Dokumentation – Emotionsbasierte Audiosynthese (EbAS)

Corinna Bohnenberger (2217555)
Hennes Römmer (2217606)
Jan Heimann (2210236)
Raoul Bickmann (2217470)

Aufgabe

Das Programm EbAS wurde im Rahmen der Veranstaltung "Audio- und Videoprogrammierung" unter dem Motto "Facial Expression Audio Synthesizer" entwickelt. Die Aufgabe hierbei war, durch Analyse von Gesichtszügen, unterschiedliche Töne zu erzeugen. Die Umsetzung konnte aber unter dieser Bedingung frei gewählt werden. Unser Projektteam, bestehend aus Raoul Bickmann, Hennes Römmer, Jan Heimann und Corinna Bohnenberger, entschied sich für eine emotionsbasierte Umsetzung. Hierbei sollten in dem aufgenommenen Gesicht Gesichtsausdrücke erkannt und zu diesem passende Audiodaten synthetisiert werden. Das Programm kann vier Gesichtsausdrücke unterscheiden. Es wird ein neutraler Gesichtsausdruck erkannt sowie ein fröhlicher, ein trauriger und die Überraschung des Nutzers.
Anschließend wird jeweils eine passende Melodie abgespielt und nachdem diese beendet ist, kann ein neuer Gesichtsausdruck erkannt werden.

Installationsanleitung

Das Programm kann entweder in Qt Creator gestartet werden oder direkt von der EBAS.exe. Für beides muss die open source Bibiliothek OpenCV heruntergeladen und Umgebungsvariablen für diese eingestellt werden. Dazu geht man unter Windows wie folgt vor:

1. Suche nach „Erweiterte Systemeinstellungen“
2. Klick auf „Umgebungsvariablen“
3. Entweder bei den Systemvariablen oder den Benutzervariablen eine Variable „OPENCV_DIR“ erstellen und dort den Speicherort für OpenCV eingeben, z.B.: C:\Program Files\opencv\build\x64\vc14
4. Die Path Variable ergänzen durch: %OPENCV_DIR%\bin

Für Start von EBAS.exe:

1. Laden Sie EBAS.rar von https://github.com/Hennnnes/face_expression_recognition/blob/master/EBAS.rar
2. Entpacken Sie die Datei in einen beliebigen Ordner.
3. Starten Sie die EBAS.exe. Das Programm sollte sich nun öffnen.

Für Start aus Qt Creator:

1. Klonen Sie das Repository von https://github.com/Hennnnes/face_expression_recognition.
2. Öffnen Sie die .pro Datei im Unterordner EBAS.
3. Ändern Sie die Build-Einstellungen auf Release.
4. Ändern Sie das Build-Verzeichnis unter Projekt -> Allgemein -> Build-Verzeichnis auf Speicherort\face_expression_recognition\EBAS, z.B.: C:\Users\Raoul\Documents\face_expression_recognition\EBAS
5. Führen Sie das Programm aus.

Bedienungsanleitung

Bevor das Programm gestartet wird, muss sichergestellt werden, dass eine funktionstüchtige Webcam und Lautsprecher vorhanden sind. Gestartet wird das Programm wie in der Installationsanleitung beschrieben, entweder über QT Creator oder direkt über EBAS.exe.
Es öffnet sich ein Fenster, das bei Klick auf den im Menü befindlichen Play-Button das Bild von der Webcam wiedergibt. Der User muss sich vor die Webcam stellen, damit sein Gesicht angezeigt wird. Es ist nur ein Gesicht zulässig, sonst wird eine Fehlermeldung geworfen, genauso wenn kein Gesicht gefunden wurde. Das Programm fängt nun an, die Gesichtszüge nach Emotionen auszuwerten und die passende Audiodaten zu generieren und abzuspielen. Sobald dieses beendet ist, kann die nächste Emotion erkannt werden. Zum Beenden des Programms kann es einfach geschlossen werden.

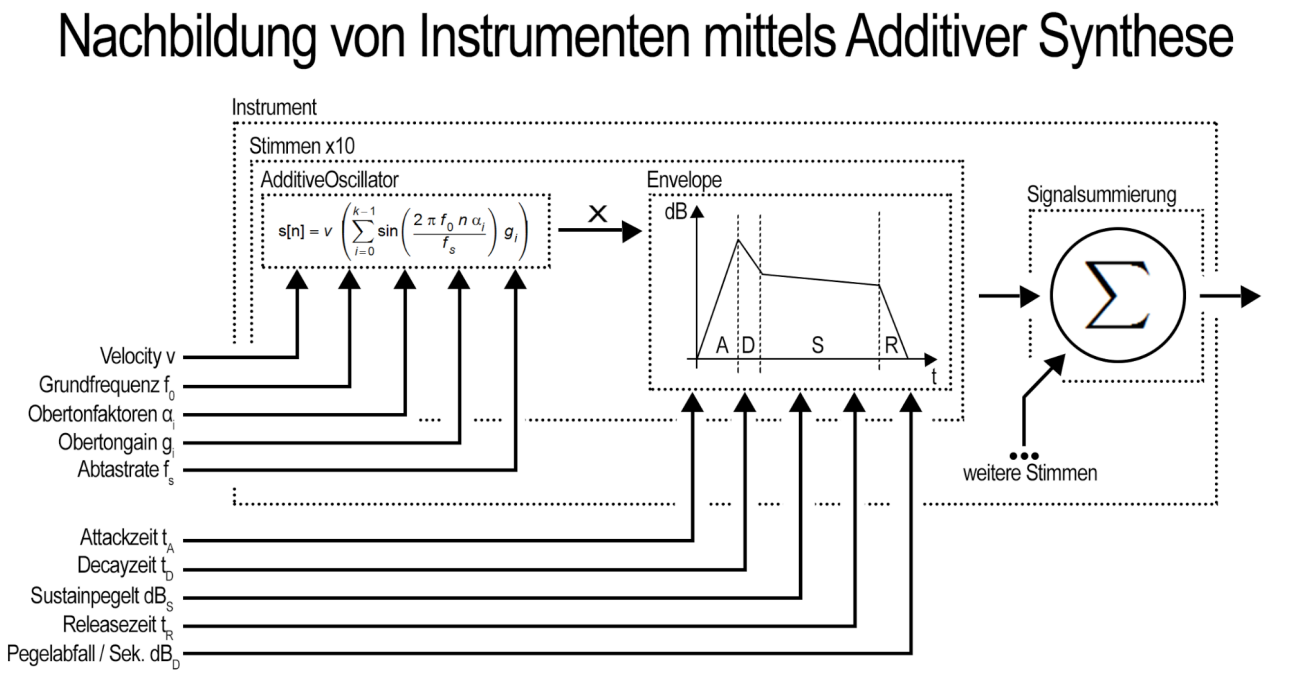
Beschreibung eines technischen Teilaspekts

Der Audio-Part des Projekts orientiert sich vom Aufbau her an den meisten gängigen DAWs (Digitals Audio Workstation). So gibt es einen Mixer, einen Sequencer und mehrere Instrumente. Dabei sind die Instrumente ein Bestandteil des Mixers und sind vergleichbar mit einzelnen Spuren in einer DAW. Jedes Instrument generiert die Samples unabhängig von den anderen Instrumenten und reicht diese dann an den Mixer weiter. Der Mixer addiert dann die Samples aller Instrumente gewissermaßen zu einem Summensignal auf. Dieses Summensignal entspricht dem Master Channel in einer DAW und kann sampleweise abgerufen werden. Eine weitere Aufgabe des Mixers ist es die Steuerdaten für die Audio-Synthese an die entsprechenden Instrumente zu verteilen, die letztendlich die Synthese durchführen. Die Steuerdaten werden von Sequencer gehandelt. Seine Hauptfunktion liegt darin, die Steuerdaten, welche als Array vorliegen, zum richtigen Zeitpunkt an den Mischer weiterzureichen, sodass die Synthese im richtigen Rhythmus angesprochen wird und entsprechend Melodien und Harmonien generiert werden können. Sowohl Sequencer als auch Mischer sind Bestandteil des AudioModules, welches alle Objekte die irgendwie mit dem Synthetisieren von Audio-Daten zu tun haben umfasst. Zudem bietet es die Möglichkeit den gesamten Synthese-Teil einfach zu initialisieren und zu steuern. Seine Funktionalitäten umfassen das Starten der Audio-Wiedergabe, Auskunft über den Status der Wiedergabe, sowie das Auswählen welche Sequenz als nächstes abgespielt werden soll.
Ein Vorteil dieser Strukturierung des Audio-Teils ist, dass er aufgrund seiner Modularität auf sehr einfache Weise erweitert werden könnte. So wäre es möglich, sowohl für das Summensignal im Mixer, als auch für die einzelnen Instrumente eine Komponente einzubinden, welche mittels digitaler Signalverarbeitung unterschiedliche Audio-Effekte realisiert. Für eine tatsächliche Umsetzung dieser Effekte war die Projektdauer allerdings zu

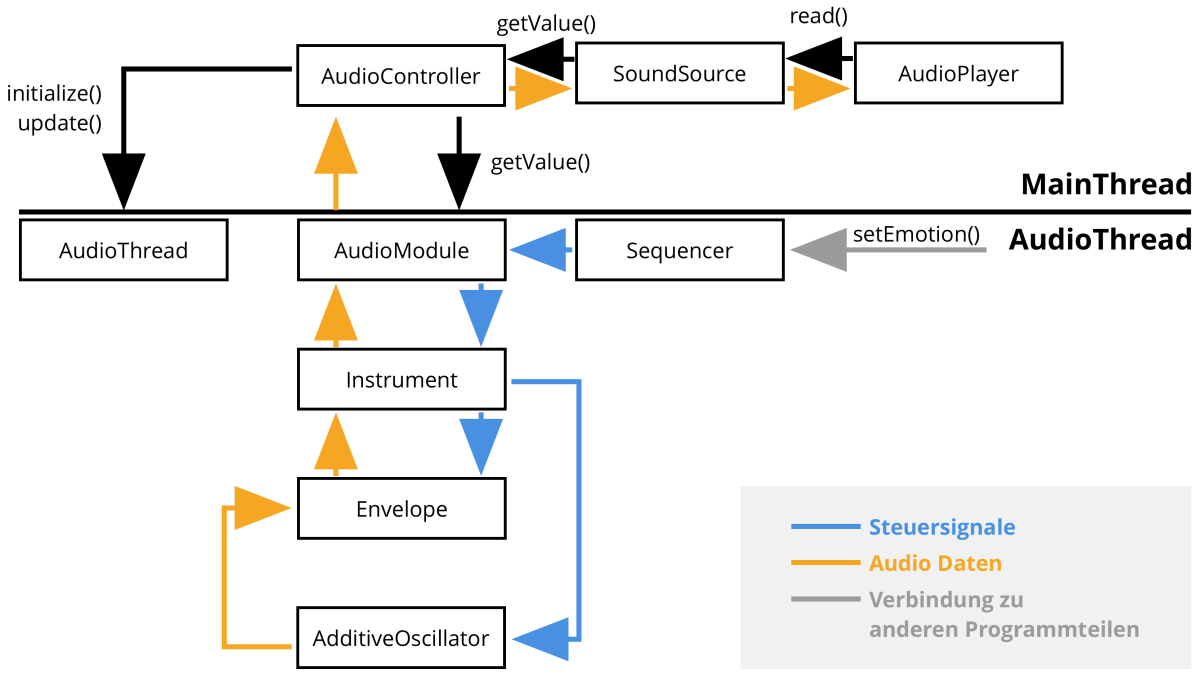
knapp. Eine weitere Möglichkeit bestünde darin, statt Instrumenten auch andere Signalquellen zu entwickeln und damit dem Mixer mit Daten zu versorgen. Zum Beispiel könnte man ein Objekt schreiben, welches Audio-Files abspielt.
Allerdings ist es mit dem AudioModule noch nicht ganz alles abgedeckt, was für die Audio-Wiedergabe wichtig ist. Denn um die Wiedergabe gleichzeitig zum Abspielen des Webcam-Videos und zur Emotionserkennung laufen zu lassen, musste der Audio-Teil parallelisiert werden. Genau das wird von dem AudioController gemanagt. Außerdem ist er die Schnittstelle für andere Programmteile zum Audio-Part. Im AudioThread laufen zum Beispiel die Aktualisierungen des Sequencers und die Verteilung und Umsetzung der Steuerdaten mit Hilfe der slots und signals welche von Qt's QObject zur Verfügung gestellt werden und eine relativ einfache Kontrolle über das Multithreading ermöglichen. Die Berechnung der eigentlichen Audio-Samples geschieht jedoch weiterhin gezwungenermaßen im Main Thread. Dies ist notwendig, da wir den AudioPlayer der AudioEngine von Prof. Plaß nutzen um den Audio-Buffer zu organisieren, einzulesen und anschließend an Qt weiterzureichen. Dies muss im Haupt-Thread passieren, da er das QMainWindow (bzw. QWidget) als Parent hat.

Kommen wir nun zur eigentlichen Synthese. Sie findet wie bereits erwähnt ausschließlich in den Instrumenten statt. Dabei sind zwei Dinge essentiell: der AdditiveOscillator und der Envelope. Zwei Objekte von den jedes Instrument jeweils mehrere besitzen kann. Dabei benötigt jeder Oscillator auch ein Envelope um korrekt zu funktionieren. Durch die Tatsache, dass ein Instrument eben über mehrere Oscillator-Envelope-Paare verfügt, ist es möglich eine Polyphonie zu erreichen. Bedeutet: Ein Instrument kann gleichzeitig mehrere Noten darstellen (eben so viele, wie es über Oszillatoren verfügt). Bei unserem Projekt sind derzeit zehn Stimmen pro Instrument realisierbar. Dadurch können nicht nur Melodien, sondern auch Harmonien wiedergegeben werden. Jedoch bringt die Mehrstimmigkeit eine Schwierigkeit mit sich: Es muss Überblick darüber gehalten werden, welche Stimmen derzeit bereits angesteuert und somit belegt ist. Sobald also eine Steuernachricht vom Mixer an das entsprechende Instrument herunter gereicht wird, schaut es selbstständig ob es einen freien Oszillator hat. Falls nicht wird die Steuernachricht einfach ignoriert. Ein Oszillator-Envelope-Paar gilt solange als „beschäftigt" bis alle folgenden generierten Samples Null sind. Dies entspricht genau der Dauer von einem ON-Befehl, der die Stimme zur Wiedergabe eines Tones anregt und ihn mit der entsprechenden Frequenz versorgt, bis die Release-Phase des Envelope abgeklungen ist. Würde man die Stimme nur von ON- bis OFF-Befehl für neue Steuersignale sperren, so würde der Ton beim Erhalten des OFF-Befehls abrupt enden, was nicht natürlich wirken würde. Die Samples, welche die einzelnen Stimmen generieren, werden vom Instrument natürlich noch zu einem Summensignal addiert, bevor sie den Mixer übergeben werden.
Wie der Klassenname AdditiveOscillator schon andeutet, bedienen wir uns des Konzepts der additiven Synthese. Diese basiert mathematisch auf der Fourier-Reihe, welche aussagt, dass sich jedes periodische Signal aus einer unendlichen Summe von harmonischen Schwingungen darstellen lässt. Da es unser Ziel war die Wiedergabe der musikalischen Motive mit synthetisierten Imitaten von echten Instrumenten stattfinden zu lassen, mussten wir diese irgendwie modellieren. Dafür eignet sich die additive Synthese ganz hervorragend, schließlich ist für die Wahrnehmung eines Instrumentes hauptsächlich seine Klangfarbe verantwortlich. Die Klangfarbe ergibt sich aus den Obertönen eines Instrumentes, also wieder einer Summe harmonischer Schwingungen. Der AdditiveOscillator erhält nun also je nachdem was man für ein Instrument umsetzen will einen Datensatz. Dieser umfasst zum einen die Frequenzverhältnisse der Obertöne zur Grundfrequenz und zum anderen die relativen

Pegeldifferenzen zur Grundschwingung, gespeichert in zwei Arrays. Sobald der Oszillator vom Instrument eine Frequenz zugeteilt bekommt, fängt er an, nach diesen Parametern zu schwingen und entsprechende Sample-Werte zu berechnen. Dazu berechnet er ganz einfach den Wert zum aktuellen Zeitpunkt der Grundfrequenz und der Frequenzen der Obertöne und addiert diese dann mit den entsprechenden Pegeldifferenzen gewichtet zu einem Gesamtsignal. Dieses Signal hat dann die charakteristische Signalform des simulierten Instruments und weist die gleiche Klangfarbe auf. Formuliert man das ganze im Frequenzbereich, so kann festgehalten werden, dass hier das kontinuierliche Frequenzspektrum eines realen Instrumentes digital und somit diskret nachgebildet wird.
Um den erzeugten Klang aber eindeutig mit einem echten Instrument in Verbindung bringen zu können, fehlt noch eine zeitabhängige Beeinflussung des Signals. Genau das ist die Aufgabe des Envelope. Wobei es sich um einen Hüllkurven-Generator handelt. Der bei unserem Projekt implementierte Envelope ist größtenteils mit dem aus der Vorlesung bereits bekannten Envelope identisch. Auch er vereinfacht das zeitliche Verhalten eines Instruments auf die vier Phasen Attack, Decay, Sustain und Release (kurz ADSR). Allerdings wurde er um eine zusätzliche Funktion erweitert. Während ein herkömmlicher ADSR-Envelope in der Sustain-Phase einen konstanten Pegel erzeugt, ist unsere Version darüber hinaus in der Lage eine Abklingen des Tones in der Sustain-Phase zu simulieren. Das ist wichtig, da während Streich- und Blasinstrumente den konstanten Pegel halten können, Zupf- und Schlaginstrumente eher dazu neigen stetig an Pegel in der Sustain-Phase zu verlieren. Damit ein authentischer Klangeindruck erzeugt werden kann, mussten wir diesen Effekt natürlich nachempfinden. So kann dem Envelope ein Pegelabfall pro Sekunde zugeteilt werden und zwischen abklingenden und konstanten Pegel entschieden werden. Außerdem gibt es noch ein paar Funktionalitäten, die mitbestimmen ob diese Stimme grade als „beschäftigt“ gilt. Gesteuert wird der Envelope ebenfalls mit ON- und OFF-Befehlen, die er vom Instrument erhält.



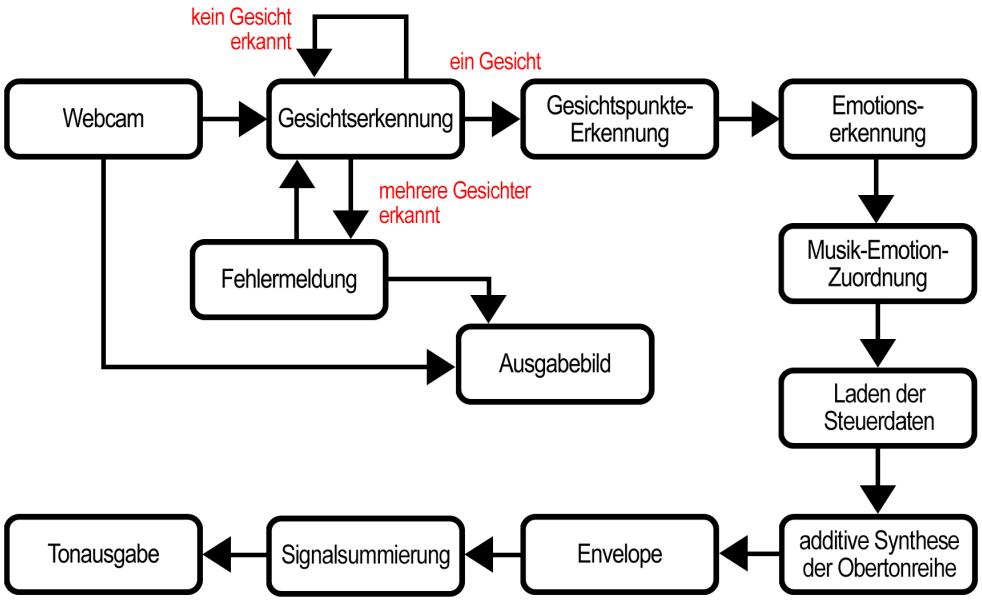
Realisierung des Audio Teils



Systemarchitektur

Das Bild der Webcam wird mit Hilfe von Opencv abgefangen und ausgewertet. Hier ist es möglich auf jeden Frame einzeln zuzugreifen. Für die Verarbeitung jedes einzelnen Frames verwenden wir Dlib. Dort gibt es eine Funktion, die das Gesicht erkennt und einzelne Punkte innerhalb eines Gesichts zurückgibt. Auf Basis dieser Punkte haben wir einen Algorithmus entwickelt, welcher aus den einzelnen Punkten im Gesicht vier unterschiedliche Gesichtsausdrücke liest. Die hier möglichen Ausdrücke sind: überrascht, traurig, glücklich und ein neutraler Ausdruck. Der Aktuelle Gesichtsausdruck wird in der oberen linken Ecke des Programmes angezeigt.
Der Gesichtsausdruck wird mit jedem Frame dem Audio-Teil unseres Programms übermittelt. Für jeden Ausdruck gibt es eine entsprechende kurze Melodie, die abgespielt wird. Eine neue Melodie wird nur abgespielt, wenn die vorherige Melodie bereits zu Ende ist und es keinen Fehler gibt. Fehler können zum Beispiel kein Gesicht im Bild, oder mehr als ein Gesicht im Bild sein.
Es gibt Arrays mit unterschiedlichen Steuerdaten für jeden Gesichtsausdruck. Diese Steuerdaten werden von dem Sequenzer eingelesen. Der Sequenzer steuert die Audiosynthese zu der richtigen Zeit an.
Die Audiosynthese findet innerhalb eines Instrumentes statt. Es wäre möglich mehrere unterschiedliche Instrumente zu implementieren, derzeit gibt es nur ein Klavier. Diese Instrumente werden von dem Audiomixer gemischt.
Des Weiteren gibt es ein AudioModule, welches die unterschiedlichen Mixer und Sequenzer organisiert.
Abschließend gibt es einen AudioController welcher die unterschiedlichen AudioModules in eigene Threads bringt. Die Daten der Threads werden an die Tonausgabe von Herrn Plaß weitergereicht.

Flussdiagramm Systemarchitektur



Was hat funktioniert / was hat nicht so gut funktioniert?

Wir haben uns dazu entschlossen unsere Gruppe in zwei kleine Untergruppen zu teilen. Die Video- und die Audiogruppe.
Gut geklappt hat die Projektorganisation und die Arbeit in diesen Untergruppen.
Bei der Videogruppe gab es am Anfang einige Schwierigkeiten dlib zu einzubinden. Der Aufwand hat sich jedoch gelohnt, da die Gesichtserkennung sehr zuverlässig funktioniert. Bewegungen einzelner Gesichtspunkte werden jedoch nur um die Mundpartie zuverlässig verfolgt. Dadurch konnte hauptsächlich diese für die Gesichtsausdruckserkennung verwendet werden.
Dieses funktioniert nun jedoch relativ gut, ist jedoch immer etwas abhängig von der Person und der Ausprägung ihrer Mimik. Auch mangelnde Kontrastunterschiede und schlechte Belichtung können ein Problem darstellen.
In der Audiogruppe war ursprünglich der Plan die Audioausgabe über Midi-Files zu realisieren. Hierfür sollte die Library “jdksmidi” verwendet werden. Nachdem die Schwierigkeiten beim Einfügen dieser Bibliothek behoben und der Code entsprechend geschrieben war, funktionierte die Ausgabe in der Theorie. Es wurden die Werte in der richtigen Taktung ausgegeben, jedoch fehlte der Ton teilweise. Bei diesem Ansatz hatten wir die Daten direkt beim Abspielen aus den Midi-Files gelesen. Wir versuchten daraufhin, die Daten zunächst in Arrays zu speichern, aber auch dies lieferte fehlerhafte Ausgaben. Da wir nicht mehr viel Zeit hatten, um die Fehler zu finden und zu korrigieren, entschieden wir uns letztendlich gegen die Verwendung von Midi-Files und erstellten dafür Vektoren mit den Steuerdaten für die Audiosynthese.
Des Weiteren brauchten wir für das richtige Verwenden von Threads in C++ länger als erwartet. Hier gab es das Problem, dass Methoden trotz Verwenden eines Threads auf dem Mainthread ausgeführt wurden. Letztendlich schafften wir es auch diese Probleme rechtzeitig zu beheben und konnten mit Threads die gleichzeitige Ausgabe von Audio und Video realisieren.