Rechercheberichtt

Synchronisation und Auswertung von Roboterfußballvideos

tj18b

Mitglieder
Dan Häßler
Robert Wagner
Sirk Petzold
Alex Eichhorn
Erik Diener
Jonas Wrubel
Tomas Daetz Chacon

Betreuer
Hans-Gert Gräbe
Tobias Wieprich
Tobias Jagla
André Köhler

Inhaltsverzeichnis

1 Glossar

Java Java ist eine objektoriente Programmiersprache, die sich durch ihre Plattformunabhängigkeit auszeichnet, der Quellcode wird in bytecode kompiliert, der dann auf jeder JVM unabhängig von System und Architektur laufen kann. [12]

XML (engl.: Extensible Markup Language) ist eine Auszeichnungssprache, die hierarchisch strukturierte Daten darstellen kann. Es wird auch eingesetzt für den plattform- und implementationsunabhängigen Datenaustausch. [8]

Parser Ein Parser ist ein Programm, das eine Eingabe derart zerlegt und umwandelt, dass es in einem, für die Weiterverarbeitung, (optimalen) Format ist. [15]

API Eine Schnittstelle zur Anwendungsprogrammierung, häufig kurz API genannt (engl.: application prgramming interface, wörtl.: Anwendungsprogrammierschnittstelle), über das andere Programme sich an das Softwaresystem anbinden kann. [17]

Library Als Library (dt.: Programmierbibliothek; kurz: Bibliothek) bezeichnet man eine Sammlung von Unterprogrammen/-Routinen inklusive, falls vorhanden, Dokumentation, templates, pre-written code, [...]. Eine Library bietet Lösungswege/-ansätze für thematisch zusammenhängende Problemstellungen. Der Zugriff auf eine Bibliothek erfolgt über eine API, die sich aus der Gesamtheit der öffentlichen Funktionen und Klassen zusammensetzt. [14] [16]

GUI (engl.: graphical user interface) ist eine grafisches Benutzeroberfläche, die als Benutzerschnittstelle dient, indem einerseits der Mensch mit dem Programm kommunizieren kann, in Form von Buttons, Slidern, o.Ä., andererseits aber auch das Programm mit dem Menschen kommunizieren kann, indem Daten auf der GUI visualisiert werden. [11]

Frame Einzelbild in Filmen, Animationen und Computerspielen

FPS Frames pro Sekunde

Synchronisation bedeutet in diesem Zusammenhang mehrere Elemente zeitlich auf dem Wiedergabegerät an eine Datenquelle anzupassen. [18]

Kanal Farbkanäle (R,G,B) aus denen ein Frame zusammengesetzt ist.

Software ist ein Computerprogramm, einschließlich ihrer dazugehörigen Dokumentation und Konfigurationen, die erforderlich sind um das Programm auszuführen.

Plattform Eine Plattform ist die Umgebung, in welcher ein Stück Software ausgeführt wird, dabei kann es die Hardware, aber auch das Betriebssystem beschreiben, im Folgenden in Bezug auf Rechnerarchitektur bzw. Betriebssystem benutzt. [6]

Fourier Transformation Verfahren zur Umwandlung komplexer Signale in die einzelnen Frequenzen ihrer Sinus-Anteile.

Framework Rahmenapplikation die grundsätzliche Strukturen und Funktionen zur Entwicklung von Programmen bereitstellt. Stellt nicht nur Klassen und Funktionen bereit wie eine Library, sondern in der Regel auch die grobe Anwendungsarchitektur. [10]

Codec (Portmanteauwort aus **co**der und **dec**oder) Ein Codec ist eine Einheit aus Kodierer und dem entgegengesetzten Dekodierer mit entsprechender Umkehrfunktion. [5]

Encoder Ein System, das eine Datenquelle in ein für einen bestimmten (Übertragungs)kanal geeignetes Format konvertieren soll. [13]

Decoder ist der Gegensatz zum Encoder, die Ausgangsdaten/-signale des Kodierers werden wieder ins ursprüngliche Format zurückkonvertiert, dabei müssen nicht die ursprünglichen Informationen erhalten bleiben, die Konvertierung kann verlustfrei und verlustbehaftet sein. [7]

2 Konzepte

2.1 Fourier Transformation

Eine der notwendigen Techniken derer man sich gegebenfalls bedienen muss, um die Bild-Audio Synchronisation durchzuführen ist die Fourier Transformation. Audiosignale bestehen in der Regel aus mehreren überlagernden Sinuswellen unterschiedlicher Frequenzen und Amplituden. Möchte man ein solches Signal nach einem bestimmten Frequenzanteil, wie den Ton einer Pfeife absuchen, ist es notwendig dieses in das Frequenzenspektrum zu zerlegen aus denen es besteht. Die Fourier Transformation kann dies leisten und fungiert so als Prisma der Wellenmechanik. Sei f(t) die Funktion die das zu untersuchende Signal beschreibt, so gilt für das kontinuierliche Spektrum:

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} \mathrm{e}^{-i\omega t}\, \mathrm{d}x$$

Dabei ist ω die Kreisfrequenz der Schwingung mit $\omega = 2{\cdot}\pi{\cdot}\mathrm{F}$, t die Zeit. Somit kann durch leichte Umrechnung F auch als Funktion der Frequenz aufgefasst werden. Eine Untersuchung dieser speziellen Stammfunktion ermöglicht das Auffinden des gesuchten Signals und kann somit verwendet werden um das Auftreten des Anpfiffs zu lokalisieren und entsprechend mit dem Video zu synchronisieren. [9]

2.2 Libraries

Zur Auswahl der geeigneten Library für die Videobearbeitung kamen vorab 3 Varianten in Frage: VlcJ, OpenCV und Xuggler.

Um die Entscheidung auf fundierten Grundlagen treffen zu können, lag der Beschluss nahe, dass sich je 1 Teammitglied zunächst so tiefgehend wie notwendig mit einer dieser Librarys auseinandersetzt. Dabei wird der Fokus vordergründig auf 3 Aspekte gelegt.

Zum einen ist die Bedien- und Modularisierbarkeit des strukturellen Aufbaus, der durch die Verwendung einer externen Quelle partiell vorgegeben wird, sehr wichtig um ins-

besondere den GUI-Aufbau klar von der Video-Logik trennen zu können und die Aufgabenverteilung schließlich möglichst unabhängig gestalten zu können. Darüber hinaus sollte das verwendete Package selbstverständlich alle notwendigen Funktionalitäten ermöglichen und dem Entwickler dabei idealerweise umfassende Kontrolle über sämtliche Bearbeitungschritte gewähren. Zentrale Funktionen sind das Abspielen von Videos, Sprünge zu spezifischen Punkten, sowie Einzelsprünge um je ein Frame. Das Bearbeiten (Schneiden) und speichern von Videodaten ist ebenso unverzichtbar wie das konkatenieren von Teilvideos. Für zahlreiche Zusatzfeatures wäre es zudem wünschenswert, wenn die Möglichkeit bestünde, einzelne Frames als Bilder herauszuziehen und zu analyiseren(automatischer Kamerawechsel), sowie eine seperate Bearbeitung der Tonspur(Audio-Synchronisation). Es muss möglich sein, mehrere Videofiles parallel abzuspielen und jeden Einzelnen separat kontrollieren zu können. Das Design des Players und der entsprechenden Kontrolleinheiten wie Buttons oder Slider sollte dabei möglichst vollständig in der Hand des Entwicklers liegen.

Idealerweise ermöglicht der Aufbau des Packages das Anlegen eines kleinen Frameworks, mit dessen Hilfe die benötigen Funktionen einfach und ohne gröbere Schierigkeiten in ein vorgegebenes GUI-Konzept implementiert werden können.

Im Folgenden werden die Rechercheergebnisse zu den einzelnen Librarys vorgestellt.

2.2.1 OpenCV

Bezüglich der Einrichtung des packages wurden zunächst negative Eindrücke gesammelt. Insbesondere für Java war es mit einigen Schwierigeiten verbunden eine saubere Funktionsweise zu erzeugen. Das Fehlen einiger DLL-Dateien sowie die Optimierung für Eclipse machten die Einrichtung unter Netbeans zu einer zeitraubenden Angelegenheit.

Die Dokumentation ist zwar weitestgehend vollständig, aber nicht zwingend ausführlich, sodass von Zeit zu Zeit einige Fragen bezüglich verschiedener Funktionsargumente oder den notwendigen Imports zur Verwendung bestimmter Klassen offen bleiben. Es finden sich aber zahlreiche Beispielanwendungen im Netz, durch die sich die meisten Dinge ohne zu großen Aufwand recherchieren lassen. Dennoch bleibt hier ein kleiner Makel.

Die erste Programmierung mit OpenCV hingegen erwies sich als sehr angenehm. Das Abspielen einer lokalen Videodatei wird über das laden in ein VideoCapture Objekt, und das Anlegen eines Threads mit entspechendem Timer verwirklicht. Dabei wird jedes abgespielte Frame tatsächlich in die Hand genommen, was sehr viel Kontrolle und Spielraum für Funktionen ermöglicht. Jedes Frame kann einzeln bearbeitet, in Kanäle gesplittet, als Java Image erzeugt oder per Stream in Schreibbare Video – Datentypen geschoben werden, wodurch zahlreiche der gewünschten Funktionen bereits abgedeckt sind.

Da sämtliche Videobearbeitungsoptionen in Streams ablaufen, wird es notwendig für sämtliche Cut- oder Merge-Prozesse jedes Frame einzeln abzugreifen. Die Befürchtung hier auf Performanceprobleme aufzulaufen liegt nahe. Insofern ist hier ein wenig Aufbereitung notwendig.

Dabei besteht die Überlegung, beim Laden jedes Videos alle Frames in ArrayListen vorzuladen. Sämtliche Operationen für Abspielen und Spulen könnten dann auf diesen Listen durchgeführt werden. Der große Vorteil ist hierbei, dass für das Schneiden und Konkatenieren von Teilvideos einfach Listen gekürzt oder zusammengefügt werden können. Neben der zu erwartenden Verbesserung der Performance ist hierbei auch die Transparenz im Bearbeitungsprozess für den Entwickler deutlich höher, da man nicht mehr auf die vorgegebenen Funktionalitäten der Library angewiesen wäre, sondern alle Prozesse selbst implementieren könnte. Erst beim Speichern der Videos könnte man wieder OpenCV – Capture und Recorder Instanzen erzeugen, um schließlich in Dateien zu schreiben. Das Package würde hierbei nur noch für die Rahmenprozesse benötigt werden und alle nötigen Freiheiten ermöglichen.

Das zentrale Problem jedoch ist die fehlende Audio-Unterstützung. Möchte man neben den visuellen Frames auch Audio entsprechend anpassen und editieren, ist OpenCV leider die falsche Wahl. Hierfür müsste man also zusätzliche packages wie FFmpeg heranziehen, wobei sich die Frage stellt, inwieweit sich diese sinnvoll kombinieren lassen.

Grundsätzlich kann man OpenCV eher eine partielle Eignung für die vorgegebene Zielsetzung attestieren. Auch wenn nur ein geringer Teil des Funktionsumfangs benötigt wird, lässt allein der Grundaufbau der Library zu, jedwede nötigen Prozesse einfach selbst zu implementieren und dadurch keine Einbußen in der Strukturierung seines Programmes zu verzeichnen. Darüber hinaus stellt OpenCV entsprechende Timer-Elemente zur Verfügung, die die Organisation der Abspielprozessse in Threads übernimmt und somit sehr viel Progammierbürokratie abwickelt.

Negativ bewerten muss man die geringe Ausgereiftheit gegenüber der Varianten für C++ und Python. Wo es dort z.b. die Möglichkeit gibt, bestimmte Frames direkt über Indizierung abzugreifen, bleibt für Java bislang nur die Möglichkeiten dorthin zu iterieren, was gegebenfalls bei größeren Datein zu Performanceprobleme führen könnte. Auch die Gefahr für Abstürze und Exceptions ist durch das nicht 100Die fehlende Unterstützung für die Tonausgabe ist allerdings ein großes Problem für das vorliegende Projekt, und es müsste geklärt werden inwieweit sich dies auffangen ließe. Sich von einer zweiten Library abhängig zu machen wäre aufgrund des Wesens von OpenCV zwar zu verschmerzen, es müsste jedoch geklärt werden ob beispielsweise FFmpeg nicht bereits allein in der Lage wäre, alles notwendige zu leisten. Durch den geringen Eingriff in die Programmphilosophie ist dennoch nahezulegen, OpenCV zumindest einzubeziehen. Sollte sich eines der anderen Packages aber als umfassender und ähnlich unkompliziert erweisen, ist es natürlich ratsam eher auf dieses zurückzugreifen.

2.2.2 Xuggler

Xuggler

> "Xuggler is the easy way to uncompress, modify, and re-compress any media file (or stream) from Java."[22]

Xuggler ist eine Library, mit der man leicht Videodateien manipulieren kann. Die Installation der Library war mühsam, da viele Dependencies bestehen und nach weiterer Recherche stellte sich heraus, dass Xuggler bekannte Probleme mit einigen Prozessorarchitekturen hat, ich verwendete Java 32bit um das Problem zu lösen, grundsätzlich ist es aber möglich es auch auf Java 64bit zu verwenden. Die Dokumentation ist fast vollständig, zusätzlich existieren mehrere Code Beispiele, Stack Overflow Beiträge, sowie Einführungen in die Library. Xuggler baut auf FFmpeg auf, sodass der Code überall laufen sollte, wo auch FFmpeg läuft.

Xuggler bietet zwei verschiedene APIs, welche für gleiche Zwecke benutzt werden können.

MediaTool API ist eine einfach zu benutzende API zum Modifizieren von Videos/Medien in Java. Dabei abstrahiert die API die kleinen Details von Containern, Codecs und anderem, sodass der Fokus auf den Medien liegt. Veranschaulicht wurde das im Beispielprogramm, trotzdem ist es möglich auf die unterliegenden Xuggler Objekte zuzugreifen und sie zu manipulieren.

Xuggler Advanced API ermöglicht es, konträr zu der MediaTool API, auf die Details der Videomanipulation einzugehen, allerdings wird damit auch die Komplexität erhöht.

Das erste Programm stellte sich als schnell geschrieben heraus, im folgenden ein beispielhaftes Programm zur Wiedergabe eines Videos mit Audio mit Benutzung der MediaTool API

Listing 2.1: Videoausgabe mit Audio

```
1 public static void main(String[] args){
2     IMediaReader reader = ToolFactory.makeReader("Szene01.MP4");
3     IMediaViewer viewer = ToolFactory.makeViewer();
4     reader.addListener(viewer);
5
6     while (reader.readPacket() == null);
7 }
```

Im Beispielprogramm sehen wir die Initialisierung eines IMediaReader und IMediaViewer. Anschließend, haben wir dem IMediaReader den IMediaViewer hinzugefügt. Solange der IMediaReader nun Pakete liest sehen wir die Ausgabe.

IMediaReader

> "An IMediaReader opens up a media container, reads packets from it, decodes the data, and then dispatches information about the data to any registered IMediaListener objects"[19]

IMediaViewer

> "You can use this object to attach to a IMediaReader or a IMediaWriter to see the output as they work."[20]

Im Folgenden noch die grundlegende Struktur einer Medienpipeline, die eine Kette von Medienmanipulation an einem Medium ausführt. (hier: ein Zeitstempel hinzufügen und die Lautstärke auf 10% setzen)

Listing 2.2: Media Pipeline[21]

```
1  public static void main(String[] args){
2      IMediaReader reader = ToolFactory.makeReader("Szene01.mp4");
3      reader.setBufferedImageTypeToGenerate(BufferedImage.TYPE_3BYTE_BGR);
4
5      // create a writer and configure it's parameters from the reader
6      IMediaWriter writer = ToolFactory.makeWriter("Szene01_berarbeitet.mp4", reader);
7
8      // create a tool which paints a time stamp onto the video
9      IMediaTool addTimeStamp = new TimeStampTool();
10
11     // create a tool which reduces audio volume to 1/10th original
12     IMediaTool reduceVolume = new VolumeAdjustTool(0.1);
13
14     // create a tool chain: reader -> addTimeStamp -> reduceVolume -> writer
15     reader.addListener(addTimeStamp);
16     addTimeStamp.addListener(reduceVolume);
17     reduceVolume.addListener(writer);
18
19     // add a viewer to the writer, to see the modified media
20     writer.addListener(ToolFactory.makeViewer(AUDIO_VIDEO));
21
22     // read and decode packets from the source file and
23     // then encode and write out data to the output file
24     while (reader.readPacket() == null);
25 }
```

Hierbei wird es nochmal deutlich die MediaTool API ist einfach zu benutzen und lässt trotzdem Manipulierungen der Medien zu. Resümierend ist also zu sagen, dass die Library optimal zu unserem Anwendungsfall passt, es ermöglicht einfache Komponenten schnell zu entwickeln, aber, falls benötigt, auch komplexere Komponenten zu realisieren. Insbesondere die Anbindung an FFmpeg ist vorteilhaft, da Xuggler so auch fast jedes Audio- und Videoformat unterstützt. Dies und die Dokumentation sowie Code Examples bieten eine umfassende Library, die gut zu unseren Anforderungen passt.

2.2.3 VLCJ

VlcJ ist eine Library für Java, welche Open-Source, von Caprica Software, entwickelt wird. VlcJ soll alles implementieren, was LibVlc enthält.

Die Library ist sehr gut dokumentiert, was zu schnellem Verständnis geführt hat und Vorteile für die Verwendung mit sich bringt. Auf der Website des Entwicklers finden sich zahlreiche Tutorials sowie ausreichend Anwendungsbeispiele in Foren. Das Entwickeln mit VlcJ ist recht unkompliziert. In wenigen Minuten hat man einen funktionsfähigen MediaPlayer implementiert. Hierzu müssen zuerst die libVlc[3]-Daten auf dem Gerät gefunden werden. Dies wird mit einer Methode registerLibrary() erledigt, welche im Konstruktor des Players aufgerufen wird.

Listing 2.3: Registrieren der Library

```
1    private void registerLibrary() {
2        NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(), "d:/vlc-2.2.1");
3        Native.loadLibrary(RuntimeUtil.getLibVlcLibraryName(), LibVlc.class);
4        LibXUtil.initialise();
5    }
```

Um den Player darzustellen wählt wählt man eine Canvas-Komponente als VideoSurface[**?**]. Auf diesem Canvas wird der Player konfiguriert.

Listing 2.4: Konfigurieren des Players (buildFrame()[2])

```
1    /** * Create the frame where movie will be played */
2    private Frame buildFrame(final Canvas videoSurface) {
3        final Frame f = new Frame("Test Player");
4        f.setSize(800, 600);
5        f.addWindowListener(new WindowAdapter() {
6            @Override
7            public void windowClosing(WindowEvent e) {
8                System.exit(0);
9            }
10       });
11       f.setLayout(new BorderLayout());
12       f.add(videoSurface, BorderLayout.CENTER);
13       f.setVisible(true);
14       return f;
15   }
```

Das zurückgegebene Frame-Objekt ist der Bereich, in dem der Player später dargestellt wird. Da es auch Factories für diese Objekte geben kann, wirkt es als könnte man so mehrere Player parallel laufen lassen.

Dann erstellt man sich noch ein EmbeddedMediaPlayer[4]-Objekt. Diesem wird die VideoSurface zugewiesen, außerdem ist das Abpielen mit diesem Objekt möglich.

Listing 2.5: CreatePlayer() (createPlayer()[2])

```
1  /** * Build the player */
2  private EmbeddedMediaPlayer createPlayer(final List<String> vlcArgs,
3  final Canvas videoSurface) {
4      EmbeddedMediaPlayerComponent component = new EmbeddedMediaPlayerComponent();
5      EmbeddedMediaPlayer player = component.getMediaPlayer();
6      MediaPlayerFactory factory = new MediaPlayerFactory(
7          vlcArgs.toArray(new String[vlcArgs.size()]));
8      factory.setUserAgent("vlcj test player");
9      player.setVideoSurface(factory.newVideoSurface(videoSurface));
10     player.setPlaySubItems(true);
11     return player;
12 }
```

Mit dem Aufrufen der Play-Methode übergibt man einen Pfad an den Player und das Video wird abgespielt.

Genau ab diesem Punkt zeigen sich allerdings erste Probleme. Denn VlcJ bietet keine Möglichkeit auf Listen von Frames oÄ zu arbeiten. Die Möglichkeit auf Tonspuren einzeln zuzugreifen ist zwar gegeben, aber auch hier wurde keine Funktion zum schneiden oder editieren gefunden. Die Library scheint eher Ihren Zweck darin zu haben, Media-Player-Features zur Verfügung zu stellen. Außerdem wird vom Entwickler davon abgeraten, mit mehreren eingebetteten Playern gleichzeitig zu arbeiten. Dies ist auf jeden Fall von Nöten.

2.2.4 FFmpeg

FFmpeg ist ein Kommandozeilen-Programm, welches vielseitige Funktionalitäten der Video- und Audio-Manipulation vereint. Außerordentlich mächtig ist das Tool im Konvertieren, da es mehr als nur alle gängigen Video- und Audio-Codecs unterstützt.

Die Tools sind namentlich in 4 Kategorien eingeteilt: ffmpeg, ffplay, ffprobe, ffserver.
ffmpeg: Hauptsächlich für Konvertierungen zuständig.
ffplay: Simpler Media Player - nutzt die ffmpeg und die externe SDL (Simple DirectMedia Layer) Libraries.
ffprobe: Anzeigen von Informationen aus Multimedia-Streams.
ffserver: Streaming-Server für Multimedia-Übertragungen (auch live).

Der andere Bestandteil sind Libraries. Besonders erwähnenswert sind dabei libavcodec und libavformat, aber es gibt noch ein paar weitere.
libavcodec: Stellt ein Framework mit zahlreiche Encoder und Decoder bereit.
libavformat: Stellt ein Framework für Multiplexing-Funktionen bereit. Dadurch lassen sich Video-, Audio- und Untertitel-Spuren "muxenünd "demuxen". Beim Muxing werden mehrere Spuren zu einem Stream zusammengeführt. Umgekehrt dazu, wird beim Demuxing ein Stream in mehrere parallele Spuren aufgesplittet.

Zu relevanten Anwendungsbereichen/beispielen zählt u.a.:

- springen zu bestimmtem Frame im Video
- schneiden/speichern von Audio/Video
- zusammenfügen mehrerer Videos

Da FFmpeg in C verfasst ist, existiert keine native Java-Kompatibilität, sodass auf einen Wrapper zurückgegriffen werden muss.

Dieser Umstand erweist sich als hinderlich für die alleinige Verwendung von FFmpeg unter Java, da es zwar in großen, gut dokumentierten und erhaltenen Bibliotheken (z.B. Xuggle) in Java genutzt wird, allerdings nur sehr spezifische (nur ausgewählte Funktionen), oberflächliche (voller Funktionsumfang, jedoch sehr unübersichtlich/schlecht dokumentiert) oder veraltete (FFmpeg wird gelegentlich aktualisiert) Wrapper für pure FFmpeg-Funktionalität existieren.

Desweiteren verlassen sich viele dieser Wrapper direkt auf die eigentliche (platformabhängige) FFmpeg-Programmgruppe (ffmpeg, ffplay, ffprobe usw.), was einer plattformunabhängigen Programmierphilosophie stark entgegen wirkt.

Da FFmpeg, wie schon erwähnt, in größeren, prominenten Java-Bibliotheken (Xuggle, OpenCV/JavaCV) als Teilgrundlage ihrer Funktionalität Verwendung findet, bietet es sich an, eine dieser Bibliotheken für die Entwicklung zu wählen, um einen Kompromiss zwischen vollständiger FFmpeg-Funktionalität und leichtem Verständis, vollständiger Dokumentation und Intuitivität beim verwenden dieser unter Java zu gewährleisten.

2.2.5 JavaCV

JavaCV ist eine Open-Source-API von Bytedeco und bezieht gleich mehrere Libraries ein - unter anderem OpenCV, FFmpeg und OpenKinect.

Die Installation bzw. Einbindung in IntelliJ IDEA war erfreulich simpel. Auch eine Automatisierung mittels Maven oder Gradle ist möglich - es besteht nur eine Dependency. Die Entwickler geben zu, dass die Dokumentation lückenhaft ist. Zunächst erst einmal ist auf github gar keine Dokumentation auffindbar, stattdessen verweist man auf ein paar Beispielcodes und Demo-Projekte. Glücklicherweise aber existiert eben diese auf der Homepage der Entwickler [1]. Leider sind aber die meisten Klassen und Methoden nicht erläutert, was für Einsteiger das Zurechtfinden und Erlernen erschwert. Einfache Anforderungen, wie z.B. das Auslesen von Metadaten einer Datei oder die Ausgabe der Zeitstempel einzelner Frames, ist in einer überschaubaren Menge an Code realisierbar.

Für unser Projekt wäre es aber auch notwendig, mehrere Video- und Audio-Streams

gleichzeitig ablaufen zu lassen und zu synchronisieren. FFmpeg kann dies und zusätzlich besteht damit die Möglichkeit Videos zu schneiden. Mir zumindest ist es aber schwer gefallen, die gesuchten Funktionalitäten in den Tiefen von JavaCV zu finden und praktisch anzuwenden. Aber theoretisch betrachtet kann JavaCV alle unsere Anforderungen verwirklichen. Kompatibilitätsprobleme aufgrund der unterschiedlichen Video-Formate dürften auch ausgeschlossen sein, dank der vielen Codecs, die FFmpeg mit sich bringt. Zur Performance lassen sich zum jetzigen Zeitpunkt nur Vermutungen anstellen. Da JavaCV eine langjährige Entwicklung durchgangen hat und stets weiterentwickelt wird und aufgrund der Tatsache, dass JavaCV sich für Multimedia-Projekte in Java etabliert hat, ist die Performance positiv einzuschätzen.

3 Aspekte

3.1 Zielsetzung

Das Ziel unseres Projektes ist die Entwicklung eines Tools zur übersichtlichen Verarbeitung von Roboterfußballvideos und den dazugehörigen Audio- und Log-Files. Dabei soll es möglich sein bis zu 4 Videos sychronisiert nebeneinander darzustellen. Des weiteren soll das Tool die Möglichkeit bieten, eben diese Video- und Audiodateien genaustens zu bearbeiten, insbesondere soll ein Frame- genauer Zugriff auf die Videos möglich sein.

3.2 Rahmenbedingungen

Den Rahmen für unser Projekt bilden verschiedene Vorgaben der Projektleiter, sowie einige Annahmen, welche getroffen werden, um sonst kritische Fälle zumindest für die Soll- Erfüllung ausschließen zu können. Es liegen GC-Logfiles und Videodateien vor. Bezüglich der Logfiles ist es wichtig, dass die zwei vorliegenden Files geladen werden können müssen. Die Videodateien sind alle im gleichen Format und haben die gleiche FPS-Zahl. Des weiteren ist der Offset der Videos zum Start des Spieles bekannt.

3.3 GUI

Da sich unser Projekt um die Bearbeitung von mehreren synchronisierten Videos dreht, ist ein intuitives und übersichtliches GUI entscheidend. Die Funktionalität und die Features bilden zwar einen wichtigen Bestandteil, dennoch ist es wichtig das GUI möglichst einfach und vor allem klar strukturiert aufzubauen, um die Übersichtlichkeit zu gewährleisten. Durch den Umstand, dass das entstehende Tool einer spezifischen Gruppe zur Verfügung gestellt wird, ist es uns möglich durch regelmäßige Rücksprache die Wünsche der Zielgruppe soweit umsetzbar zu implementieren.

3.4 Problemfälle

Im späteren Projektverlauf wird die Behandlung der in den Rahmenbedingungen bereits erwähnten Problemfälle möglicherweise ein bedeutender Aspekt. Es ist zu betrachten, wie man mit Änderungen des GC-Formates umgehen kann, da für dieses keine Spezifikation vorliegt. Auch gesplitete Log-Dateien, eine bruchstückhafte oder sogar fehlende Audiospur, unterschiedliche Videoformate oder verschiedene Fps-Zahlen der Videos, sowie fehlende Informationen bezüglich der Offsets sind zu betrachten.

3.5 Ansatz

Die Wahl der geeigneten Libary ist ein wesentlicher Bestandteil für den Erfolg des Projektes. Darum ist es wichtig, sich im Rahmen der Recherche ausführlich mit den zur Verfügung stehenden Liberys auseinanderzusetzen. Im Idealfall ist die gewählte Libery intuitiv bedienbar, modularisierbar und erfüllt sämtliche Funktionalitäten, welche im Rahmen des Projektes verlangt werden. Bei der Auswertung der Konzepte ist aufgefallen, dass sowohl VLcJ als auch OpenCV unseren Ansprüchen nicht vollends gerecht werden. VLcJ bietet keine Möglichkeiten der Video- und Audiobearbeitung, des weiteren ist eine multiple Einbettung von Videos nicht möglich, somit besitzt das Package essentielle Funktionalitäten nicht und ist daher für unser Projekt eher ungeeignet. OpenCV bietet umfangreiche Möglichkeiten der Videobearbeitung, jedoch fehlt die Audio-Unterstützung. Es besteht die Möglichkeit, dies durch ein anderes Package zu kompensieren. Xuggler bietet die geforderten Funktionalitäten in vollem Umfang hinsichtlich der Video- und Audiobearbeitung.

Literaturverzeichnis

[1] Bytedeco. Javacv 1.4.3 api. "http://bytedeco.org/javacv/apidocs/, zuletzt besucht am 12.11.2018.

[2] MrBool. Sample media player. http://mrbool.com/api-vlcj-creating-java-audio-video-players/33033, zuletzt besucht am 12.11.2018.

[3] VideoLan. Videolan wiki. https://wiki.videolan.org/LibVLC/, zuletzt besucht am 12.11.2018.

[4] VideoLan. Videolan wiki. http://caprica.github.io/vlcj/javadoc/3.10.1/index.html?uk/co/caprica/vlcj/player/embedded/videosurface/CanvasVideoSurface.html, zuletzt besucht am 12.11.2018.

[5] Wikipedia. Codec. https://de.wikipedia.org/wiki/Codec, zuletzt besucht am 12.11.2018.

[6] Wikipedia. Computing platform. https://en.wikipedia.org/wiki/Computing_platform, zuletzt besucht am 12.11.2018.

[7] Wikipedia. Dekodierer. https://de.wikipedia.org/wiki/Dekodierer, zuletzt besucht am 12.11.2018.

[8] Wikipedia. Extensible markup language. https://de.wikipedia.org/wiki/Extensible_Markup_Language, zuletzt besucht am 12.11.2018.

[9] Wikipedia. Fourier-transformation. https://de.wikipedia.org/wiki/Fourier-Transformation, zuletzt besucht am 12.11.2018.

[10] Wikipedia. Framework. https://de.wikipedia.org/wiki/Framework, zuletzt besucht am 12.11.2018.

[11] Wikipedia. Grafische benutzeroberfläche. https://de.wikipedia.org/wiki/Grafische_Benutzeroberfl%C3%A4che, zuletzt besucht am 12.11.2018.

[12] Wikipedia. Java (programmiersprache). https://de.wikipedia.org/wiki/Java_(Programmiersprache), zuletzt besucht am 12.11.2018.

[13] Wikipedia. Kodierer. https://de.wikipedia.org/wiki/Kodierer, zuletzt besucht am 12.11.2018.

[14] Wikipedia. Librarry (computing). https://en.wikipedia.org/wiki/Library_(computing), zuletzt besucht am 12.11.2018.

[15] Wikipedia. Parser. https://de.wikipedia.org/wiki/Parser, zuletzt besucht am 12.11.2018.

[16] Wikipedia. Programmbibliothek. https://de.wikipedia.org/wiki/Programmierschnittstelle, zuletzt besucht am 12.11.2018.

[17] Wikipedia. Programmierschnittstelle. https://de.wikipedia.org/wiki/Programmierschnittstelle, zuletzt besucht am 12.11.2018.

[18] Wikipedia. Synchronisation. https://de.wikipedia.org/wiki/Synchronisation, zuletzt besucht am 12.11.2018.

[19] Xuggle. Xuggle documentation website. http://www.xuggle.com/public/documentation/java/api/com/xuggle/mediatool/IMediaReader.html, zuletzt besucht am 11.11.2018.

[20] Xuggle. Xuggler documentation website. http://www.xuggle.com/public/documentation/java/api/com/xuggle/mediatool/IMediaViewer.html, zuletzt besucht am 11.11.2018.

[21] Xuggle. Xuggler documentation website. http://www.xuggle.com/public/documentation/java/api/com/xuggle/mediatool/package-summary.html, zuletzt besucht am 12.11.2018.

[22] Xuggle. Xuggler website. http://www.xuggle.com/xuggler/, zuletzt besucht am 11.11.2018.