

Labor Media Computing

Audio Visualizer



Studenten: Manuel Wehrli, Nicola Keller, Josiane Manera

Betreuung: Fredrik Gundelsweiler

Datum: 17. Januar 2015

# Inhaltsverzeichnis

[Inhaltsverzeichnis 1](#_Toc409467503)

[Einleitung 2](#_Toc409467504)

[Ausgangslage 2](#_Toc409467505)

[Aufgabenstellung 2](#_Toc409467506)

[Anforderungen 2](#_Toc409467507)

[Vorarbeit 3](#_Toc409467508)

[Recherche Technologien 3](#_Toc409467509)

[Programmablauf 4](#_Toc409467510)

[UI Prototyp 5](#_Toc409467511)

[Projektaufbau 6](#_Toc409467512)

[Technologien 6](#_Toc409467513)

[Maven 6](#_Toc409467514)

[Umsetzung 7](#_Toc409467515)

[Userinterface 7](#_Toc409467516)

[Video Decoder und Encoder 9](#_Toc409467517)

[Bildbearbeitung 10](#_Toc409467518)

[Audio Analyse 12](#_Toc409467519)

[Offene Punkte 13](#_Toc409467520)

[Projekt Installation 14](#_Toc409467521)

[Voreinstellungen 14](#_Toc409467522)

[JDK Version 14](#_Toc409467523)

[JDK Version 14](#_Toc409467524)

[Maven 14](#_Toc409467525)

[Videodatei ändern 14](#_Toc409467526)

[Bedienung 15](#_Toc409467527)

[Fazit 17](#_Toc409467528)

[Josiane 17](#_Toc409467529)

[Quellen 18](#_Toc409467530)

# Einleitung

## Ausgangslage

Im Rahmen des Modules Media Computing soll in einer abschliessenden Arbeit das Gelernte in einer praktischen Arbeit umgesetzt werden. Das Projektteam hat sich dafür entschieden als Projekt einen Audio Visualizer zu implementieren.

## Aufgabenstellung

Die Aufgabenstellung des Dozenten sieht eine Implementierung eines Audio Visualizers vor, der auf den Input der Kamera oder eines Videomateriales reagiert, sowie eine Visualisierung des analysierten Bildes und der Musik auf Basis der Frequenzanalyse ausgibt.

## Anforderungen

Gemäss Aufgabenstellung wurden folgende Anforderungen für die Laborarbeit erhoben:

* Entwicklung eines Audio Visualizers, der auf Basis des Audiomateriales, das Bildmaterial des Videos anpasst und ein neues Video erzeugt.
* Erstellung eines Userinterfaces auf dem das Video abgespielt und konvertiert werden kann.
* Analyse des Audiomateriales
* Erstellen eines Mappings wie sich das Audio auf die Bildbearbeitung auswirken soll.
* Aufsetzen einer Entwicklungsumgebung unter Versionskontrolle.
* Recherchearbeit um passende Technologien zu finden.

# Vorarbeit

## Recherche Technologien

Das Team entschied sich anfänglich dafür mit den Technologien Java (als Basis und für das Userinterface), OpenCV (für die Bildbearbeitung) und Minim (für die Audioanalyse) zu arbeiten.

Im Verlauf des Projektes stellte sich heraus, dass OpenCV zu schwerfällig für die Anwendung ist. Die Bearbeitung der Bilder mit OpenCV erwies sich, im Verhältnis zu dem was gemacht werden sollte, als sehr aufwändig. Daraufhin wurde eine simplere Technik verwendet um die Bildsequenzen zu manipulieren.

Relativ bald nach den Recherchen der Technologien wurde klar, dass keine der ausgewählten Technologien, Videodateien decoden und wieder encoden kann, so dass es uns möglich gewesen wäre das Audiomaterial mit Minim zu untersuchen oder an die Frames des Videos zu gelangen. Nach weiteren Recherchen wurde dann das Xuggler Framework eingesetzt, mit dem sowohl Audio wie auch Bilder extrahiert werden konnten.

## Programmablauf

Der Programmablauf wurde zu Beginn des Projektes grob festgelegt. Darin ist vorgesehen, dass der Benutzer über die Oberfläche seine Videodatei laden kann und die Musik wird daraufhin bereits analysiert. Auch bei benutzerdefinierten Einstellungen, wie der Film umgewandelt werden soll, findet diese Analyse statt. Weiter kann das Userinterface den Film abspielen.

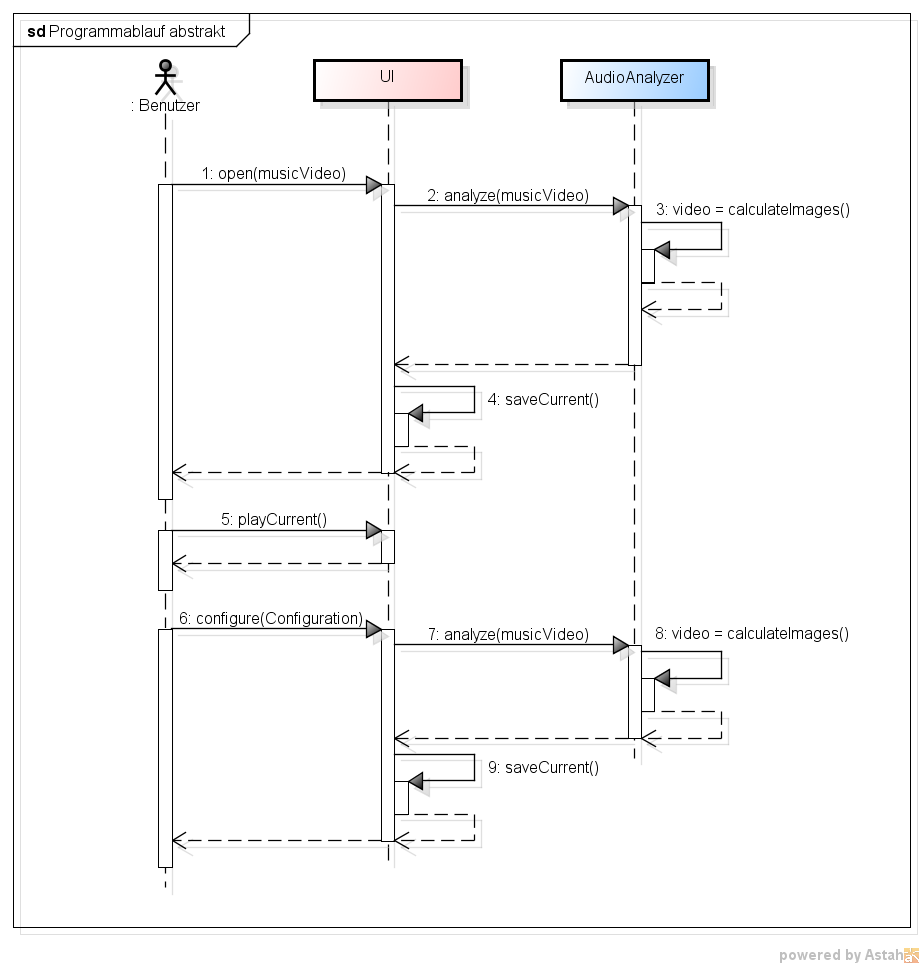


Abbildung 1: Erster grober Entwurf des Programmablaufes. Der AudioAnalyzer ist dabei ein Block aus verschiedenen Klassen die sich sowohl um die Audio Analyse wie auch das zusammenstellen des Filmes beinhaltet.

## UI Prototyp

Auf der Benutzeroberfläche sind die ursprüngliche und die manipulierte Version des Videos zu sehen. Für das Abspielen oder anhalten der Videos steht eine Toolbar zur Verfügung welche sich auf beide Videos auswirkt.



Abbildung 2: Papierprototyp des Userinterfaces.

# Projektaufbau

## Technologien

Das Projekt basiert auf folgenden Technologien:

* JDK 1.8
* Minim 2.2.0
* Xuggler 5.4

Als Versionskontrolle wird Github verwendet. Die präferierte Entwicklungsumgebung ist IntelliJ IDEA.

## Maven

Das Projekt ist als Maven-Projekt aufgesetzt. Leider war es bis zum Zeitpunkt der Abgabe nicht möglich auch das Framework *Minim* über Maven zu integrieren (vgl. Moten, 2015).



Abbildung 3: Projektübersicht mit Maven Standard Verzeichnislayout.

# Umsetzung

## Userinterface

Für das Userinterface wurde Java FX eingesetzt, das standardmässig in der JDK Version 1.8 integriert ist. JavaFX bot dem Team die Möglichkeit einfach Multimediaformate in das Userinterface einzubinden und abzuspielen. Gemäss Oracle (vgl. Castillo, 2013) sind folgende Formate unterstützt:

**Audio**

* MP3
* AIFF mit unkomprimierter PCM
* WAV mit unkomprimierter PCM
* MPEG-4 Multimedia-Container-Format mit AAC (Advanced Audio Coding) Audio

**Video**

* FLV mit VP6 Video und MP3 Audio
* MPEG-4 Multimedia-Container-Format mit H.264/AVC (Advanced Video Coding) Videoverarbeitung.

Im unteren Bereich des Userinterface befindet sich die Toolbar mit extra grossen Knöpfen. Über diese lassen sich die Videos abspielen, pausieren oder stoppen. Selbst während des Konvertierens lässt sich das UI nach wie vor bedienen. Beim konvertieren des Files (D:\Daten\Entwicklung\Java\AudioVisualizer\src\main\resources\images\film.png) erscheint zuunterst ein Fortschrittsbalken die angibt, dass der Vorgang noch nicht beendet ist. Wenn die Konvertierung stattgefunden hat verschwindet der Fortschrittsbalken und die Videos werden gestoppt.

Für die Oberfläche wurde zusätzlich ein Konfiguration Button (D:\Daten\Entwicklung\Java\AudioVisualizer\src\main\resources\images\config.png) vorbereitet, der aber noch nicht implementiert wurde und deshalb inaktiv ist. Die Idee dahinter ist, dass der Benutzer selber definieren kann wie sich die Schwindungsmerkmale des Audios farblich auf die Bilder auswirken.



Abbildung 4: Fertiges Userinterface mit Play-, Pause- und Stopp-Button.

## Video Decoder und Encoder

Beim Decodieren und Encodieren von Videodateien musste das Team als erstes nochmals eine Recherche betreiben um ein fähiges Framework zu finden, das unseren Anforderungen entspricht.

Als erstes Framework für das Decoden von Videos wurde *Humble Video* verwendet. Obwohl es momentan weiterentwickelt wird, steckte es noch in den Kinderschuhen. Es war sehr aufwändig Video und Audio zu extrahieren ohne gleich einiges über Implementierung dahinter wissen zu müssen. Einiges besser war API vom *Xuggler* Framework. Es gab weiter auch gute Dokumentationen, die beschrieben, wie man Bilder aus einem Videostream ziehen kann und ein Video aus Bilder erstellen kann (vgl. Tsagklis, 2011). Mit der Schnittstelle MediaListener kann man sich, während des Lesevorganges des Videos, informieren lassen, welches Bild und welcher Audiostream gerade gelesen wurden (siehe Listing 1). Anhand dieser Informationen kann die Audioanalyse und die Bildbearbeitung über die Schnittstelle IVideoProcessor angesprochen werden.

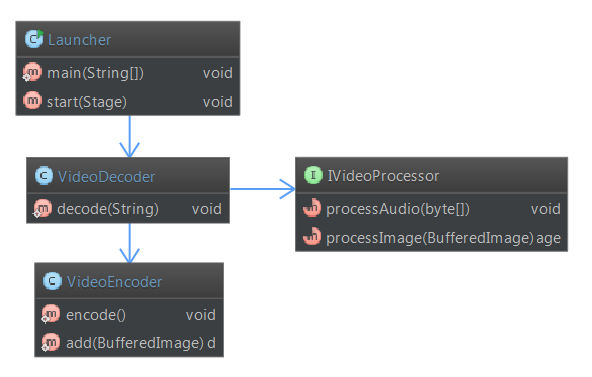


Abbildung 5: Die Applikation startet mit der Benutzeroberfläche. Wenn die Videodatei verarbeitet wird, kommt der VideoDecoder zum Zug und speichert Audio in ein Byte Array und die Frames in einzelne BufferedImages ab. Die bearbeiteten Bilder werden an den VideoEncoder übergeben der eine MP4 Datei mit Videocodec H.264 erstellt.

@Override  
public void onAudioSamples(IAudioSamplesEvent event) {  
 final IAudioSamples audioSamples = event.getAudioSamples();  
  
 final byte[] byteBuffer = audioSamples.getData().getByteArray(0, audioSamples.getSize());  
  
 processor.processAudio(byteBuffer);  
}

Listing 1: Eine der MediaListener Methoden die während des Lesevorganges von Videodateien von Xuggler aufgerufen wird. In diesem Fall kann das erhaltene Audiosample direkt als Byte Array einer Instanz von IVideoProcessor übergeben werden.

Xuggler stellt beim Schreiben einer Videodatei verschiedene Videokompressionen zur Verfügung. Für die Anwendung wurde die Videokompression H.264 verwendet, da JavaFX nur dieses Format abspielen.

## Bildbearbeitung

Bei der Bildbearbeitung gab es einige grössere Anpassungen. Zuerst stand die Verwendung von OpenCV im Raum. Das Framework war allerdings zu umfänglich für Anforderungen der Applikation. Deshalb wurden die Bilder als BufferedImage selbstständig, Pixel für Pixel bearbeitet. Dieser Prozess war jedoch äussert zeitaufwändig und dauerte auf leistungsschwächeren Computern einiges Länger als erwartet. Dieser Mangel konnte schliesslich mit einer Verarbeitungsmethode behoben werden die zusammen mit der Grafikkarte arbeitet.

Die Bilder werden über die Klasse IVideoProcessor verarbeitet, die verschiedene Strategien für die Bearbeitung von Bilder und Audio vorsieht. In der momentanen verwendeten Strategie FilterProcessor werden die Bilder mittels eines transparenten, farbigen Layers überdeckt, der das Bild aussehen lässt als wäre es eingefärbt worden (Listing 2). Die Farbe, die der Layer erhält, stammt von der Audioanalyse.

Dadurch, dass nicht mehr auf die einzelnen Pixels des BufferedImages mit getRGB() zugegriffen wird, sondern die Grafikkarte die Arbeit mit dem Zusammenführen der Layers übernimmt, wird die Konvertierungsgeschwindigkeit um ein Vielfaches erhöht.

private BufferedImage colorImage(BufferedImage image, Color newColor) {  
  
 BufferedImage result = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.*TRANSLUCENT*);  
 Graphics2D graphics = result.createGraphics();  
 graphics.drawImage(image, null, 0, 0);  
 graphics.setComposite(AlphaComposite.*SrcOver*);  
 graphics.setColor(newColor);  
 graphics.fillRect(0, 0, image.getWidth(), image.getHeight());  
 graphics.dispose();  
  
 // Bringt Bild in das richtige Ausgabeformat  
 BufferedImage finalResult = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.*TYPE\_3BYTE\_BGR*);  
 graphics = finalResult.createGraphics();  
 graphics.drawImage(result, null, 0, 0);  
 graphics.dispose();  
  
 return finalResult;  
}

Listing 2: Die Klasse FilterProcessor bearbeitet die Bilder anhand eines Filters. Auf einem halbtransparenten BufferedImage wird das Ausgangsbild gezeichnet und danach ein farbiger Layer darüber gezeichnet. Zum Schluss muss das Bild nochmals neu als 8-bit BGR Farbbild gespeichert werden, damit es ohne Komplikationen weiterverarbeitet werden kann.

## Audio Analyse

Die findet, wie die Bildbearbeitung in einer Implementierung der Schnittstelle IVideoProcessor statt. In der Methode processAudio(byte[] audio) werden konstant, vom Film eingelesene Audiosamples, als Byte Array übermittelt. Das Byte Array wird bei der Implementierung, in der Klasse FilterProcessor, in ein Float Array übergeleitet, da Minim nur mit diesem Datentyp arbeitet.

Zur Analyse des Audiosamples wird auf den Float Array eine Fast Fourier Transformation ausgeführt. Anhand der Stärke der Schwingung der Amplitude wird eine Farbe generiert (Listing 3). Die Farbe wird während der Bildverarbeitung verwendet.

public static Color getFftColor(float[] audio) {  
  
 final FFT fft = new FFT(audio.length, *DOUBLED\_SAMPLE\_RATE*);  
 fft.forward(audio);  
  
 float avgBand = 0;  
 float highestBand = 0;  
 for (int i = 0; i < fft.specSize(); i++) {  
 final float band = fft.getBand(i);  
 avgBand += band;  
 if (band > highestBand) highestBand = band;  
 }  
 avgBand /= (float) fft.specSize();  
 float colorMultiplier = (avgBand / highestBand) \* *MAGNIFIER*;  
  
 return new Color(colorMultiplier, 1 - colorMultiplier, 0, *ALPHA\_VALUE\_OF\_COLOR*);  
}

Listing 3: Die Analyse des Audios liegt ausgelagert in der Klasse AudioAnalyzer. Innerhalb eines Audiosamples wird die mittlere Schwingung der Amplitude berechnet. Anhand ihr und der höchstgemessenen Schwingung wird entschieden welche Farbe gesetzt wird. In diesem Beispiel bedeutet maximale Schwingung eine rötliche Farbe und minimale Schwingung eine grüne Farbe.

# Offene Punkte

Momentan scheinen Audio und Video nicht synchron verarbeitet zu werden. Dadurch passt die Videoausgabe gefühlsmässig nicht immer zur Visualisierung. Das Team geht davon aus, dass die Abtastrate von Audio und Bild nicht übereinstimmt. In einem nächsten Schritt müsste man mit einem Profiler analysieren ob die Audioanalyse auf die Bildbearbeitungen wartet.

# Projekt Installation

Das Projekt kann unter der <https://github.com/Violella/AudioVisualizer> bezogen werden.

## Voreinstellungen

### JDK Version

Da Xuggler kann in dem Umfang in dem es im Projekt verwendet wird nur mit einem 32bit JDK laufen. Auf 64bit stürzt die VM ab, da es Probleme mit dem darunterliegenden Framework FFmpeg gibt, das auf native DLLs zugreift. Als JDK wird die 32bit Version 1.8.0\_25 empfohlen.

### JDK Version

Durch die Verwenden eines 32bit JDKs, steht weniger Memory zur Verfügung. Dieses Problem kann gelöst werden mit den folgenden VM-Optionen:

-Xmx1g –XX:MaxHeapSize=1g

### Maven

Über Maven warden die für das Programm notwendigen Xuggler Libraries integriert. Vor dem Start des Projektes müssen diese über Maven bezogen werden und verlinkt werden.

## Videodatei ändern

Neben den beigelegten Videodateien können eigene Videodatei muss unter dem Pfad src.main.resources.video hinterlegt werden. Um die Datei zu laden, muss der Videoname bei den Konstanten unter src.main.java.ch.fhnw.meco.util.Constants.VIDEO\_NAME eingetragen werden.

# Bedienung

Mit dem Film konvertieren Button wird das aktuelle Video konvertiert. Wenn die Konvertierung zu Ende (sichtbar anhand des Fortschrittsbalkens unter der verschwindet) ist werden die beiden File gestoppt und das Resultat kann abgespielt werden.



Abbildung 6: Während der Konvertierung wird ein Fortschrittsbalken angezeigt.



Abbildung 7: Original und Resultat können abgespielt werden.

# Fazit

## Josiane

Ich war am Anfang skeptisch darüber wie wir das Projekt umsetzen würden. Ich hatte bisher noch nie an einem Projekt programmiert das Audio oder Video verarbeitet.

Das Interessanteste für mich war die Erkenntnis wie ein Video an sich aufgebaut ist aus Sicht eines Entwicklers und dass zum Beispiel MP4 nur ein Container Format ist mit separaten Video- und Audiospur die unterschiedlich codiert sind.

Es war spannend zu sehen, wie man mit einem Framework wie Xuggler ganz einfach eigene Videos erstellen kann ohne Verwendung eines teuren Videoprogrammes.

Meine nächsten Ziele für das Projekt sind das Fine Tuning zwischen Audio und Video und das Anbieten einer Konfigurationsschnittstelle, damit der Benutzer eigene Farben einstellen kann.

# Quellen

* **Moten David (2015)** *Maven or Gradle for build?* Auf GitHub am 11. Januar 2015. <https://github.com/ddf/Minim/issues/31>, abgerufen am 17. Januar 2015
* **Tsagklis Ilias (2011)** *Xuggler Tutorial: Frames Capture and Video Creation*. Auf Java Code Geeks am 23. Februar 2011. <http://www.javacodegeeks.com/2011/02/xuggler-tutorial-frames-capture-video.html>, abgerufen am 17. Januar 2015
* **Castillo Cindy (2013)** *Incorporating Media Assets Into JavaFX Applications*. Auf docs.oracle.com am April 2013. <http://docs.oracle.com/javafx/2/media/overview.htm>, abgerufen am 19. Januar 2015