Projektbericht
„Air-Synthie“

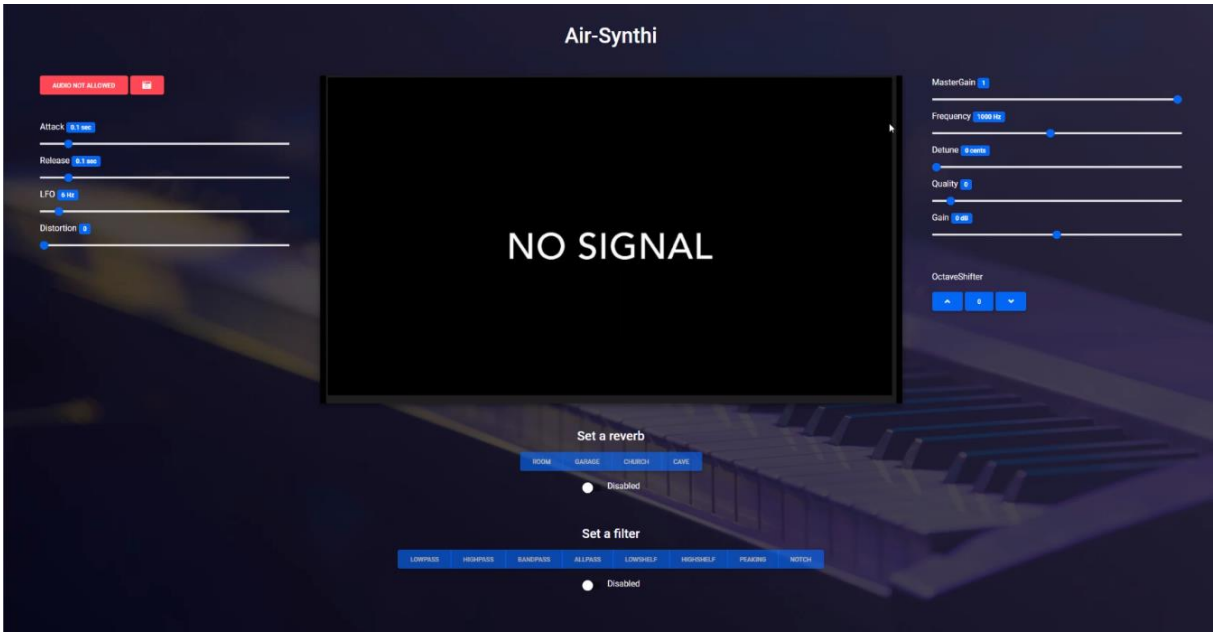
Audio-Video-Programmierung
SoSe2020
Bachelor Medientechnik
12.01.2021

Hannah Lenz 2339432
Tobias Rawald 2380225
Yannic Kühn 2382131

1. Benutzeroberfläche Webseite
2. Technische Besonderheiten
3. Darstellung
4. Aufwandsschätzung Auswertung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Benutzeroberfläche Webseite



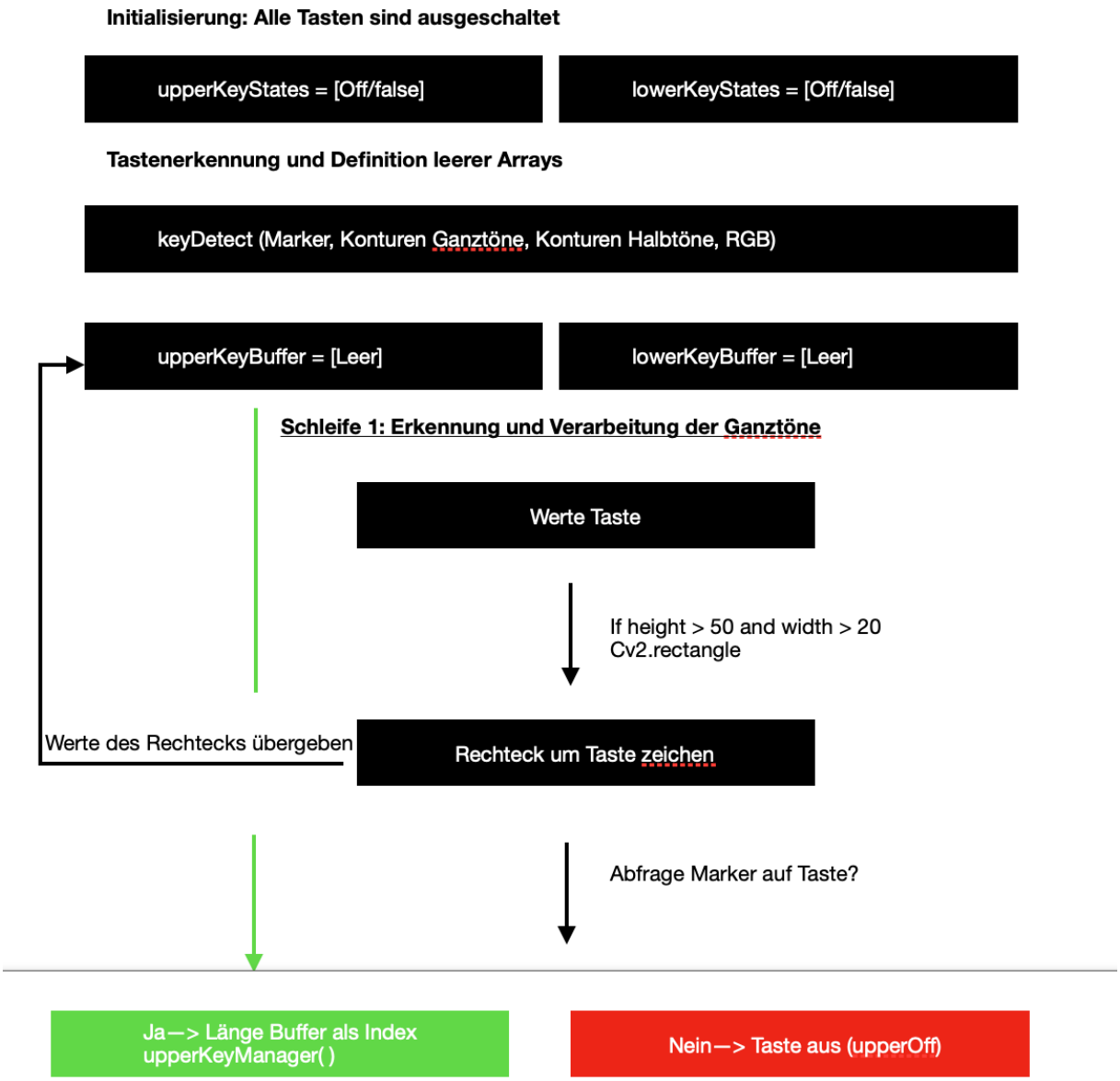
Oben links befindet sich ein Button, der es durch eine Klick-Interaktion ermöglicht, Sounds über die Webseite abzuspielen, da diese initial zunächst vom Browser blockiert werden. Über den daneben liegenden Button lässt sich ein Programm-Fenster auswählen, welches anstelle des Platzhalters angezeigt werden soll. Wird in unserem Fall zunächst das Python-Programm gestartet, können wir anschließend über die Webseite das neu geöffnete Fenster mit der Live-Video Quelle auswählen. Alternativ wird ein Platzhalterbild angezeigt, solange kein Fenster ausgewählt ist. Links und rechts des Medieninhalts lassen sich verschiedene Regler einstellen, die den Sound verändern. Unterhalb des Medieninhalts lassen sich via Buttons verschiedene Reverbs und Filter anwenden. Möchte man keinen verwenden, so lassen sie sich einfach komplett ausstellen. Ein weiteres Feature ist der Octave-Shifter, welcher per Up- und Down-Buttons die Oktaven hoch- bzw- runterstuft. Da es keine Anzeige über die aktuelle Oktave gibt, lässt sich die Oktave des C´ über den 0-Button wieder erreichen.

Technische Umsetzung (Besonderheiten)

```
def keyDetect(marker, contours, contours2 , r, g, b):
    upperKeyBuffer = []
    lowerKeyBuffer = []
    for cnt in contours:
        x,y,width,height = cv2.boundingRect(cnt)
        if height > 50 and width > 20:
            cv2.rectangle(frame,(x,y),(x+width,y+height),(b, g, r),2)
            lowerKeyBuffer.append((x, y, width, height))
            if (marker[0] > x and marker[0] < x + width and marker[1] > y and marker[1] < y + height):
                lowerKeyManager(len(lowerKeyBuffer))
            else:
                lowerOff(len(lowerKeyBuffer))
    for cnt in contours2:
        x,y,width,height = cv2.boundingRect(cnt)
        if height > 50 and width > 20:
            cv2.rectangle(frame,(x,y),(x+width,y+height),(b, r, g),2)
            upperKeyBuffer.append((x, y, width, height))
            if (marker[0] > x and marker[0] < x + width and marker[1] > y and marker[1] < y + height):
                upperKeyManager(len(upperKeyBuffer))
            else:
                upperOff(len(upperKeyBuffer))
```

Die wichtigste Funktion in dem Programm ist die KeyDetect-Funktion, die wie der Name schon sagt, die jeweiligen Tasten erkennt und maskiert. Sie beinhaltet zum einen zwei Arrays – eines mit den sechs Halbtönen, eines mit den neun Ganztönen – die initial alle auf Off gesetzt sind. Außerdem werden für die oberen und unteren Tasten zwei leere Buffer-Arrays definiert. Die KeyDetect-Funktion selber beinhaltet Daten zu dem Marker, mit dem die Tasten angeschlagen werden, den Kanten der einzelnen Tasten und deren Farbe und teilt sich in zwei Schleifen für die Ganz- und Halbtöne auf. Zunächst werden die x, y, Breite und Höhe Werte mittels *cv2.boundingRect(cnt)* in eigenen Variablen gespeichert. Bildfehler durch zu kleine Kästchen werden im nächsten Schritt herausgefiltert, indem die Grenzen für Höhe (>50) und Breite (>20) gesetzt werden. Die Funktion *cv2.rectangle* zeichnet nun um die jeweilige Taste ein Rechteck und die Werte des Rechtecks werden ein das passende vorhin erwähnte leere Array übergeben. Es folgt sie Abfrage darüber, ob der Marker sich innerhalb des erkannten Vierecks befindet. Wenn das der Fall ist, wird die Länge des Buffers (als die Länge des Anschlags) als Index an den *lowerKeyManager* übergeben, welcher sich um das Senden der Töne kümmert. Wird kein Marker innerhalb eines Rechtecks erkannt, so bleibt die Note ausgeschaltet.

Darstellung



Auswertung Aufwandsschätzung

***Meilenstein:** Grundgerüst der Software sollte stehen, also: Ton-Tastaturanschläge sollten zuverlässig erkannt werden.
****Meilenstein:** Das Oktav-Shifting sollte nun funktionieren, die Klangqualität sollte nun angepasst worden sein, sodass bis zur Abschlusspräsentation noch an Stellschrauben wie Latenzen und Design gearbeitet werden kann.

Zeitlicher Aufwand:
20 PS bis zum ersten Meilenstein
15 PS bis zum zweiten Meilenstein/Prototyp
15 PS bis zur Abschlusspräsentation
10 PS für die Nachbereitung und Dokumentation
60 PS Gesamt

Der Workflow war nicht ganz der, der zuerst geplant war. Der erste Schritt bestand darin, eine zuverlässige Tastenerkennung zu programmieren. Dafür musste zunächst digital ein Video erstellt werden, welches keinerlei Belichtungseinflüssen unterliegt. Danach folgte die Tastenerkennung mit MIDI-Verknüpfung sowie das Grundgerüst der Webseite. Die Zusammenführung der verschiedenen Arbeitsbereiche führte zu weiteren Problemen, beispielsweise konnte das Live-Videosignal nicht gleichzeitig über Python und auf der Webseite eingebunden werden.

Tatsächlicher Aufwand

Recherche und Vorgehen, Planung	4h
Bis Tastenerkennung Testvideo (inkl. Videoerstellung)	20h
Grundgerüst Webseite, einbinden von Framework etc.	10h
MIDI-Verknüpfung	8h
Setup Live-Video (inkl. Basteln)	6h
Einbindung Live Video sowie restliche Arbeiten an der Webseite	14h
Vorbereitung Präsentation, Nachbereitung, Code aufräumen, Bericht	10h
Gesamt:	72h

Die Aufwandsschätzung von Anfang war also zu gering angesetzt, da hier nicht genug an unerwartete Probleme gedacht wurde.

Den kompletten Code zu diesem Projekt finden Sie im Git-Repository über den folgenden Link: https://github.com/YannicKuehn/AirSynthi.git .

Die Webseite wurde mit Hilfe des Framework MDBootstrap erstellt, welches einem ermöglicht das Styling mit vorgefertigten JavaScript- und CSS-Klassen zu optimieren. https://mdbootstrap.com/ .

Unsere Dateien befinden sich im **img**- und **src**-Verzeichnis.