actually working best for this works purposes, event tough the other methods performed well.

A fixed threshold is then used to differentiate between shot cut and no shot cut, because the differences between them are usually significant. However, [18] uses adaptive thresholding of the histograms to be even more accurate, which would be a possibility that is not too complex.

To complete the algorithm, certain optimizations were applied: The threshold function, implemented to decide wether an image is considered a shot cut or not, adapts to the situation. In normal situations the threshold is set relatively low. The differences from one image to another are usually marginal, so that a small deviation can be considered an anomaly. When a logo is detected however, we know that the scenes coming up are replays. As the frame rate of the video remains the same in that parts, we can assume that the difference between images becomes bigger. Hence the threshold is moved up.

The pseudocode in listing 6 explains what steps the shot cut detection algorithm is going trough in a simplified way of the basic implementation.

Listing 6. Pseudo code explaining the shot cut detection and replay algorithm

```

1: read background logo
background=readBackground(logo)

2: convert logo to hsl and create histogram

3: read 1. frame:
 frame=readFrame(video)

4:
   for frame in frames [1:end]
         hist  =frame . convert
         # if replay mode; put up threshold
         if flag =true {
                 threshold + = 0.5        }
         # skip frame if shot cut foundM (no shotcuts expected for next
             skipper frames)
         if skipper>0 {
                 skipper--
                 continue }
         correlation= compareHist (oldFrame,newFrame)
         if correlation > threshold  {  # shot found
                 list.add (frame)
                 skipper=10
                 if replay
                         write to Dict
                 }
                 }
```

6.8 Replay Detection

For replay detection several completely different solutions are possible. OpenCV would for example allow to detect fragments in images and then track them in the following images. That way the speed of movement can be estimated.

In the provided broadcast video, as well as generally in most broadcasts, a short video fade is put before and after a replay occurs. Usually the image that gets faded is some logo. In this case it is the logo of the Champions Hockey League:

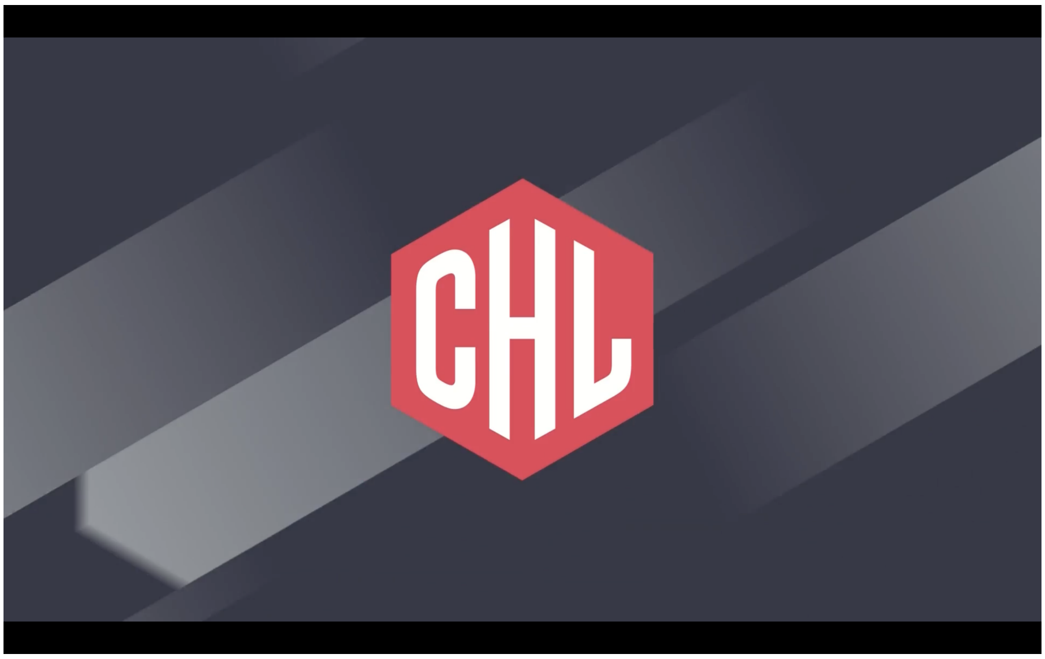


Fig. 25. CHL logo background extracted from broadcast

The logo image does only need to be compared for correlation to the images extracted from the video at shot cuts. Most of the time replays of certain events are shown right after the event has happened. Therefore a pretty specific time that can be searched for replay parts is known. It was decided to choose this timespan depending on the event type, as goals will usually result in a higher number of replays.

To save computing time, the implemented algorithm does directly work together with the shot cut detection algorithm. When a shot cut gets detected the replay detection jumps in and compares the present image with the logo.

This implementation of the replay detection is similar to the shot cut detection by computing and comparing histograms.

Again there are many more reliable ways to do it, but testing with histograms confirmed to be the fastest. However, template matching would be a good alternative besides the before mentioned methods. In OpenCV, the "matchTemplate" function can also be used to detect parts of images inside other full images. That is however pretty overpowered for these purposes.

Whenever a shot cut is detected, the replay algorithm jumps in and tests if the shot cut was due to the logo. As the logo fades into the screen, the algorithm skips a couple of images before it compares the next image to the logo template. When the logo is found, the program switches to replay mode. Most of the times the broadcast presents more than one replay. Still, after all replays are done, the logo fade is presented again. That is why we have to track for logo fades in order to end replay mode and thereby finish the event and replay detection.

In order to separate the replays from each other, the shot cut detection runs over the following images. As already described, the threshold that decides wether or not a match is found, is adapted due to bigger differences of the images in slow sections.

6.9 Video Features - Motion Analysis

When the first prototype of this framework and its output was analyzed, one of the bigger noticeable problems was that the time, that was shown before each event, was not adapted to the situation. In this sense the "event" itself is considered the moment of its occurrence, for example the goal itself.

Post event time is not considered important in that aspect. It can however get important if maximum time of the final highlight video plays a role. In that case, it can very easily be adapted. It can be set at the beginning by considering the event type, but does not need to change through any analyzation process other than shot cut detection (which is covered in the sections before). With that being said here is a closer look at pre-event time:

The time, that needs to be shown before a certain event, in order to give a user an idea over what has happened, differentiates a lot. Sometimes that can be helped by choosing the pre-event time manually beforehand, considering only the type of the event. When a penalty is shown for example, the situation before that event is usually not important. The standard pre-event time can therefore be limited to a smaller value. Other events, like goals and goal scoring chances require a longer look in the past to understand what has happened.

Skipping that history before the event, would make the final video a concatenation of shots on goal only. This problem however, is something that can be seen in a lot of manually cut highlight videos as well. The approach used in this work to solve this problem is the following:

As ice hockey is considered the fastest team sport in the world and speeds up to fifty kilometers an hour can be reached by skaters, an offensive attack, even if it is over the full span of the ice, will take no longer than maybe eight seconds. So even the full attack sequence can be put into the highlight video without a time problem. This has worked out to be far less confusing to a user, than starting a scene in the middle of an ongoing attack. The development of for example a goal, is something fans of the receiving team would also be interested in, where as the goal itself is most likely only of interest for the fans of the scoring team. Accordingly we want to look for the last vertical change of motion before the event.

Once that moment is found, another two seconds get added to the pre-event to let the user have a look at that situation also. In many cases this change of motion will actually be a turnover, meaning that the possession of the puck changes from one team to another. Of course we do not want the scene to start too far away from the actual event either, so a hard pre-event limit is set. The pre-event is supposed to last somewhere between four and fifteen seconds. Within those boundaries the algorithm searches for the overall vertical change of motion. [59] does also see turnovers as an important visual cue for the detection of shot boundaries. These event boundaries get then only altered a bit by the shot cut component.

The approach taken in this work to find a vertical change is described in the following:

At the beginning it has to be mentioned, that a sport broadcast video motion analysis has some special requirements. The main camera in broadcasts in ice hockey (and most other team sports) is at the center of the length side of the playing field with the goals at the very left and right. (Figure 26)

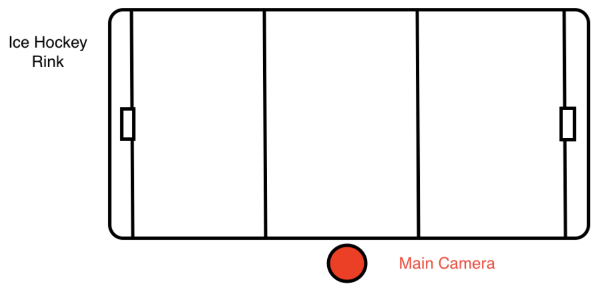


Fig. 26. Position of the main camera relative to the playing field in common ice hockey broadcasts

The camera is usually actively following the puck, either manually done by a cameraman or sometimes even steered by software [29]. So relative to the video, the puck is hardly moving at all. But by tracking the movement of surrounding players and also stationary obstacles, the flow of the puck can be estimated nicely.

The basic idea to detect good features to track, is to split an image into small windows. The next step is to find features that are as unique as possible, especially regarding the area around its window. So the easiest way to find a correct spot is a piece that contains a corner. Up to that we can expect large image variations in the neighborhood, in all directions of corners. These pieces of corners will then be searched for in around the area they were found in the last image. If this is done picture by picture, it is possible to track an object and even estimate an objects "flow".

The OpenCV library provides an algorithm for optical flow estimation that can be used for the implementation of these concepts. Optical flow is the

> .. estimation of velocity of image structures from one frame to another frame in time sequence of 2-D image. ...the 2D motion results from the projection of moving 3D objects on the image plane. The 2D projection is called apparent motion or optical flow. [41]

A similar motion flow method is explained in [7] and [9].

Optical flow is often used for video stabilization and compression. The estimation of the flow in this algorithm is done using the "Lucas-Kanade" method.

This method solves the basic optical flow equations by the least squares criterion [41], where the least squares criterion searches for a function that is closest to all points of data.

The implementation of this algorithm in OpenCV does use "Shi-Tomasi" corner detection to detect possible objects edges [50]. It is an improvement to Harris and Stevens [27], or Moravec algorithm [39] to detect corners, both of them first introduced in the 1980s, but they all work around the same simple principles: The algorithm takes out a small window of the image and looks if its features are equally found across the whole image. If so, the window moves ahead. If not, the algorithm probably found an edge.

By the help of the "Lucas-Kanade method", we now calculate the optical flow by saving the information inside those edges as an objects features and compare those features to similar regions in the next image, to finally get the optical flow of the object.

> The advantages and disadvantages of "Lucas kanade method" is: Advantages - this method is easy compare another method, very fast calculation and accurate time derivatives. Disadvantage - errors on boundaries of moving object... [41]

Figure 27 and 28 show the calculated motion vectors in a scene before a goal. In Figure 27 the puck moves along the boards from left to right.

The points are the positions of the detected edges and the line behind the point represents the way they have taken, since starting the estimation. The players are starting to move in the pucks direction. The camera, however is following the puck, which is moving faster than the players at that moment. So relative to the video, the players are moving to the left. Figure 28 clearly shows where the turnover takes place:

It is the moment when the camera stops and starts moving to the left. The players on the left are still skating to the right towards the puck. It can be seen that the algorithm finds a lot of edges on the boards. Those seem to be moving to the right as a consequence of the camera starting to move left.

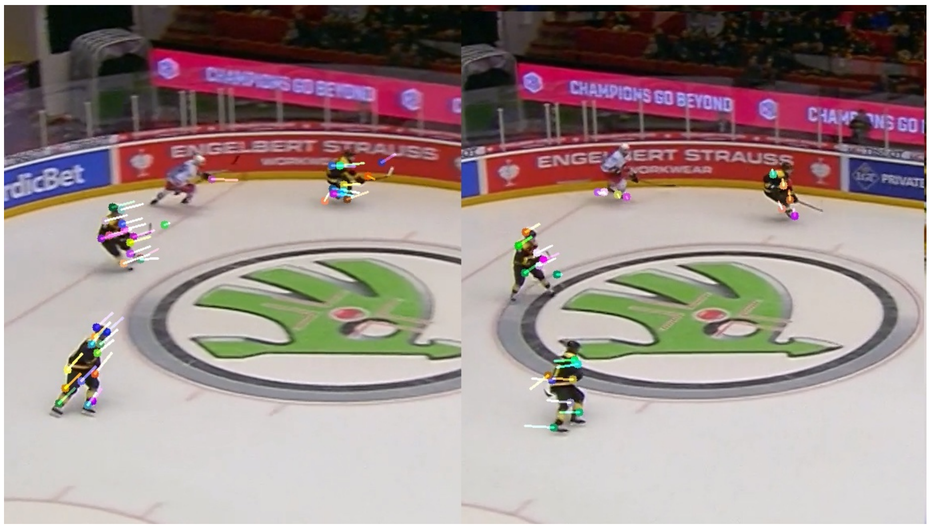


Fig. 27. Typical in game motion during the ice hockey game, that gets noticed by the algorithm

This works implementation does for reasons of optimization only consider every fifth image, which is no problem whatsoever for the quality of results. Furthermore one fifth of the borders around the images get cut away, since most of the information is captured in the middle of the image. The adapted algorithm does than return the sum and the average distance of all points for each image. For distance calculation the euclidean distance is measured. The maximum number of points is limited and a turnover is only considered as such, if a certain threshold is exceeded, since there is not always a clear turnover before every goal or goal scoring chance. In the implementation this process clearly states, that the highest motion happened in the before mentioned picture.

Listing 7 shows the algorithm used for motion detection. In this implementation, its structure is taken from a standard example, that is freely available in the docs. It is adapted as such, that only the inner part of the video is tracked, in order to optimize computing time and most of the information is captured in the middle of the image. At the end, a part that calculates the euclidean distance to the last point, is added to the algorithm, in order to compare the distances.

Listing 7. Motion algorithm is an adaption of the example implementation

```

# params for ShiTomasi corner detection
    feature_params = dict( maxCorners = 300,
```

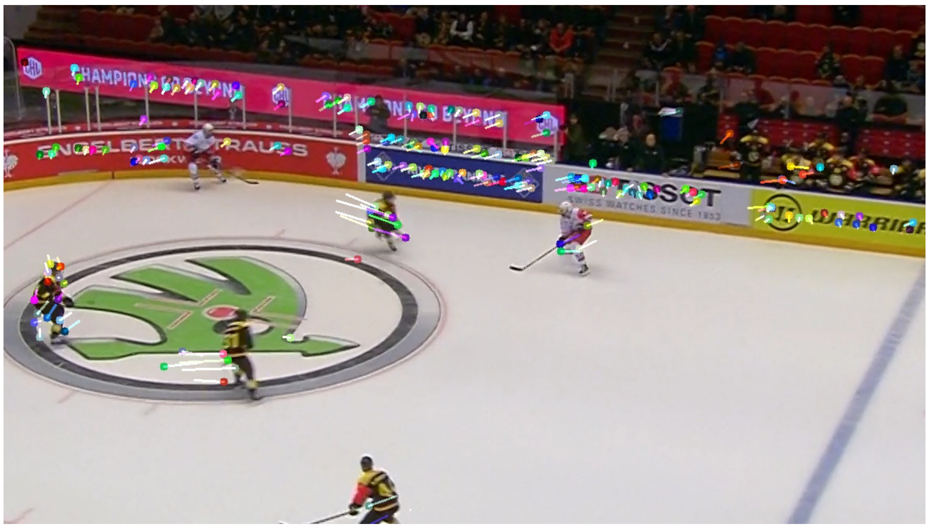


Fig. 28. Scene from Brynas vs Mannheim. Moment of highest motion in this sequence. It is the moment the turnover happens.

```
          qualityLevel = 0.3,
          minDistance = 7,
          blockSize = 7 )

# Parameters for lucas kanade optical flow
    lk_params = dict( winSize  = (15,15),
    maxLevel = 2,
    criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Take first frame and find corners in it
    old_frame=listofFrames[0]

    # nur inneren bereich vom video nehmen!
    old_frame = old_frame[int (width/5):int (width-width/5), int (height/5)
        :int (height-height/5)] # Crop from x, y, w, h -> 100, 200, 300,
        400
.
.
.
            # calculate distance between points vector b - vector und dann
                euclidean distance a^2+b^2=c^2
            v1=a-c
            v2=b-d

            d=math.sqrt(math.pow(v1,2)+math.pow(v2,2))
```

```

                fulldistance=fulldistance+d
```

Let us now take a look at two quick examples to illustrate how the proposed motion algorithm works: The pictures in Figure 29 and 30 show a goal in the matchup between Brynas IF and Adler Mannheim.

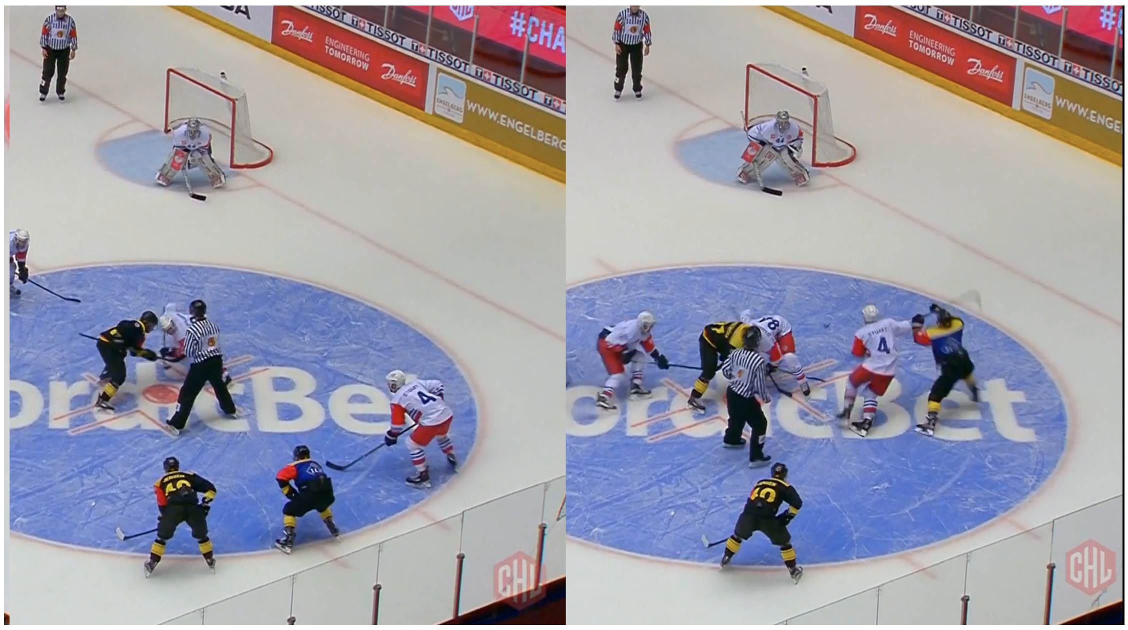


Fig. 29. Example 1: faceoff goal - short pre-event wanted 1/2

The scene starts with a face off in Brynas offensive zone. One of the offensive players is quicker at the bouncing puck than the defensemen. He plays a short cross pass to the middle and the receiving player scores. The goal is scored no more than two seconds after the play starts. In this case the algorithm detects the highest motion right after the face off, because before that, the players where standing almost completely still. As mentioned above, it adds up another two seconds, before that moments and saves that value for the pre-event boundary. Together with the play itself, that only makes approximately four seconds, that are shown before the goal which is, as mentioned, also the lower limit for scenes.

For each event type manual time settings exist. Within those time borders the algorithm analyzes the event. The second part calls the motion algorithm and

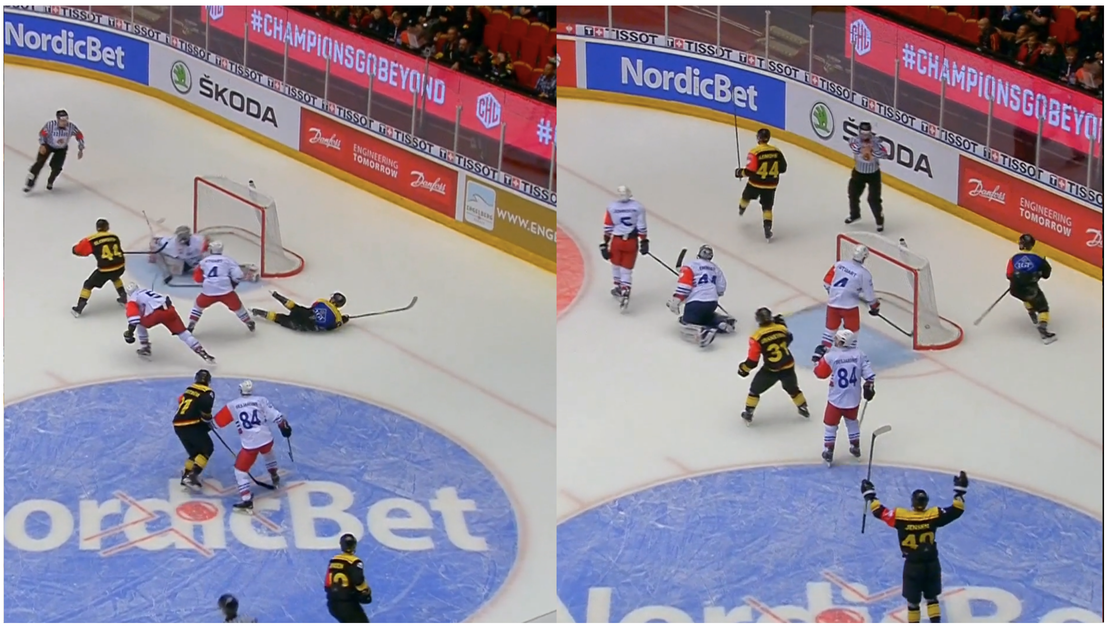


Fig. 30. Example 1: faceoff goal - short pre-event wanted 2/2

finally chooses the resulting time depending if it is at least four seconds before the event and the motion information.

Clearly it would not be helpful to have a longer pre-event span in the final video. The second example shows quite the opposite: The pictures in Figure 31 and 32 show the very next goal of the "Brynas IF" against "Adler Mannheim" game.

It is the last minute of the game and Mannheim (white jerseys) have pulled their goalkeeper for an extra skater, as they are in puck possession. In Figure 31 on the right part however, it can be seen that the player looses the puck in the corner, whereupon the opponent makes a quick pass and one of their players ends up scoring into the empty net (Figure 32).

In this case we want a longer pre-event setting, as it is better to show the development of the goal, instead of only showing a player skating down the ice with the puck before scoring into the empty net.

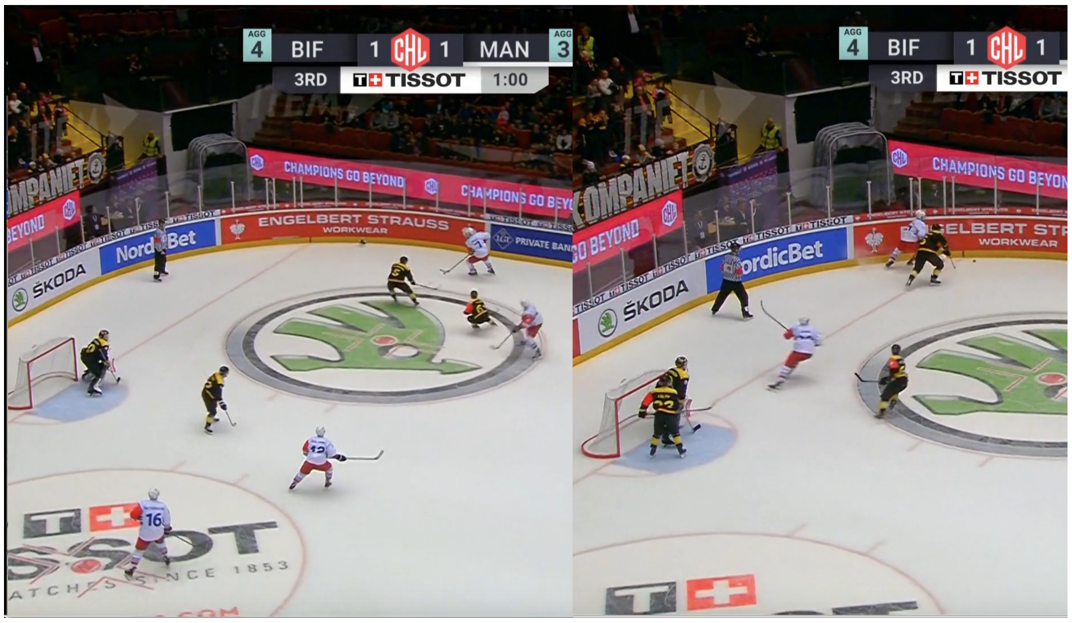


Fig. 31. Example 2: empty net goal - long pre event wanted 1/2

6.10 VideoCutting

The component of this part receives all event objects that are supposed to be part of the final video. They are already sorted timely and contain all metadata information as well as calculated information, like the event borders, or event scores. The time borders of their replays are also known, if existing.

Video fades for the time before the game, between the periods and after the game, were already extracted before the start of the program from an original "Champions Hockey League" highlight video, that is freely available on the video distribution platform "youtube" (Figure 33 and 34).

Next the audio of all clips gets manipulated by creating a fade in and fade out for all of them. Otherwise it would seem disturbing if one clip ends very quiet and the next clip starts loud. This provides better transitions and was also suggested by experts.

Then all events get concatenated and the video fades get put in between the event clips at the according times, again known from metadata extraction. If monitoring mode is enabled the text overlays get written over each clip.

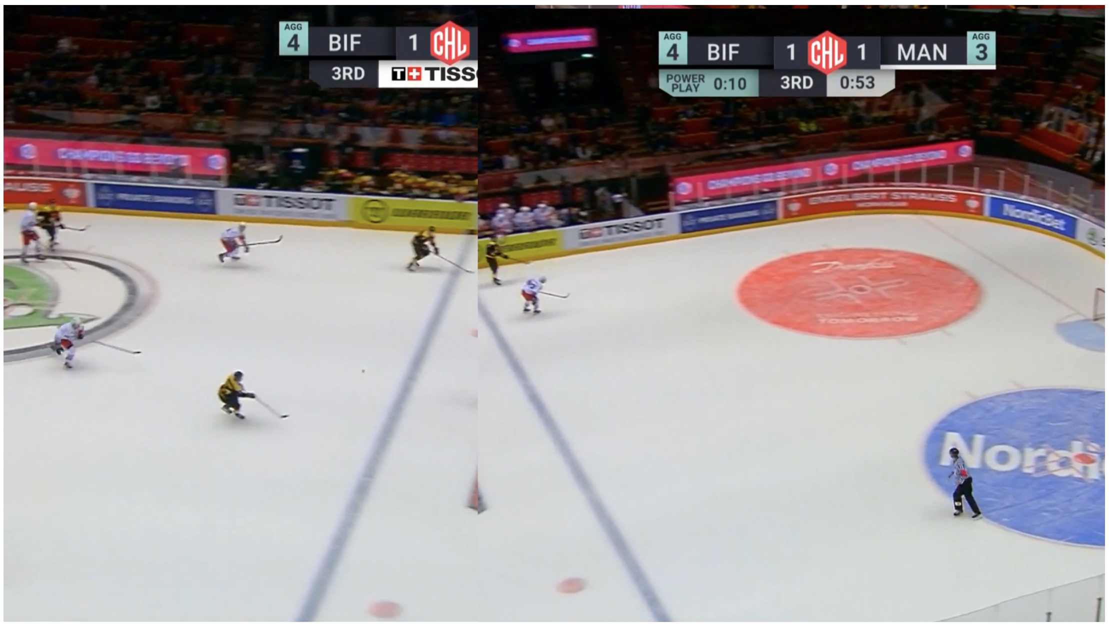


Fig. 32. Example 2: empty net goal - long pre event wanted 2/2

This part of the work is implemented using a library called MoviePy. It is easy to use and provides functions for practical things like "textOverlay" for example. Listing 8 shows code pieces from the creation of the audio fades, the overlay texts and the final concatenation of the videoclips.

Listing 8. Video cutting

```

        audioclip=afx.fx.audio_fadeout.audio_fadeout (clip.audio ,1.0)
        audioclip=afx.fx.audio_fadein.audio_fadein (audioclip ,1.0)
        clip=clip.set_audio(audioclip)




        if (monitoring):
            textclip=writeTextOverlay(clip, event.get_text())





            # Overlay the text clip on the first video clip
```

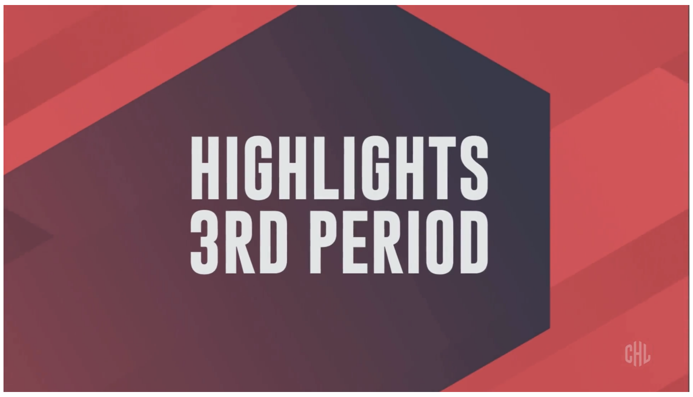


Fig. 33. Example picture of an video fade that is found in official ice hockey highlight videos from the CHL

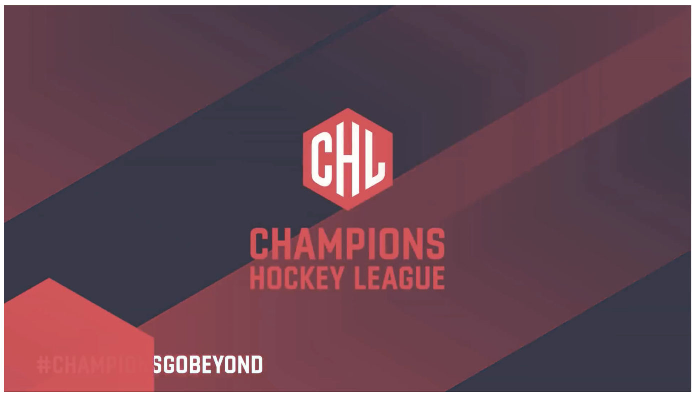


Fig. 34. Videofade example picture start and end

```
                    clip = CompositeVideoClip([clip, textclip]).set_duration(
                        end-begin)




    if cliplist:
        video= concatenate.concatenate_videoclips(cliplist, method="compose
            ")
        videolist.append(video)
```

7 Conclusion

This sections summarize the work of this thesis and discusses results, limitations and possible usages. A short experimental evaluations is done on the results and a critical reflection is provided. Finally, in the Outlook section **??**, it then provides detailed information on how to extend and improve the frameworks' concepts and implementations.

7.1 Results and Experimental Evaluation

The produced videos clearly showed that the discussed goals could be achieved. A fully working prototype program, that can automatically produce a highlight video from user input, could be implemented. It does not need any additional user input other than the input video file and the metadata file. It has all mentioned concepts and methods implemented and can be easily extended by various methods by adapting certain parts.

For a brief experimental evaluation some questions were asked to experts from the video domain to provide some insight in technical and semantic quality of the highlight video:

Are all technical requirements fulfilled? Are the shot cuts, transitions and event boundaries in the final video set correctly? Are all highlights included? Is the videos composition according to the timeline? Is it possible for a user to verify which team scored, when they scored and what the score is? Is the storyline of the video understandable? What is the overall impression? Are the chosen scenes relevant? Are the video and audio fades implemented correctly?

The set of videos that was available for this work is already described in section 4 Approach. The videos did also get compared to their manually created counterparts. Some requirements can be checked manually. A small expert team tried to answer the rest of the questions.

In qualitative evaluation the experts stated that in practice sometimes there is a hard upper limit of video length. In order to at least show all goals while still holding that limit, they would usually do one or all of the following:

– leave out scenes other than goals

- shorten the length of the video after events
- cut down the amount of replays (maximum one per event)
- shorten pre-game footage

If there are very few good scenes available, it is of course the other way around. That is something that could definitely be more adapted in this work.

In regards to content, the choice of scenes was compared to the manually created videos. According to the experts, the videos were on a very high level content wise and sometimes even included scenes that the team that cut manually missed out, but definitely should have been in the final video.

They also mentioned that the scenes that the audio component of this work ranked high, were mostly from the third period. This is probably because the fans naturally react louder and more excited at the end of games, especially when the game is tight. This however could be adapted by for example weighting the audio rating relative to the match time.

The motion analysis, that is responsible for the start of the event boundary, worked exceptionally good, could however be a little bit of an issue once the maximum length of the video is reached and time needs to be saved in order to show more scenes.

A bigger issue is the fact, that it is very hard to detect a replay after any events that do not stop the game immediately. There are time stamps in the metadata whenever the game stops and that is where replays are probably going to be played. However, it is not certain what scene that replay is going to show.

As a consequence of leaving those replays out of the video and switching to the next event immediately, there are sometimes two far shots after each other, which is not considered good practice.

Possible solutions are looking for that replay anyway, although we cannot be sure it is the one we are looking for. If we only look within the next minute of the game time or so, the probability that there was no other replay worth scene in between is high. This would have to be tested. Sometimes there are no replays available at all. In this case, experts do put in short view shots from the audience in between the scenes. Those could be extracted by shot type classification for example. Sometimes it might be enough to show the event a little longer, until the broadcast brings up a shot type change on its own that can be used immediately.

Another non optimal part in the created videos was the first scene. In the created videos and in many manually created videos the first face off of a game is shown. That is however not a relevant scene and is in good videos usually replaced by warm up scenes from important players with graphical inputs, if available.

This could be implemented by searching for that graphical inputs before the game, for example by template matching. A disadvantage is that this is really computing intensive, as long as we do not approximately know when the graphical inputs are going to be shown. Furthermore, in order to understand which player is shown, the graphic has to be read out, for example by using methods like in [6].

At the end a look at some smaller issues:

The audio fades are not implemented totally correctly. The audio gets turned off slowly until it reaches zero and then the next audio clip is slowly turned on. However completely correct would be to let the sounds overlap a little bit.

Before replays, the logo fade was included in the produced videos most of the time. However, sometimes the algorithm missed the logo. It is good practice to not include it at all. If it is included then it should be consistently in there. In this work it would basically be necessary to use the cut detection again for that issue.

In the videos that had commentary in it, a speech detection could have been implemented in order to not cut when the commentator is speaking.

Some minor adjustments could also be done, like one false frame at a cut or a goal celebration that was shown a little bit longer than others. A human cutter would probably eliminate those at the end when he is doing the quality control.

The qualitative part of the evaluation could be done before the interview: The thresholds were set according empiric knowledge of specialists. All of the created videos could fully satisfy the restrictions that there are no shots shorter than one second and no scenes are shorter than four seconds, just like specified in the sections above.

More videos would have to be analyzed to get a significant result, as by the time of the evaluation, unfortunately only the four specified videos and their corresponding meta data files were available for ice hockey.

Due to the high amount of events to analyze (lots of shots on goal in ice hockey) and the high amount of events in the final video (many goals compared to for example soccer), the composition can take up to over 20 minutes on a desktop computer.

In a side project for the evaluation, this framework was adapted to the needs of soccer. This shows the generic nature of the framework, and provided a way to test it with more videos. Broadcasts and metadata from the German 3. Soccer League was used. Because there were fewer events to analyze, those highlight compositions were much faster, and therefore finished in around 10 minutes.

All future considerations and improvement possibilities of current problem areas are stated in the following detailed Outlook section (7.2).

7.2 Future Work

In many parts of the work optimizations can be implemented. Either by improving the implemented algorithms and techniques, or by adding completely new methods. This section shall give a quick overview of possibilities:

7.2.1 Meta Data As mentioned in section 6.4 about the event calculation, timeouts and similar events are hard to implement in the final video, because of inconsistent timestamps in the meta data. A solution is of course, as done in a lot of research papers, to analyze and classify the shot cuts around the event and, by adding knowledge about domain specific cutting habits, find good event boundaries for the video. Also the amount of replays in the broadcast (or if there is a replay at all), could help in those cases, but as they are not always played exactly after the event, they are hard to assign to the right event [61].

The metadata text file provides a lot, but not all information about the game. It would be desirable to extend the knowledge further. As mentioned, various works do fetch data in real time from online web casts [62].

That is similar to this approach with the difference that those web texts usually do not provide any accurate timestamps for the events. However implementing this exactly would make it possible to query other event information, that was not included in the metadata files. In our case that is for example the type of penalty a player receives. A fighting major penalty would be really interesting for the final highlight and it would make it easy to rate a fighting penalty higher than a tripping penalty.

Another option would be to fetch career data of any player from the internet. For ice hockey this would be possible over a REST API provided by "eliteprospects.com", which is the standard player database for ice hockey worldwide. In addition to that, a nice idea would be to let a text with the goal scorers' statistics appear when a goal is shown in the final video.

Another drawback concerning the provided metadata was, that a lot of events were timestamped a little bit off time. It follows that the events happen two to four seconds before the timestamp. This of course makes it more difficult to compare the audio data as we have a window of two seconds of uncertainty.

7.2.2 RuleSets As previously stated, shot cut type patterns are used in several works to find important events independently of metadata information. These cues are not extensively used in this work, but could help to increase quality of the score calculation and save computing time. [17] for example claims that cut rates do specifically correlate with excitement in sport videos, which could be worth a try. Others do analyze shot type pictures over a lot of videos, to get an exact idea of the patterns.

The limitations of this approach, on the other hand are, that those analysis would probably work for games of the CHL 2017/2018 season. It is however questionable if they will also work for the year after, or the year before, since the patterns are not standardized anywhere. In order to avoid that, it would be necessary to analyze a huge amount of games over several international leagues, to get a common picture of the patterns.

As mentioned in the regarding sections, a way to extend the rule sets is to implement them as an ontology. After adding new rules, the ontology reasoner could also detect any further complex relations to create new rules. When the rule set grows bigger it would be better to put them in a structured file, or in an ontology and then generically read them into the program and react to them from the methods. That way it would also be easier to further extend or alter the rules, especially if the alteration is done by a non computer expert.

7.2.3 Shot Cut Detection Shot cut detection in this work does use a fixed threshold to decide whether a shot cut is found or not. Only when the system is looking at a replay scene the threshold gets adapted. [18] and others do use adaptive thresholding of the histogram thresholds to be even more accurate the longer the program goes. This could make the algorithm significantly more robust, since it does not rely on the video quality anymore. However in this works approach there was almost no situation were a shot cut could not be detected.

7.2.4 Time Synchronisation In the current version, time synchronization needs a video starting point to work accurately, due to time differences of a couple of seconds between different computer systems. As described in the according section there are several possible solutions for that, although defining a manual entry point is not uncommon at all according to specialists. The easiest way to get rid of the problem is of course to save the broadcast and create the highlight video on the same computer or even real time. In this way synchronization is no problem whatsoever and does only require to read out the timestamps. Other solutions would be possible, but exceptionally more complex.

In [6] a way of reading out the scoreline (including the remaining playing time) from a broadcast video is presented. Also background template comparison with certain situations before the game together with motion analysis would be a possible idea. The reason for that is that big sports leagues provide a certain script called "run down" to tell the broadcaster what exactly to show before the games. If we now know that they are supposed to show the league logo at full screen at approximately 20 sec before the start of the game, we could detect it and then 20 seconds later run a motion analysis to detect the start of the game (as in ice hockey players must stand still before the face off).

7.2.5 Audio Analysis For audio analyzation several other improvements can be made: Unfortunately it was very difficult to detect significant changes in audio in most away team events. Firstly because, especially in those European, international games that were used, there are very few away team fans at a rink. This of course would be better if national wide games would be considered only.

Lastly, especially in European sports, fans tend to support and cheer for their team for long periods, no matter if there is a special situation on the field, or not.

Working with north American games would make it a lot easier, since the culture differs a lot from the European and people mostly react to important events.

To choose the best features and weight them correctly, of course machine learning techniques will work more correctly than approaches of this work. For example could it be tried to cluster the samples with the "k - nearest neighbor" algorithm, or to run an optimization algorithm to weight all features.

Most of the related works like [45] and [58], however do use supervised learning, which would make this more time consuming, since someone needs to define the samples as good or bad. Up to that, generally a bigger amount of samples would be needed to run machine learning techniques. Still it would be worth it as there are so many features, that could be used and easily extracted and did not seem to have a meaning when they were tested without machine learning.

Another idea would be to create common template samples, or signatures for sounds like whistling, drumming or fan singing, that could eventually be filtered out afterwards. A possible way would be to use harmonic features as those sounds seem repetitive and harmonic in contrast to others like goal cheering. Within this work it was assumed that the pitch at important events is also higher than usual. This might not be true for all possibilities and should be further analyzed.

7.2.6 Overall Small adaptions can also make the framework usable in various different ways: Like [18] describes, highlights might be adapted to special users by changing video events and/or video length [31]. A solution for a user driven highlight video would therefore be, as already mentioned, to let the user specify some parameters before the start of the score calculation process, in order to take care of the users preferences. He could be a fan of one of the teams and hence receive more scenes from that team for example.

Another possibility would be to provide chosen events to the user, just before cutting them together and in this way let him choose which of those clips he wants in his final video and which of them seems not to be necessary to him or her. This would lead to a semi automatic highlight video generation where a user could make adaptions with minimal effort and is also separately proposed by experts during the evaluation process.

[62] also discusses scenarios of how to deploy the proposed solution into various devices and at real time. This way the synchronization process would be simplified and the finished highlight video could be provided on the fly, a couple of minutes after the real game has finished. The adapted framework could for that reason already calculate event boundaries of priority 1 events as they happen and finish the video within a few minutes after the game.

To fully analyze a video with a lot of goals, the program can take twenty to thirty minutes. This seems rather long, but is actually pretty fast, compared to a manually cut video. On the other hand, optimization could help making it a lot faster. Faster algorithms and not running the program over redundant data (for example, usually analyzing every second image would be enough), are just the easiest of many possibilities to do so. This will especially be required, because before mentioned future adaptions would need additional analyzation and would therefore make running time longer.

In the evaluation section several other future improvements were mentioned. In case of a time limit for the highlight video, improvements regarding the composition of the video can be made. Whenever time is spare, the amount of replays, amount of priority 2 scenes as well as the time after events, have to be shortened accordingly. If there are not many goals, the mentioned parts can be made longer.

Next it would make sense to make the audio analyzation relative to the match time, because observations showed that fans react more excited and louder, the longer the match takes. The same event would therefore get different ratings depending on the match time. Hence it could be tried to weight the rating by using the match time.

The problem of having two far shot cuts after each other, because of missing replays, can be addressed in several ways. Searching for the replay in the next game break makes sense, but the question, if the replay is showing the right scene, is hardly answerable. Waiting for a shot cut after a scene is easy, but it is not guaranteed that it is going to happen.

The most reliable solution is two put in a short sequence of a close up from fans, from somewhere within the broadcast. This could be done by shot type classification. The first scene in the result video can be exchanged by scenes showing the players in warm ups. During the broadcasts there are usually graphical inserts that could be detected by template matching.

7.3 Discussion and Critical Reflection

Big data is the term used for the sheer amount of huge datasets available in any domains nowadays, that are too big to analyze traditionally. Decent video equipment is getting cheaper and better and therefore the big data problem has also reached to video domain. In order to make use of these already existing immense amount of videos, generating a automatically created summary might be a way to go.

This work has shown that it is possible to fully automatically create a highlight video from a broadcast and event metadata extracted from live feeds, that is adaptable to user wishes and robust against all kinds of different event types in an ice hockey game.

Results of tests on various different ice hockey broadcast videos have proven that automatic video highlight generation is possible in a quality, that is very close to hand made videos. I believe that, by implementing the suggestions provided in section 7.2 Future Work, this small gap can be further reduced.

While many similar works do not use metadata, it is obvious that by using event metadata, the computing calculation effort can be drastically reduced and the quality significantly raised. Keep in mind that the metadata used for this work is not tailored for the task, but rather already existing data that can be reused.

With minor adaptions the framework can be used to analyze various other team sports as it is generic in general. However sport specific knowledge gives important insights and therefore to become a generic framework for all sports, the base rule set that is, as explained, very domain specific has to be adapted according to the new sport. In the evaluation phase this could be proved by adapting the system in a way that it is able to handle soccer broadcasts as the input. It was also possible to deploy the framework on a cloud service in order to shift computation from the local device. Instead of the User Interface, a simple RESTful Web Application provides the user with the possibility to choose the correct files.

That is why it can be expected that frameworks like this can be really helpful in practice. The use case might reach from pre selecting honorable scenes from a sport game and presenting them to users, to sparking the whole power of the system and creating and cutting full highlight videos all together. Looking at the single steps offers a good insight that some of the tasks on the way to creating the final video are predestinated to being automated by a computing system. In any case the described framework could therefore be helpful to save time for people to concentrate on the really creative and important steps, that are hard to accomplish by a machine.

7.3.1 Metadata Meta data extraction and analyzation was pretty straight forward. The harder part was to analyze the structure and detect a pattern throughout the files. Sometimes that led to complex algorithms in order to get the exact timestamp of an event, because it was mentioned several times after each other. As already mentioned, the data could have been more detailed in some parts, like the one from the NHL play-by-play statistics. They also provide more secondary information about the events, for example all statistics for the goalscorer. This is especially crucial as this part of the framework is the least computing intensive. Any improvement in this part has incredible impact on the overall systems performance and quality. In this work for example the timestamps of shots on goal were sometimes imprecise, so that the system had to run its algorithms over a larger part of the video in consequence.

7.3.2 Shot Cut Detection Shot cut detection was implemented with histogram comparison and worked very solid. All important cuts were always detected. Improved algorithms could of course make the system even more robust.

Replay detection in this case was easier than expected since all replays were introduced by a logo fade. By the help of background comparison this was detected. This method worked very stable also.

7.3.3 Audio Analysis Judging the results of the audio rating component is hard. On one hand it would have been possible to rate the fans excitement by the loudness alone too, on the other hand the combination of audio features could also be far more complex and therefore provide better results. The implemented method finally deletes events, that did score well below the average of the audio feature. This is a rather simple approach, that can be smoothed by using some function for the weighting of the score.

However the most simplistic goal for the audio extraction in this work was to distinguish between good and bad events, which certainly succeeded. Mostly good and very good events could be figured out too, but in order to consistently accomplish that, it would probably require to run machine learning techniques over audio samples. In this way one could divide good and bad features and eventually more of the available features could be used in the first place. A problem that was not in the focus at the start, is rating events from away teams by audio differently, if there are no fans of that specific team at the game.

7.3.4 RuleSets The composition and rule set for the final video can be extended in a couple of ways. It should take more information about the length of the video into account, for example by reacting to the amount of goals.

7.3.5 Time Synchronisation Due to in-precise computer clocks, time synchronization is only approximately possible in the current implementation, as long as broadcast video and highlight video are not composed on the same machine. One way to solve this problem would be to detect a certain starting point in the broadcast video.

7.3.6 Overall Summing it up all together, the goals of this work could be achieved, although various new questions and problems opened up in many different areas. Because of the considerably large range of the whole framework, it was hardly possible to optimize all algorithms and methods to a further point. However, it could be shown by the results that by the help of timestamped play by play data and multimedia techniques, a very decent highlight video could be created. Sometimes, the algorithm did even choose important scenes of a game, that were missed out on the original video, that was made by professionals. Many algorithms and techniques implemented in this work can be optimized by more sophisticated techniques. For this work however, the cost-benefit ratio in terms of time and complexity was not good enough to always choose the top level algorithms. Nevertheless the experiments showed, that the implemented algorithms are well enough able to complete their tasks with minimal error rates. It has to be said that the basis for the implementation were only four full videos, so even tough it was tried to implement all methods as general as possible, over-generalization could be an issue especially regarding the weightings of features.

Of course composing a creative video as good as a manually created one is probably not consistently possible. It is never guaranteed that the meta data and/or the composited scenes are correct and are what they are supposed to be. Also, rare controversial scenes can not be part of a rule based composition.

While the question remains if a fully automatically generated highlight video can consistently perform better than a manually created video, it could be proven that such a framework can be extremely helpful in the creation process, for example by choosing events beforehand, by detecting event boundaries and many more.

This would probably be one of the more convenient ways to use the framework created in this thesis. Because at the end of the day the final goal should not be to create without human input at all. Or as Satya Nadella, the CEO of Microsoft put into into words:

> "Ultimately, it's not going to be about man versus machine. It is going to be about man with machines [9]."

[9] https://www.financialexpress.com/industry/technology/ultimately-its-going-to-be-man-with-machines-microsoft-ceo-satya-nadella/231281/

References

[1] URL: http://www.zdnet.com/article/ibm-watson-is-creating-highlight-reels-at-the-masters/.

[2] Muhammad Ajmal et al. "Video Summarization: Techniques and Classification". In: *Computer Vision and Graphics: International Conference, ICCVG 2012, Warsaw, Poland, September 24-26, 2012. Proceedings*. Ed. by Leonard Bolc et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 1–13. ISBN: 978-3-642-33564-8. DOI: 10.1007/978-3-642-33564-8_1. URL: https://doi.org/10.1007/978-3-642-33564-8_1.

[3] R.G. Bachu et al. "Voiced/Unvoiced Decision for Speech Signals Based on Zero-Crossing Rate and Energy". In: *Advanced Techniques in Computing Sciences and Software Engineering*. Ed. by Khaled Elleithy. Dordrecht: Springer Netherlands, 2010, pp. 279–282. ISBN: 978-90-481-3660-5.

[4] Anant Baijal et al. "Sports highlights generation based on acoustic events detection: A rugby case study". In: *CoRR* abs/1509.06279 (2015). URL: http://arxiv.org/abs/1509.06279.

[5] MÜJDAT BAYAR. "EVENT BOUNDARY DETECTION USING WEB-CASTING TEXTS AND AUDIO-VISUAL FEATURES". PhD thesis. THE GRADUATE SCHOOL OF NATURAL and APPLIED SCIENCES OF MIDDLE EAST TECHNICAL UNIVERSITY, 2011.

[6] Vinay Bettadapura, Caroline Pantofaru, and Irfan Essa. "Leveraging Contextual Cues for Generating Basketball Highlights". In: *Proceedings of the 2016 ACM on Multimedia Conference*. Amsterdam, The Netherlands: ACM, 2016, pp. 908–917. ISBN: 978-1-4503-3603-1. DOI: 10.1145/2964284.2964286. URL: http://doi.acm.org/10.1145/2964284.2964286.

[7] Aaron F. Bobick and James W. Davis. "The Recognition of Human Movement Using Temporal Templates". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 23.3 (Mar. 2001), pp. 257–267. ISSN: 0162-8828. DOI: 10.1109/34.910878. URL: http://dx.doi.org/10.1109/34.910878.

[8] Gary R. Bradski. "Computer Vision Face Tracking For Use in a Perceptual User Interface". In: *Intel Technology Journal* (Feb. 1998).

[9] Gary R. Bradski and James W. Davis. “Motion segmentation and pose recognition with motion history gradients”. In: *Machine Vision and Applications* 13 (2000), pp. 174–184.

[10] Sung-Hyuk Cha and Sargur N. Srihari. “On measuring the distance between histograms”. In: *Pattern Recognition* 35.6 (2002), pp. 1355–1370. DOI: 10.1016/S0031-3203(01)00118-2. URL: https://doi.org/10.1016/S0031-3203(01)00118-2.

[11] C. H. (Chi-Hau) Chen. *Statistical pattern recognition.* English. Includes bibliographies. Rochelle Park, N.J. : Hayden Book Co, 1973. ISBN: 0876711778.

[12] H. D. Cheng et al. “Color image segmentation: Advances and prospects”. In: *Pattern Recognition* 34 (2001), pp. 2259–2281.

[13] *chl regulations 2017-18 pdf.* URL: https://res.cloudinary.com/chl-production/image/upload/v1504437509/chl-prod/assets/CHL_Sport_Regulations_2017-18.pdf.

[14] *cognitive-movie-trailer.* URL: https://www.ibm.com/blogs/think/2016/08/cognitive-movie-trailer/.

[15] D. Comaniciu, V. Ramesh, and P. Meer. “Real-time tracking of non-rigid objects using mean shift”. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No.PR00662).* Vol. 2. Hilton Head Island, SC, USA: IEEE Comput. Soc, 2000, pp. 142–149. ISBN: 0-7695-0662-3. DOI: 10.1109/CVPR.2000.854761. URL: http://dx.doi.org/10.1109/CVPR.2000.854761.

[16] Patricio Cordova. “Human Whistle Interface”. In: *University of Toronto* ().

[17] Thomas G. Dietterich. “Ensemble Methods in Machine Learning”. In: *Proceedings of the First International Workshop on Multiple Classifier Systems.* MCS ’00. London, UK, UK: Springer-Verlag, 2000, pp. 1–15. ISBN: 3-540-67704-6. URL: http://dl.acm.org/citation.cfm?id=648054.743935.

[18] A. Ekin, A. M. Tekalp, and R. Mehrotra. “Automatic Soccer Video Analysis and Summarization”. In: *Trans. Img. Proc.* 12.7 (July 2003), pp. 796–807. ISSN: 1057-7149. DOI: 10.1109/TIP.2003.812758. URL: http://dx.doi.org/10.1109/TIP.2003.812758.

[19] Angelika Erhardt. *Einführung in die digitale Bildverarbeitung : Grundlagen, Systeme und Anwendungen ; mit 35 Beispielen und 44 Aufgaben.* Wiesbaden: Vieweg + Teubner, 2008. ISBN: 9783519004783 351900478X 9783834895189 3834895180. URL: `http://www.amazon.de/Einf%C3%BChrung-Die-Digitale-Bildverarbeitung-Anwendungen/dp/351900478X`.

[20] Stephen Ferg. *Thinking in Tkinter.* July 2005.

[21] W. Fischer. "Digitale Fernseh- und Hörfunktechnik in Theorie und Praxis". In: *Springer-Verlag Berlin Heidelberg* (2015).

[22] Ligue internationale de hockey sur glace. *Official Rule Book: Adopted by the IIHF Congress April 1981, Gothenburg, Sweden.* IEHV, 1981. URL: `https://books.google.at/books?id=DrormgEACAAJ`.

[23] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing (2nd Ed).* Prentice Hall, 2002.

[24] François Goudail, Philippe Réfrégier, and Guillaume Delyon. "Bhattacharyya distance as a contrast parameter for statistical processing of noisy optical images". In: *J. Opt. Soc. Am. A* 21.7 (July 2004), pp. 1231–1240. DOI: `10.1364/JOSAA.21.001231`. URL: `http://josaa.osa.org/abstract.cfm?URI=josaa-21-7-1231`.

[25] Pan Hao, J. L. van Beek Peter, and Sezan M. Ibrahim. "Detection of slow-motion replay segments in sports video for highlights generation". In: *IEEE International Conference on Acoustics, Speech, and Signal Processing, ICASSP 2001, 7-11 May, 2001, Salt Palace Convention Center, Salt Lake City, Utah, USA, Proceedings.* 2001, p. 1649. DOI: `10.1109/ICASSP.2001.941253`. URL: `https://doi.org/10.1109/ICASSP.2001.941253`.

[26] Hadi Harb and Liming Chen. "Highlights Detection in Sports Videos Based on Audio Analysis". In: *Third International Workshop on Content-Based Multimedia Indexing, CBMI03.* Rennes, France, Sept. 2003, pp. 1–7. URL: `https://hal.archives-ouvertes.fr/hal-01587119`.

[27] Chris Harris and Mike Stephens. "A combined corner and edge detector". In: *In Proc. of Fourth Alvey Vision Conference.* 1988, pp. 147–151.

[28] Berthold K. P. Horn and Brian G. Schunck. "Determining Optical Flow". In: *ARTIFICAL INTELLIGENCE* 17 (1981), pp. 185–203.

[29] M. C. Hu et al. “Robust Camera Calibration and Player Tracking in Broadcast Basketball Video”. In: *IEEE Transactions on Multimedia* 13.2 (Apr. 2011), pp. 266–279. ISSN: 1520-9210. DOI: 10.1109/TMM.2010.2100373.

[30] Kuo C. Jay. *Video Content Analysis Using Multimodal Information: For Movie Content Extraction, Indexing and Representation*. Norwell, MA, USA: Kluwer Academic Publishers, 2003. ISBN: 1402074905.

[31] Kai Jiang, Xiaowu Chen, and Qinping Zhao. “Automatic Compositing Soccer Video Highlights with Core-Around Event Model”. In: *2011 12th International Conference on Computer-Aided Design and Computer Graphics* (2011), pp. 183–190.

[32] Hongliang Jin et al. “Replay Detection in Broadcasting Sports Video”. In: *Image and Graphics, International Conference on* 00 (2004), pp. 337–340. DOI: doi.ieeecomputersociety.org/10.1109/ICIG.2004.124.

[33] Lawrence A. Rowe John S. Boreczky. “Comparison of video shot boundary detection techniques”. In: *Journal of Electronic Imaging* 5 (1996), pp. 5–7.

[34] P. K. Kaiser and R. M. Boynton. *Human Color Vision*. Ed. by Washington D.C. 2nd ed. Optical Society of America, 1996.

[35] James M. Kasson and Wil Plouffe. “An Analysis of Selected Computer Interchange Color Spaces”. In: *ACM Trans. Graph.* 11.4 (Oct. 1992), pp. 373–405. ISSN: 0730-0301. DOI: 10.1145/146443.146479. URL: http://doi.acm.org/10.1145/146443.146479.

[36] Beth Logan. “Mel Frequency Cepstral Coefficients for Music Modeling”. In: *In International Symposium on Music Information Retrieval*. 2000.

[37] David L. Mills. “Internet Time Synchronization: the Network Time Protocol”. In: *IEEE Transactions on Communications* 39 (1991), pp. 1482–1493.

[38] L.F.M. Morante. *Editing and Montage in International Film and Video: Theory and Technique*. Routledge, 2017. ISBN: 9781138244085. URL: https://books.google.at/books?id=zWctvgAACAAJ.

[39] Hans Peter Moravec. “Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover”. AAI8024717. PhD thesis. Stanford, CA, USA, 1980.

[40] Asgeir Mortensen et al. “Automatic Event Extraction and Video Summaries from Soccer Games”. In: *Proceedings of the 5th ACM Multimedia Systems*

Conference. MMSys ’14. Singapore, Singapore: ACM, 2014, pp. 176–179. ISBN: 978-1-4503-2705-3. DOI: 10.1145/2557642.2579374. URL: http://doi.acm.org/10.1145/2557642.2579374.

[41] Dhara Patel and Saurabh Upadhyay. “Article: Optical Flow Measurement using Lucas Kanade Method”. In: *International Journal of Computer Applications* 61.10 (Jan. 2013). Full text available, pp. 6–10.

[42] Ken C. Pohlmann. *Principles of Digital Audio*. 4th. McGraw-Hill Professional, 2000. ISBN: 0071348190.

[43] Kari Pulli et al. “Real-time computer vision with OpenCV.” In: *Commun. ACM* 55.6 (2012), pp. 61–69. URL: http://dblp.uni-trier.de/db/journals/cacm/cacm55.html#PulliBKE12.

[44] Edward Rosten and Tom Drummond. “Machine Learning for High-speed Corner Detection”. In: *Proceedings of the 9th European Conference on Computer Vision - Volume Part I*. ECCV’06. Graz, Austria: Springer-Verlag, 2006, pp. 430–443. DOI: 10.1007/11744023_34. URL: http://dx.doi.org/10.1007/11744023_34.

[45] Yong Rui, Anoop Gupta, and Alex Acero. “Automatically Extracting Highlights for TV Baseball Programs”. In: *Proceedings of the Eighth ACM International Conference on Multimedia*. MULTIMEDIA ’00. Marina del Rey, California, USA: ACM, 2000, pp. 105–115. ISBN: 1-58113-198-4. DOI: 10.1145/354384.354443. URL: http://doi.acm.org/10.1145/354384.354443.

[46] Ulrich Schmidt. *Professionelle Videotechnik : Grundlagen, Filmtechnik, Fernsehtechnik, Geräte- und Studiotechnik in SD, HD, DI, 3D*. Berlin u.a.: Springer Viewig, 2013. ISBN: 978-3-642-38991-7.

[47] Gottfried Schröder. *Technische Optik : Grundlagen und Anwendungen*. Kamprath-Reihe. Vogel Buchverlag, 1998.

[48] Cheng-ya Sha. “Time Frequency Analysis for Acoustics”. In: *Graduate Institute of Communication Engineering National Taiwan University, Taipei, Taiwan, ROC* ().

[49] D. S. Shete and Prof. S. B. Patil. “Zero crossing rate and Energy of the Speech Signal of Devanagari Script”. In: IOSR Journal of VLSI and Signal Processing (IOSR-JVSP), Jan. 2014.

[50] Jianbo Shi and Carlo Tomasi. "Good Features to Track". In: 1994, pp. 593–600.

[51] Carsten Steger, Markus Ulrich, and Christian Wiedemann. *Machine Vision Algorithms and Applications*. ISBN: 3527407340, 9783527407347.

[52] O. Straka and M. Simandl. "Using the Bhattacharyya distance in functional sampling density of particle filter". In: vol. 1. Oxford: Elsevier, 2006, pp. 1006–1011. ISBN: 0-08-045108-X. URL: `http://www.kky.zcu.cz/en/publications/StrakaO_2006_Usingthe`.

[53] Tilo Strutz. *Bilddatenkompression: Grundlagen, Codierung, JPEG, MPEG, Wavelets*. Springer, 2002.

[54] Richard Szeliski. *Computer Vision*. Springer, 2011.

[55] Christopher Schultz Thorsten Hermes. "Automatic Generation of Hollywood-like Movie Trailers". In: *Technologie-Zentrum Informatik, Universitat Bremen* ().

[56] Olegs Verhodubs. "Ontology as a Source for Rule Generation". In: *CoRR* abs/1404.4785 (2014). arXiv: `1404.4785`. URL: `http://arxiv.org/abs/1404.4785`.

[57] Jinjun Wang et al. "Automatic composition of broadcast sports video". In: *Multimedia Systems* 14.4 (Sept. 2008), pp. 179–193. ISSN: 1432-1882. DOI: `10.1007/s00530-008-0112-6`. URL: `https://doi.org/10.1007/s00530-008-0112-6`.

[58] Jinjun Wang et al. "Automatic composition of broadcast sports video". In: *Multimedia Syst.* 14.4 (2008), pp. 179–193. DOI: `10.1007/s00530-008-0112-6`. URL: `https://doi.org/10.1007/s00530-008-0112-6`.

[59] Zengkai Wang et al. "Event boundary determination based on attack-defense transition analysis in soccer video". In: *2014 19th International Conference on Digital Signal Processing* (2014), pp. 321–326.

[60] Z. Xiong, R. Radhakrishnan, and A. Divakaran. "Generation of Sports Highlights Using Motion Activity in Combination with a Common Audio Feature Extraction Framework". In: *IEEE International Conference on Image Processing (ICIP)*. Vol. 1. Sept. 2003, pp. 5–8. URL: `http://www.merl.com/publications/TR2003-118`.

[61] Changsheng Xu et al. “A Novel Framework for Semantic Annotation and Personalized Retrieval of Sports Video”. In: *IEEE Trans. Multimedia* 10.3 (2008), pp. 421–436. DOI: 10.1109/TMM.2008.917346. URL: https://doi.org/10.1109/TMM.2008.917346.

[62] Changsheng Xu et al. “Live Sports Event Detection Based on Broadcast Video and Web-casting Text”. In: *Proceedings of the 14th ACM International Conference on Multimedia*. MM ’06. Santa Barbara, CA, USA: ACM, 2006, pp. 221–230. ISBN: 1-59593-447-2. DOI: 10.1145/1180639.1180699. URL: http://doi.acm.org/10.1145/1180639.1180699.

[63] Changsheng Xu et al. “Sports video analysis: semantics extraction, editorial content creation and adaptation”. In: *Journal of Multimedia*. Vol. 4. 2. 2009, pp. 69–79.

[64] Changsheng Xu et al. “Using Webcast Text for Semantic Event Detection in Broadcast Sports Video”. In: *IEEE Trans. Multimedia* 10.7 (2008), pp. 1342–1355. DOI: 10.1109/TMM.2008.2004912. URL: https://doi.org/10.1109/TMM.2008.2004912.

[65] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. “Comparison of different implementations of MFCC”. In: *Journal of Computer Science and Technology* 16.6 (Nov. 2001), pp. 582–589. ISSN: 1860-4749. DOI: 10.1007/BF02943243. URL: https://doi.org/10.1007/BF02943243.

[66] Yijia Zheng et al. “Highlight Ranking for Racquet Sports Video in User Attention Subspaces Based on Relevance Feedback”. In: *International Conference on Multimedia Expo*. Multimedia and Expo, 2007 IEEE International Conference. IEEE Computer Society, 2007, pp. 104–107.

[67] Di Zhong and Shih-fu Chang. “Structure Analysis of Sports Video Using Domain Models”. In: *IEEE ICME* (2001), pp. 22–25.

List of Figures

Listings

Programmverzeichnis

MacOs: MacOS Sierra 10.12.6.

Python: Python 3.6.3.

Eclipse: Oxygen.1a Release (4.7.1a)

Audacity: 2.2.1.

OpenCV: OpenCV 3.4.1.15 Python

FFmpeg: 3.4

LibRosa: 0.6.0

Tkinter: 3.4

MoviePy: 0.2.3.2

numpy: 1.14.2

A Appendix

A.1 Results

Video results are available at:
https://drive.google.com/open?id=1ohtn1hC1h-2yxSgA2la9KkLiZWPDukgZ

A.2 Deployment and Installation Guide

The code of this work is available via a private repository on bitbucket.com.
Access can be granted after email correspondence at:
christoph.draschkowitz@gmail.com
None of the projects can however successfully finish without a proper metadata file generated following the same XML Schema that "Sportradar" uses for it.
There are currently three projects available. The following applies to each of these projects:
As they are python projects "Python 3" has to be installed. A requirements.txt file specifies the needed libraries. Those can be installed by the use of "pip" and its command pip install <library> [10]. Pip is already pre installed in proper Python installs. If the "contrib" version of OpenCV 3.4.1.15 is installed with pip, it should be possible to run the program without installation of OpenCV on the computer. However, if an OpenCV error occurs, it might be necessary to install the package manually on your computer. If "FFMPEG" is not installed on the device it gets installed automatically through code in class Controller.
The three projects are described in the following:

A.2.1 HighlightComposition_Python is the initial project that was created for this thesis. It is a Python desktop application that lets a user specify the input files over a graphical user interface. A monitoring mode can be enabled. This mode is not available in the other two projects. The constant video offset is already saved hardcoded inside the program code for the available broadcasts. This was more practical during the deployment phase. Besides the constant time offset caused by inaccurate device clocks, the events specified in the metadata

[10] http://www.pypi.org/project/pip

files for this project are extremely accurate. (One second late for goals and up to five seconds for other events). Due to the high amount of events to analyze and the high amount of events in the final video (many goals compared to for example soccer), the composition can take up to 20 minutes on a desktop computer. In order to use monitoring mode it is required to install "imagemagick". For me it was furthermore necessary to specify the path to the binaries in the "VideoCutter" Class (column 25+26). It is currently out commented but might need to be specified in order to run the program in monitoring mode. MainClass.py has to be called to start the framework.

A.2.2 HighlightComposition_soccer is the same project as above, adapted to the needs of soccer. Monitoring mode is not available. The time to calculate and cut the video is much shorter due to less events in soccer games. In an ice hockey game 40 shots can be expected where as a soccer game very often does have less than 10. The constant offset needs to be set by the user inside the User Interface. The offset here is the time (in seconds) that the input video already runs when the game starts. Besides the constant time offset caused by inaccurate clocks, the events specified in the metadata files for this project are rather inaccurate. (5 to 15 seconds). MainClass.py has to be called to start the framework.

A.2.3 flasktest is another version of the HighlightComposition_soccer project. Instead of the GUI it is realized as a Python Flask app, that can be deployed on web servers. The User Interface is similar apart from being realized inside the web browser. Also the final composition video is provided as a download via a hyperlink, shown in the browser. application.py has to be called to start the framework.

A.3 Component Description

This section describes the modules of the framework. Text is copied from the code repositories' readme files:
package event
Event ()

used to create Event objects
def $_{init}$(self, id, info, matchscore, mtime, periodnumber, remainingtimeperiod, side, stime, type, extrainfo, posx,posy):
as constructor
get and set methods for most of the variables
contains a variable TimeBorders and Replay
which are both objects with additional information about the event
TimeBorders(object):
belongs to exactly one event object; consists furthermore of two numbers that tell the videocutter where to cut;
does alredy save the time that is run on the video; NOT the timestamp from metadata file!
created when events get created; adapated while composition
——
package io
def getDataDir(self):
returns the absolute path of the data folder inside the project; is used several times within the framework
def readMetadataFile(self,mpath):
reads in the metadata file from hard disc ;
video is imported directly with opencv;
—
package metadata
def getDataDir(self):
XML Extraction checks xml file for validity
extracts the information from meta data file and creates Event objects
Time Synchronisation
gets video encoded timestamp from video with library of mediainfo and calculates the offset to metadata file
——
package rating
ScoreCalculator
is used to handle priorities of the events, chosen by type

—–

package test
modules that run certain tests
maintests test all components for basic functionality
audiofeatures is used to create, test and adapt audio features

—

package ui
User Interface
called by MainClass
collects input
def open GUI () gets called;
opens up new window for user to specify files
then calls controller to start with processing
to do that it shows the user a dynamic "please wait" window and opens up a new thread that handles the workload
once the thread is done; a flag gets turned and the program returns to Main Class to finish

—–

package cutting
Sequence Extractor
calculates the right time boarders for the events of the final eventlist:
uses type specific hard coded numbers for the inconstant event type offset;
then calls EventBoundaryDetaction for the information on shot cuts and replays, and MotionAnalysis for the existance of turnovers,
VIdeoCutter
is called at the very end of the processes controlled by the controller;
uses the library moviepy
to cut together all event clips that are part of the parameter eventlist
adds all specified fades between the clips.
adds Replays after clips
and includes audio fades
if specified it writes a textOverlay
and finally saves the video;

——

package audio extraction
uses library librosa for basic audio handling
usage:
seperately call audio extraction for home and away team events;
since fans are usually unevenly spread, those are not comparable;
extracts audio
and calculates all audio features;
at the end weights all features and
returns best events regarding its audio;

—

package video
FrameExtraciton
uses opencv to extract frames from a video file
EventBoundaryDetection
gets all frames a given events and looks for all shot cuts and replays and returns them in dictionaries
MotionAnalysis
uses opencv to calculate motion direction
calculates optical flow of image sequence and returns length and euclidean distance of points

A.4 Abstract

Motivation of this work is the fact that the creation of highlight videos nowadays takes a lot of manpower and is time consuming. There is a rising interest in global sports and more and more sports get spread around the world. Big sport teams nowadays sell their rights to more than a 100 countries. Due to video cameras and technical equipment getting cheaper and better, even rather small sport events can be recorded and very often these are also getting broadcasted, for example on online channels. The demand for those broadcasts might be small, whereas a short summary of events might very often be something more people would want to watch. This master thesis uses a regular TV broadcast and a text file, exported from a live ticker about the game, as inputs and aims to produce a sport highlight video without the full assistance of experts in the progress. However, an expert and/or user could adapt the model on the fly, by giving inputs. Thus the overall thematic area of this work is on one hand the retrieval of multimedia features. On the other hand it is also about creating and adapting a model (for events) using various methods that include any type of data analysis. Results confirm that, by the help of timestamped play by play data and multimedia techniques, a very decent highlight video can be created. While the question remains if a fully automatically generated highlight video can consistently perform better than a manually created video, it could be proven that such a framework can be extremely helpful in the creation process, for example by choosing events beforehand, by detecting event boundaries and many more.

A.5 Zusammenfassung

Die Motivation dieser Arbeit basiert auf dem Fakt, dass das Kreieren von Highlight Videos auch heutzutage noch zeitintensiv ist. Es existiert eine steigende Nachfrage nach Sportkonsumation und viele Sportarten verteilen sich mittlerweile über den gesamten Globus. Große Sport Teams verkaufen ihre Rechte an mehr als 100 Länder weltweit. Da Videokameras und technisches Equipment immer besser und billiger verfügbar sind, werden sogar Randsportarten wahrgenommen und sehr oft werden Spiele davon zum Beispiel in Online Kanälen live gezeigt. Während die Nachfrage danach oft eher gering ist, wäre eine Kurzzusammenfassung der Videos etwas, das viele Menschen ansprechen könnte. Diese Masterarbeit benutzt einen regulären TV Broadcast und ein Text File, das von einem Live Ticker über ein Eishockeymatch exportiert wird, als Input und produziert daraus ein Sport Highlight Video, ohne dazu auf die Hilfe von Experten zurückzugreifen. Ein User könnte jedoch über ein Interface Einfluss nehmen. Daher ist diese Arbeit thematisch im Bereich "Multimedia Retrieval" angesiedelt. Es geht jedoch auch darum Modelle zu erstellen und zu adaptieren, die sowohl Micro Events, als auch Regeln zur Komposition des Videos beschreiben. Die Resultate zeigen, dass mithilfe von "play by play" Daten mit Zeit Informationen, dem dazugehörigen Video und Multimedia Techniken,

qualitativ gute Highlight Videos erstellt werden konnten. Die Frage ob vollautomatisch kreierte Videos dauerhaft besser als manuell erstellte Videos sein können, bleibt natürlich bestehen. Allerdings konnte bewiesen werden, dass ein solches Framework extrem hilfreich in einem solchen Prozess sein kann. Zum Beispiel um im vornherein interessante Events zu erkennen und Abgrenzungen zu schaffen, aber auch durch viele andere Möglichkeiten.