Dokumentation zum mathematisch–naturwissenschaftlichen Labor Mathesis

Das Orchester ist Programm

Henriette Behr Henriette Rilling Max Wehner
Robin Krüger

Sommersemester 2016

mathematisch-naturwissenschaftliches Labor Mathesis
Sommersemester 2016

Dozent: Dr. Stefan Born
Technische Universität Berlin
Fakultät II für Mathematik und Naturwissenschaften

Studierende:
Henriette Behr, Orientierungsstudium MINT$^{\text{grün}}$
Henriette Rilling, Orientierungsstudium MINT$^{\text{grün}}$
Max Wehner, Orientierungsstudium MINT$^{\text{grün}}$
Robin Krüger, Orientierungsstudium MINT$^{\text{grün}}$

Im folgenden Dokument wird die Entstehung des Python-Programmes »Das Orchester ist Programm« dokumentiert. Dieses Programm hat die Aufgabe über die Bewegungen eines Dirigenten ein Orchester zu steuern. Um dies zu realisieren wurde sich mit Bildverarbeitung auseinandergesetzt, sowie mit Midi-Dateien. Besondere Beachtung fand bei der Bildverarbeitung die Python-Bibliothek *OpenCV*. Die Dokumentation beschreibt dabei nicht nur das fertige Programm, sondern ebenso den Prozess, der zum Endprodukt führte. Gleichzeitig stellt die Dokumentation den Abschluss des mathematisch-naturwissenschaftlichen Labors Mathesis aus dem Orientierungsstudium MINT$^{\text{grün}}$ dar.

Inhaltsverzeichnis

1. Projektfindung

Von Seiten der Laborleitung gab es keine festgesetzten Themengebiete oder Aufgaben, die erfüllt werden sollten. Lediglich sollte die Fragestellung durch ein Programm lösbar sein. Dieses Programm sollte zusätzlich in der neu erlernten Sprache *Python* geschrieben werden. Die Brainstormingrunden fanden zu Beginn mit der gesamten Laborgruppe in einem online-Dokument statt. Einige der Projektideen, welche zusammengetragen wurden, standen wie folgt in dem Dokument:

- Im Physikpraktikum gibt es einen bisher nicht genutzten Luftkissentisch, aus dem ein Experiment aufgebaut werden soll. Dazu müsste eine Kamera über dem Tisch angebracht werden und dann versucht, die Bewegungen der Pucks auf dem Tisch (die z.B. ein Gas simulieren) zu erkennen und in analysiserbare Daten umzuwandeln. Man kann im zweiten Schritt aus den Daten viele Größen der statistischen Mechanik berechnen: Mittlere freie Weglänge, Druck, Maxwell-Verteilung,...
- Thermodynamikversuche: Druck, Temperatursensoren anbringen, Daten auslesen und was draus machen.
- Mitarbeit bei einer Ausstellung über Emanuel Goldberg (in Dresden)
- ähnlich wie Shazam oder Soundcloud Audiodateien aus einem Pool anhand von Audiosignalen erkennen
- Analyse Fußgängerampel Ernst-Reuter-Platz. Wann wie wo gehen die Leute? Gibt es einen bei rot laufen Schwarmeffekt? bei eskorten...
- Supermarkteinkaufsschlange: Verhalten der Leute
- Zeitungsschlagzeilen von großen Tageszeitungen sammeln. Per zufallsgenerator neue Schlagzeilen erstellen.
- Bilder sammeln zu bestimmten Thema (z.B. Hashtag), analysieren ein »Idealbild«kommentieren
- 3D drucker!! Was auch immer
- Zufallskochrezepte generieren
- Bilder auf Bearbeitung untersuchen (Photoshop etc)
- Web-crawler, z.B. Jobangebote nach bestimmten Kriterien listen

- Fußallturnier zur ïdealenËrmittlung des besten Spielers, mit wechselnden Mannschaften, sodass die Bewertung aussagekräftig wird, mit der Möglichkeit, Spieler zwischenzeitlich hinzuzufügen oder zu entfernen
- Fraktale zeichnen und rendern (+ evtl. Parallelisierung auf Grafikkarten mit Cuda)
- Smart-Home/ Smart-Watch
- optimaler Radwegeplan für Berlin
- Betriebssoftware für einen aus alten CD-Laufwerken zusammengebauten 3D-Drucker (Von der Ansteuerung der Schrittmotoren bis zum Verarbeiten von entsprechenden Dateien)
- Analyse von Bewegungen eines Dirigenten und darüber Steuerung von Musik (Geschwindigkeit, Dynamik,...)
- Portrait-Zeichner: Man lädt ein Portrait in das Programm und es liefert eine (künstlerische) Zeichnung des Gesichts zurück
- KI für ein (rundenbasiertes) Spiel

Bevor die Projekte exakt definiert wurden, bildeten sich bereits kleinere Gruppen, welche zusammenarbeiten wollten. In unserer Gruppe bestand der Konsens im Projekt Videoerkennung und Musikverarbeitung zu verwenden. Schließlich einigten wir uns auf die im folgenden Abschnitt beschriebene Projektidee.

2. Projektidee

2.1. Motivation

Die meisten Menschen hatten es schon einmal in irgendeiner Weise mit der Person des Dirigenten/der Dirigentin zu tun. Dies kann von weitem bei einem Besuch im Konzerthaus geschehen sein oder vielleicht von nahem im Orchester, in dem man mitspielt. Dem Dirigenten kommt dabei eine wichtige Aufgabe zu. Er koordiniert beim Spiel die einzelnen Instrumente im Orchester/Chor. Er sorgt dafür, dass eine Geschwindigkeit vorgegeben wird, Dynamik angewandt wird, Phrasierungen ausgespielt werden, Einsätze wahrgenommen werden und vieles mehr. Es ist ein wichtiges und verantwortungsvolles Amt.

Wie wäre es allerdings wenn man sich selbst in diesem Amt einmal ausprobieren könnte. Den Einsatz zu einem großartigen Stück zu geben und mit dem Körper dafür zu sorgen, dass das Orchester das Stück nicht langweilig herunterspielt sondern dynamisch arbeitet und die Fermaten eine angenehme Länge haben. Ein Programm welches ein Orchester simuliert und dieses nach dem Dirigat vor einer Kamera steuert, wäre wohl eine interessante Möglichkeit dies einmal auszuprobieren.

2.2. Ziel

Ziel war es bis zum Ende des Semesters ein Programm zu schreiben, welches ein Musikstück abspielt und dabei durch Gesten folgende Parameter verändert werden können:

- Geschwindigkeit
- Dynamik
- Fermaten und Einsätze
- zur Taktarkt passendes Musikstück abspielen

3. Methodik

3.1. Organisation

Die Arbeit am Projekt fand in den wöchentlichen Laborsitzungen statt. Manchmal wurde darüber hinaus am Wochenende an den Teilprogrammen weitergearbeitet. Zudem fand in der vorlesungsfreien Zeit ein dreitägiger Block des Labors statt, in welchem noch einmal intensiv an den Projekten gearbeitet wurde. Zu Beginn jeder Sitzungen besprachen wir die Ziele für den Tag. Diese wurden gleich in der Wiki dokumentiert. Meist teilten wir uns anschließend in Zweiergruppen auf, die eines der Teilziele bearbeiteten.

3.2. Werkzeug

Jedes Gruppenmitglied benutzte für die Programmierarbeit seinen eigenen Laptop. Drei waren dabei mit Windowsbetriebssystem in Verwendung, einer unter Linux. Dieser Unterschied führte zu einigen Komplikationen während der Projektarbeit, wie im Abschnitt 6.2 genauer erläutert wird.

Zur Bilderkennung wurden drei eingebaute und eine externe Webcam verwendet. Auch hier kam es zu Problemen (vgl. Abschnitt 6.2).

Für das Dirigat stellte Robin Dirigierstäbe her. Hierbei handelte es sich um grünlackierte Styroporbälle auf metallenen oder hölzernen Schaschlickspießen (näheres zur Farbwahl siehe Abschnitt4.4).

4. Projektverlauf

4.1. Interaktion mit Kameras

Da wir das Dirigat mittels Video erkennen wollten, mussten wir uns damit beschäftigen, wie wir auf die Kameras zugreifen und die entsprechenden Daten verarbeiten können. Dazu empfahl uns der Laborleiter die Pyhton-Bibliotihek OpenCV. Der erste Schritt war also die Installation der Bibliothek. Diese gelang problemlos. Im folgenden wurde ein Minmalprogramm (Code 4.1) geschrieben, mit welchem die Bilder der Kamera ausgegeben und gespeichert werden konnten. Dieses Programm ist im folgen den zu sehen. Beim speichern stellte die Wahl eines passenden Codecs eine besondere Herausforderung dar. Zudem muss für das Abspielen des Videos ein VLC-Mediaplayer installiert sein.

```
1  import numpy as np
2  import cv2
3
4  cap = cv2.VideoCapture(0)
5
6  # Define the codec and create VideoWriter object
7  # Syntax Videowriterobjekt: cv2.cv.CV_FOURCC([filename, fourcc, fps,
       frameSize[, isColor]]) fps = frames per second
8  #speichert video mit dateinamen output in laufendem ordner
9  fourcc = cv2.cv.CV_FOURCC('F','M','P','4') #legt codec des videos fest
10 out = cv2.VideoWriter('output.avi', fourcc, 24.0, (640,480))
11
12 while(cap.isOpened()):
13     ret, frame = cap.read() #ret= returnvalue
14     if ret==True:
15
16         cv2.imshow('frame',frame)
17         out.write(frame)
18
19         if cv2.waitKey(1) & 0xFF == ord('q'):
20             break
21     else:
22         break
23
24 # Release everything if job is finished
25 cap.release()
26 out.release()
27 cv2.destroyAllWindows()
```

Code 4.1: Webcamansteuerung und Videoerzeugung

4.2. Objekterkennung

Wir haben uns überlegt, wie man die Bewegungen des Dirigenten auswerten kann. Dabei gingen wir davon aus, dass man die Hände des Dirigenten verfolgen sollte. Aus diesem Grund beschäftigten wir uns mit der Erkennung von Körperteilen. In unserem Fall Gesicht und Händen. Für diesen Zweck verwendeten wir das Erkennungsverfahren mit Haarcascaden. Haarcascaden sind das Wissen über typische Merkmale von beispielsweise Gesichtern. Sie werden von einem Programm erzeugt, welches verschiedene Fotos auswertet und dabei dazu lernt. Da die Erstellung von Haarcascaden mit einer großen Menge an Bildmaterial und Rechenleistung verbunden ist, verwendeten wir bereits fertige Haarcascaden. Nichtsdestotrotz beschäftigten wir uns mit deren Funktionweise und der Fourieranalyse von Bildern.

Das Einbinden der Haarcascaden erfolgte nach kurzer Recherche mittels der OpenCv-Funktion `cv2.CascadeClassifier`. Wir beschränkten uns auf das Erkennen von Händen und Köpfen. Dabei ist zu bemerken, dass für die Handerkennung verschiedene Haarcascaden existieren. Es besteht ein Unterschied zwischen Handflächen, Fäusten und Händen. Durch Tests ermittelten wir die beste Erkennung. Jedoch stellte sich heraus, dass die besten Ergebnisse erzielt werden, wenn alle drei Merkmale geprüft werden und so eine maximale Handerkennung erzeugt wird, die eine erhebliche Rechenleistung benötigt.

Um die erkannten Objekte zeichneten wir mit Hilfe des Befehls `cv2.rectangle` ein buntes Rechteck.

4.3. Bewegungserkennung

4.3.1. Optischer Fluss

Ein weiterer wichtiger Aspekt für die Dirigaterkennung ist die Bewegung im Bild. Hierfür gibt es verschiedene Methoden in den Bilddaten Bewegungen zu detektieren. Wir beschäftigten uns mit der Taylor-Approximation und dem optischen Fluss. Der optische Fluss berechnet aus dem Unterschied zwischen zwei Bildern einen Bewegungsvektor. Wir hofften aus diesen Bewegungsvektoren mehr Informationen über das Dirigat des Dirigenten zu erhalten. Zur Untersuchung der Bewegungsvektoren untersuchten wir die farbige Darstellung dieser. Abb. 4.1 zeigt ein Standbild aus einem solchen Video.

Für unsere Zwecke erwies sich diese Methode jedoch nicht als nützlich. Stattdessen versuchten wir durch die Daten aus der Objekterkennung, Bewegungen zu ermitteln. Wir plotteten die Daten (Abb. 4.2) und versuchten auswertbare Stellen zu ermitteln. Bei diesem Unterfangen stellen wir jedoch fest, wie lange die Erkennung von Objekten über Haarcascaden benötigt und suchten nach einem alternativen Verfahren.

4.4. Farberkennung

Mithilfe eines farbigen Dirigierstabs, der sich farblich vom Hintergrund unterscheidet, sollte von nun an dirigiert werden. Folglich schrieben wir ein Programm, welches uns

Abbildung 4.1.: Optischer Fluss

ermöglichte, eine in unserem Fall grüne Kugel auf der Spitze des Dirigierstabes zu verfolgen und die Position in Koordinatenform zu ermitteln. Bei dem Programm spielt die Funktion `cv2.inRange` eine wesentliche Rolle. Sie hilft uns eine Maske (Abb. 5.1) zu erstellen, die allen Werten der Quelle, welche innerhalb festgelegter Werte liegen, ein `True` zuzuordnen und allen Werten, die außerhalb liegen ein `False`. Hierbei ist es weitaus einfacher das zu analysierende Bild nicht im BGR-Farbraum zu betrachten, sondern im HSV-Farbraum. Dieser besteht ebenfalls aus drei Werten, dem Hue, der Saturation und dem Value. Hiermit ist es uns einfacher möglich, die `inRange`-Funktion zu verwenden. Die verwendeten Werte müssen regelmäßig an die Umgebungsbedingungen angepasst werden, da das Programm sonst nicht korrekt funktioniert. Letztlich ersetzte schnell eine Kalibrierungsfunktion das lästige manuelle Anpassen der `inRange`-Farbwerte. Die `bitwise_and`-Funktion ermöglicht es uns dann letztlich aus dem Originalbild nur die Pixelwerte anzeigen zu lassen, die auf der gleichen Position, wie ein True der Maske liegen. Um die Position des getrackten Objektes in einer Koordinate zu bestimmen nutzen wir noch `numpy.mean`, da die Koordinate annähernd immer dem Mittelwert des Arrays der True-Postitionen entspricht. Hierbei spielt die Genauigkeit der zu Beginn gewählten Range-Werte eine wichtige Rolle.

4.5. Bewegungsanalyse

Da wir nun fähig waren die aktuelle Position des Dirigierstabes auszulesen, machten wir uns Gedanken über die Verarbeitung dieser Information. Zuerst betrachteten wir die

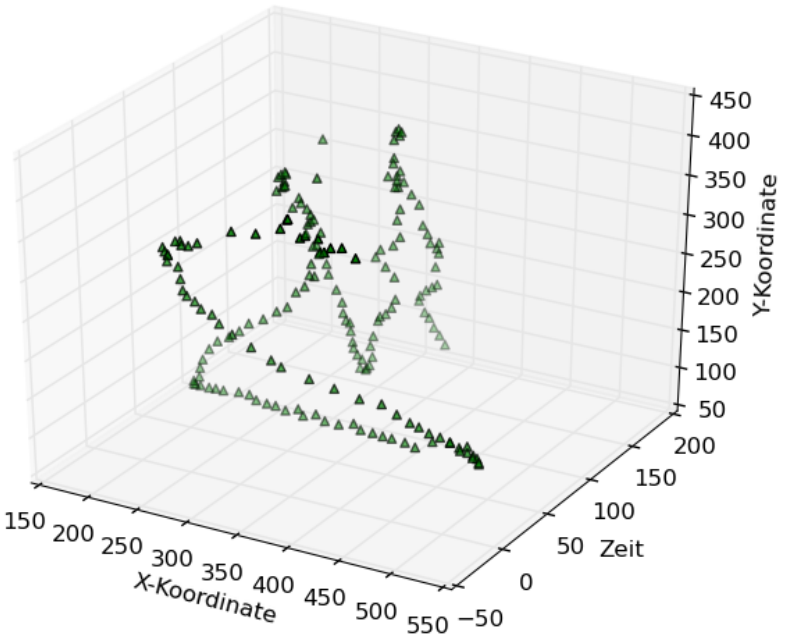


Abbildung 4.2.: Plot von Objektkoordinaten über die Zeit

X- und Y-Koordinatendifferenz der Stabposition in zwei aufeinanderfolgenden Frames. Um die ermittelten Daten zu veranschaulichen ließen wir sie mithilfe von Matplotlib in Diagramme zeichnen. Diese Veranschaulichung (Abb. 4.3) half uns dabei zu erkennen, dass wir unsere Kenntnisse aus dem Bereich der Analysis im Hinblick auf Extrema und Ableitungen für unser Programm nutzen können.

4.5.1. Dirigierfrequenz

Eine Extremstelle in der Dirigierbewegung kann also über einen Vorzeichenwechsel der Differenz von Folgeframes detektiert werden. Mit der Detektion der Extremstellen waren wir kurz davor die Dirigierfrequenz der Bewegung herauszulesen, welche später die Schnelligkeit der Musik steuern sollte. Dieses Ziel erreichten wir indem wir die Systemzeit nutzten, um die zeitlichen Abstände zwischen zwei Extremstellen zu messen und in einer Liste zu speichern. Die Dirigierfrequenz war somit der erste Parameter des Dirigenten, den der Analyseteil feststellen konnte. Letztlich wurde für den weiteren Projektverlauf die Dirigierfrequenz immer in BPM (beats per minute) umgewandelt. Um die Schläge Pro Minute (BPM) zu emitteln, welche in der Musik oft als Schnelligkeitsangabe verwendet werden, werden die Zeitpunkte von zwei aufeinanderfolgenden Extrema von einander ab-

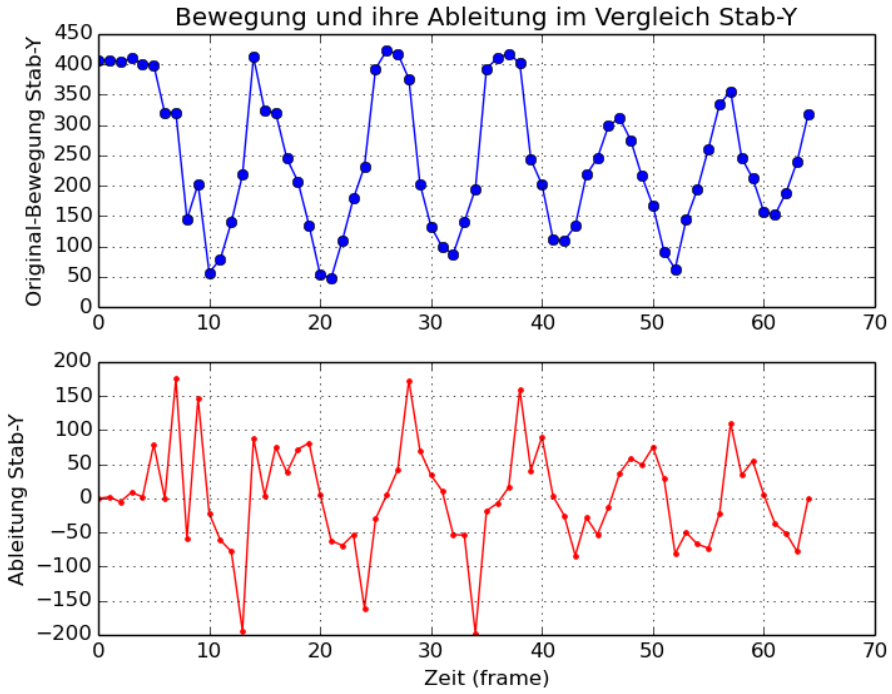


Abbildung 4.3.: Plot von Objektkoordinaten über die Zeit

gezogen. Das Ergebnis wird dann in eine Frequenz umgewandelt (von der Periodenlänge) und mal 60 genommen, um den gewünschten Wert zu bestimmen.

4.5.2. Taktart

Neben der Analyse der Dirigierfrequenz wollten wir auch in der Lage sein Taktarten erkennen zu können. Dies stellten wir uns so vor, dass ein mit dem Taktstab gezeichnetes Viereck den 4/4-Takt, ein rechtwinkliges Dreieck den 3/4-Takt und ein Strich den 2/4-Takt darstellen sollte. Eine grundlegende Idee war das Feststellen von Richtungsvektoren, welche die aktuell analysierte Bewegung des Dirigierstabes beschrieben. Hierbei war unser erster Ansatz, dass wir immer zwischen dem aktuellen und dem vorhergehenden Frame einen Vektor bildeten. Um die ganze Sache zu Vereinfachen führten wir Schwellenwerte ein, welche entschieden, ob ein Vektor als relevant betrachtet wird oder nicht. Kleinere Schwankungen wurden also folglich nicht mehr wahrgenommen. Aus jedem Vektor ermittelten wir einen weiteren Vektor, der lediglich die aktuelle Bewegungsrichtung beinhaltet. Anhand dieser Vektoren wollten wir zunächst die Taktarten erkennen. Dies funktionierte mehr oder weniger auch zufriedenstellend was senkrechte oder waagrechte Bewegungen betraf, Diagonalbewegungen bereiteten aber große Probleme. Lange Zeit testeten wir verschiedenste Wege um dieses Problem zu beseitigen. Zunächst versuchten wir unsere Schwellenwerte so zu modifizieren, dass bei der schwierigeren Diagonalbewe-

gung eine größere Toleranz galt, dies beseitigte aber nicht das Problem, dass es sehr schwierig war per Hand saubere Diagonalbewegungen zu dirigieren. Als nächstes überlegten wir, dass es Richtungsvektoren gibt, die das Programm erkennen kann, die aber in bestimmten Situationen keinen Sinn ergeben. Beispielsweise ergeben aufeinanderfolgende Diagonalbewegungen keinen Sinn, da wir uns darauf geeinigt hatten, dass wir nur rechtwinklige Dreiecke für die Taktartbestimmung zulassen. Dies war und ist weiterhin ein sehr guter Ansatz für die Verbesserung unseres Programms, jedoch erschien er uns nach einer Weile zu kompliziert und wir widmeten unsere Aufmerksamkeit einer anderen Idee. Wir entschieden auf dem Bild, welches dem Benutzer des Programms angezeigt wird, eine minimalistische graphische Benutzeroberfläche hinzuzufügen. Diese sollte vier gleich große Kreise beinhalten, die in einem Viereck angeordnet sind. Zusätzlich schrieben wir Funktionen, die prüften in welchem Kreis sich der Taktstab aktuell befindet und welcher Richtungsvektor aus welcher Kreisfolge resultiert. Dies stellte sich als simple und elegante Lösung unseres Problems heraus und mithilfe dieser Neuerung war es uns möglich auch die aktuelle Taktart zu bestimmen.

4.5.3. Lautstärke

Lange Zeit nutzten wir die Pixelentfernung zwischen den festgestellten Extremstellen als bestimmenden Parameter für die Lautstärkebestimmung. Dies funktionierte einwandfrei und war nicht besonders kompliziert. Problematisch wurde es erst, als wir uns entschieden die graphische Benutzeroberfläche in Form der vier Kreise zu verwenden. Hiermit beschränkten wir die Bewegungsfreiheit des Dirigenten, sodass es deutlich schwieriger war als Benutzer Einfluss auf die Lautstärke zu nehmen. Dies ist ein Punkt in dem unser Projekt noch Nacharbeit benötigen kann. Vorstellbare Lösungsansätze wären das Entwickeln einer weniger fehleranfälligen Vektorberechnung oder eine Veränderung der graphischen Benutzeroberfläche, sodass auch ausladendere Bewegungen möglich sind.

4.6. Musik

Nachdem wir erreicht hatten, die von uns als Ziel festgelegten Parameter (Geschwindigkeit, Lautstärke und Taktart) im Analyseteil festzustellen, galt unsere Aufmerksamkeit der Übertragung dieser Parameter auf das Abspielen von Musikstücken. Lange Zeit beschäftigten wir uns damit alle nötigen Python-Erweiterungen auf unseren Rechnern zu installieren. Dies war ein nicht zu vernachlässigender Zeitaufwand. Nachdem diese Aufgabe bewältigt war, war es mithilfe eines kleinen Skriptes von Stefan Born schnell möglich parallel zur Bildverarbeitung Musikdateien abzuspielen. Die hierfür benötigten, frei verfügbaren Midi-Dateien stehen zu großer Zahl im Internet zur Verfügung. Beschäftigen mussten wir uns lediglich mit der Umrechnung des festgestellten BPM-Wertes in die entsprechenden Miditicklänge und schon konnten wir live in unserem Programm die Geschwindigkeit mit den Dirigierbewegungen manipulieren.

5. Endprodukt

Um das fertige Programm möglichst verständlich zu erklären, haben wir uns dafür entschieden es in seine Bestandteile zu zerlegen. Es wird das Hauptprogramm erklärt und auf die Unterfunktionen verwiesen. Die genaue Funktionsweise der Unterfunktionen wird dann im folgenden erklärt.

5.1. Bibliotheken

Am Anfang des Programms steht das Einbinden der Bibliotheken. Von diesen wurde eine große Anzahl genutzt. Beispielhafte Anwendungsgebiete der Funktionen im Programm sind im folgenden aufgeführt.

- `from __future__ import division` ermöglicht dividieren, ohne das der Rest wegfällt. Ist dies gewünscht, so wird im folgenden `//` verwendet.
- `random` gibt Zugriff auf Zufallszahlen, welche zum Beispiel bei der Auswahl eines Musikstückes eine Rolle spielen.
- `time` bietet unter anderem die Systemzeit, welche bei der Bestimmung der Geschwindigkeit eine Rolle spielt.
- `numpy` liefert eine Vielzahl an Objekten, Operationen und Funktionen, welche sowohl bei mathematischen Berechnungen (Bsp. Wurzel) eine Rolle spielen, als auch bei der Arbeit mit Arrays.
- `cv2` importiert OpenCV in das Programm. Hierbei handelt es sich um die bei der Videoanbindung wohl wichtigste Bibliothek. Es kann sowohl die Kamera ausgelesen werden, als auch das Video ausgegeben werden. Hinzu kommen noch viele weitere nützliche Funktionen.
- `sys` wird genutzt, um auf den Interpreter zuzugreifen und findet beispielsweise bei der Beendigung des Programms Anwendung.
- `copy` ermöglicht ein besonderes Kopieren von beispielsweise Listen.
- `pickle` wird genutzt, um Werte außerhalb des Programms zu speichern und später weiterzuverwenden. Im vorliegenden Programm wird es genutzt um die Kalibrierung zu speichern.
- `midi` findet bei der Ausgabe der Musik Anwendung. Es werden Midi-Dateien verarbeitet.

- `fluidsynth` ist ein SoundFont-Synthesiszer, welcher im Programm Anwendung findet, um Musik abzuspielen.
- `threading` bietet die Möglichkeit mehrere Prozesse laufen zu lassen und wird bei der Musikverarbeitung verwendet.
- `os` wird verwendet um mit dem Betriebssystem zu kommunizieren und ermöglicht es das Dateisystem auszulesen, um beispielsweise ein Musikstück auszuwählen.

```
24 from __future__ import division
25 import time
26 import random
27 import numpy as np
28 import cv2
29 import sys
30 from copy import deepcopy
31 import pickle
32 import midi
33 import fluidsynth
34 import threading
35 import os
```

Code 5.1: Die Bibliotheken

5.2. Hauptprogramm

Das Hauptprogramm umfasst viele Variablendefinitionen und Funktionsaufrufe, ebenso wie eine sich immer wiederholende Schleife. Im folgenden (Code 5.2) ist das Hauptprogramm zur besseren Übersicht einmal komplett abgebildet. Danach wird auf die einzelnen Teile eingegangen. Dort finden sich dann auch die Verweise zu den Funktionen

```
470 #=======gestures=======#
471
472 rectangleCW = [(1,0),(0,-1),(-1,0),(0,1)]
473 rectangleCCW = [(1,0),(0,1),(-1,0),(0,-1)]
474 triangleA = [(1,1),(0,-1),(-1,0)]
475 triangleB = [(-1,-1),(1,0),(0,1)]
476 triangleC = [(1,-1),(-1,0),(0,1)]
477 triangleD = [(-1,1),(0,-1),(1,0)]
478 line = [(0,1),(0,-1)]
479 gestures = [rectangleCW, rectangleCCW, triangleA, triangleB, triangleC,
        triangleD, line]
480
481 #=======circles=======#
482
483 c1 = ((320+120),(240-120))
484 c3 = ((320-120),(240+120))
485 c2 = ((320-120),(240-120))
486 c4 = ((320+120),(240+120))
487 cmenu = (640,480)
488 color = (10,10,255)
```

```
489
490 #=======variables=======#
491
492 global fnr, bpm, new
493
494 fnr = 0
495 bpm = 0
496 cthresh = 95
497 ref = (320,240)
498 old = 0
499 new = 5
500
501 #=======lists=======#
502
503 veclist = [None]              #liste beinhaltet richtungsaenderungen
504 bpmlist = []
505
506 #=======program=======#
507
508 cap = cv2.VideoCapture(0)
509 schwellenwerte_einlesen()
510
511 while True:
512   _, frame = cap.read()
513   clean = np.copy(frame)
514
515   position = detect_color(frame)
516
517   get_visual(position)
518
519   new = in_circle(position)
520
521   if new == "menu":
522     menu_visual(cap)
523     continue
524
525   vec = get_vec(old, new)
526
527   if vec != None and vec != veclist[-1]:
528     veclist.append(vec)
529     detect_bpm(len(veclist))
530
531   if new != None and new != old:
532     old = new
533
534   bpm = bpm_mean(bpm)
535
536   frame = cv2.addWeighted(frame, 0.5, clean, 0.5, 0)
537   mirror = cv2.flip(frame,1)
538
539   check_gestures()
540
541   cv2.putText(mirror, str(int(bpm)), (560,40), 2, 1, (255,255,255), 0)
542
```

```
543   cv2.imshow("frame", mirror)
544   fnr += 1
545
546   if fnr == 200:
547     try:
548       p = Player()
549       p.load(waehle_stueck(counter(), "../Midi"))
550       p.play()
551     except:
552       print "Bitte neu starten und Dirigieren."
553   if fnr > 200:
554     try:
555       if bpm > 0:
556         p.tick_duration = tick_laenge_ermitteln(bpm)
557     except:
558       print "Fehler. Bitte neu starten."
559
560   if cv2.waitKey(1) & 0xFF == ord('k'):
561     get_values(cap)
562
563   if cv2.waitKey(1) & 0xFF == ord(' '):
564     if fnr > 200:
565       p.stopit()
566     print "herunterfahren"
567     break
568
569 cap.release()
570 cv2.destroyAllWindows()
```

Code 5.2: Hauptprogramm

Zu Beginn des Hauptprogramms werden einige Variablen angelegt und definiert (Code 5.2 Z.470–506). Der erste Abschnitt »gestures« definiert die unterschiedlichen Gesten, die der Dirigent mit seinem Stab zur Steuerung des Programms vollführen kann. Diese Gesten sind Listen von Richtungsvektoren, welche die Stabposition durchläuft. Im Anschluss werden diese Gesten in einer Liste zusammengefasst.

Im zweiten Abschnitt »circles« werden die Positionen der Kreise des Benutzerinterfaces festgelegt. `c1` bis `c4` stehen dabei für die vier Kreise, über welche die Dirigierbewegung erfasst wird und `cmenu` steht für den Kreis in der Ecke, über welchen das Menü eingeblendet werden kann. `color` enthält die Farbe der vier großen Kreise im RGB-Format.

Der dritte Abschnitt »variables« definiert nun einige weitere Variablen. Dabei sind `fnr`, `bpm` und `new` globale Variablen, auf welche auch in Funktionen zugegriffen werden kann, denen diese nicht übergeben wurden. Die folgenden Variablen werden definiert:

- `fnr` steht für »Framenummer« und enthält die Nummer des aktuellen Frames. Über diese Variable wird beispielsweise gesteuert, dass die Musik erst ab dem 200sten Frame beginnt. Der Wert wird hochgesetzt, wenn ein neuer Frame gezeigt wurde.

- `bpm` respräsentiert die Geschwindigkeit des Dirigats. In der Musik wird diese Einheit (Schläge pro Minute) standardmäßig verwendet, um die Schnelligkeit eines Stückes

zu bemessen. Sie stellt eine genauere Alternative zu den Namen für verschiedene Tempi (Adagio, Moderato, Presto,...) dar.

- `cthresh` enthält den Radius der Kreise des Benutzernterfaces.
- `ref` repräsentiert den Mittelpunkt des Bildes.
- `old` enthält die Nummer des letzten Kreises, in dem der Stab detektiert wurde.
- `new` enthält die aktuelle Nummer des Kreises, in dem der Stab detektiert wurde.

Im vierten Abschnitt »lists« werden zwei Listen erstmalig angelegt. `veclist` ist dabei die Liste, welche die detektierten Verschiebungsvektoren enthält und `bpmlist` die Liste, welche die berechneten Geschwindigkeiten sammelt, um Geschwindigkeitsschwankungen beim Dirigenten zu kompensieren. Diese Kompensation findet in der `bpm_mean`-Funktion statt.

In Zeile 506 geht es nun mit dem eigentlichen Programm los. Zunächst wird über die `cv2.VideoCapture`-Funktion ein Objekt erstellt, welches mit der angeschlossenen Webcam kommunizieren kann. Die Zahl, welche der Funktion übergeben wird steht für eine Kamera und ist ist in dem meisten fällen gleich Null. Sind mehrere Kameras angeschlossen, werden die unterschiedlichen Kameras ausgewählt, indem die Zahl geändert wird.

Als nächstes wird die `schwellenwerte_einlesen`-Funktion (Code 5.11) aufgerufen, welche die Schwellenwerte für die Farberkennung festsetzt. Näheres zur Funktionsweise ist im entsprechenden Abschnitt nachzulesen.

Nun Beginnt die Schleife, welche stetig durchlaufen wird und stets ein neues Bild holt, es auswertet, auf Befehle vom Nutzer wartet und sich wiederholt, bis das Programm beendet wird. In dieser Schleife geschieht auch die Kommunikation mit den meisten Funktionen, sowie Start und Interaktion mit der Musik.

Zu Beginn der Schleife wird ein neues Bild von der Kamera geholt. Hierfür wird auf das vor der Schleife angelegte `cap`-Objekt die `read`-Methode angewendet. Zurückgegeben wir ein Tupel. Der erste Teil wird nicht weiter benötigt und fällt weg. Der zweite Teil enthält den aktuellen `frame`. Dieser wir in einer gleichnamigen Variable gespeichert.

Im folgenden (Zeile 513) wird der Frame über eine Numpy-Funktion kopiert in die Variable `clean`. An dieser Stell ist darauf hinzuweisen, dass es sich bei `frame` um ein dreidimensionales Array handelt, welches auf zwei Dimensionen die einzelnen Pixel aufgetragen hat und auf die dritte Dimension die Farbe, welche sich aus drei Werten (RGB) zusammensetzt. An dieser Stelle wird nun eine Kopie des Arrays erstellt. Dabei entspricht `clean` keinem Verweis auf `frame`, sondern repräsentiert eine eigene Stelle im Speicher. Die Kopie wird angelegt, um für das Benutzerinterface transparente Kreise zu erstellen.

In Zeile 515 wird die aktuelle Position des Stabes ermittelt, welche von der `detect_color`-Funktion (Code 5.9) zurückgegeben wird.

`get_visual` wird im folgenden aufgerufen und bekommt die soeben ermittelte Position des Stabes übergeben. Sie ist dafür zuständig, das Benutzerinterface zu visualisieren. So zeigt sie die aktuelle Postion an, zeichnet die Kreise, färbt den aktuellen Kreis und ist für einige weitere Visualisierungen zuständig. Näheres ist zu finden bei Code 5.15.

In Zeile 519 wird über die `in_circle`-Funktion (Code 5.17) der Kreis ermittelt, in dem sich der Stab aktuell befindet. Das Ergebnis wird folglich in `new` vermerkt.

In den Zeilen 521 bis 523 wird das Menü in die Schleife integriert. Sollte sich der Stab Momentan im Menü-Kreis befinden (`new == "menu"`) wird die `menu_visual`-Funktion (Code 5.23) aufgerufen, welche das Menü anzeigt.

Im folgenden wird der aktuelle Vektor über die `get_vec`-Funkion (Code 5.14) bestimmt. Hierzu wird der letzte und der aktuelle Kreis zu Rate gezogen. Der aktuelle Vektor ist dann zu finden in `vec`.

In den Zeilen 527 und 528 wird nun der soeben ermittelte Vektor an die Liste der Vektoren angehängt, falls er überhaupt existiert und verschieden zum vorigen Eintrag in der Liste ist. Die Liste wird also nur erweitert, wenn sich der Vektor ändert. Ändert sich dieser ist somit auch gleichzeitig ein Schlag detektiert worden und die Funktion `detect_bpm` kann aufgerufen werden, um eine aktualisierte Geschwindigkeit zu bestimmen. Näheres zu deren Aufbau ist zu finden bei Code 5.13.

Hat sich die Kreisposition geändert und ist bereits eine Kreisposition detektiert worden wird nun aus der neuen Position auch die alte Position `old`.

In der Zeile 534 wird nun eine neue Geschwindigkeit ermittelt. »ermittelt« ist in diesem Zusammenhang tatsächlich der passende Begriff, da die aufgerufene `bpm_mean`-Funktion einen Mittelwert aus vergangenen Geschwindigkeiten bestimmt. Näheres zur entsprechenden Funktionsweise ist wieder zu finden beim entsprechenden Code 5.21.

Im folgenden wird das Bild zur Ausgabe aufbereitet. Die `cv2.addWeigted`-Funktion legt zwei Bild-Arrays übereinander und kann so Transparenz-Effekte erzeugen. Im vorliegenden Fall wird das zu Beginn gesicherte Bild `clean` überlagert mit dem Bild `frame`. Auf diese Art und Weise werden die Transparenzeffekte der Kreise im Benutzerinterface erzeugt.

Damit der Dirigent bei der Betrachtung des Bildes nicht »spiegelverkehrt« denken muss wird nun das Bild über die `cv2.flip`-Funktion an der horizontalen Achse (Achse ist zweites Argument der Funktion) gespiegelt. Es ist nun für den Nutzer so, als würde er sich in einem Spiegel betrachten, wenn er die Ausgabe betrachtet. Das Ergebnis wird in `mirror` gespeichert. Dieses Bild wird im folgenden noch mit weiteren Informationen versehen und dann ausgegeben.

Solch eine Veränderung wird beispielsweise von der `check_gestures`-Funktion hervorgerufen (Code 5.19). In dieser wird über die `detect_gesture`-Funktion die aktuelle Bewegung erkannt und über die `count`-Funktion eine Ausgabe auf dem Bild erzeugt, welche über Art der Bewegung und Position der Bewegung informiert. Eine weitere Veränderung des `mirror`-Arrays findet in Zeile 541 statt. An dieser Stelle wird der Ausgabe die momentane Geschwindigkeit hinzugefügt.

Die Ausgabe ist nun vorbereitet worden und wird über die `cv2.imshow`-Funktion vollzogen. Dabei ist das erste Argument der Fenstername und das zweite Argument das zu visualisierende Array. Nachdem dies geschehen ist wird die Framenummer um eins erhöht.

Der folgende Block (Z.546–558) ist für den Start und die Steuerung der Musik zuständig. Zum Zeitpunkt des 200sten Frame (`if nr == 200`) wird versucht (`try`) die Musik zu starten. Dies geschieht, indem zuerst ein Player-Objekt (Code 5.3) angelegt wird.

Im folgenden wird eine Midi-Datei über die `load`-Methode (Code 5.4) geladen. Diese Methode bekommt an sich nur den Pfad zur entsprechenden Datei übergeben. Im vorliegenden Fall ist die Dateiauswahl jedoch nicht immer gleich. So wird der Dateipfad von der `waehle_stueck`-Funktion (Code 5.24) an die `load`-Methode weitergeben. `wahle_stueck` an sich bekommt über die `counter`-Funktion die Taktart und als zweites Argument den Ordner mit dem Midi-Verzeichnis übergeben. In diesem Verzeichnis sind Ordner zu bestimmten Taktarten abgelegt, welche die entsprechenden Dateien enthalten.

Sobald die Datei geladen ist wird sie über die `play`-Methode abgespielt. Sollte das Laden oder Abspielen fehlschlagen, weil beispielsweise der Dirigent noch nicht begonnen hat zu dirigieren und keine Taktart erkannt werden konnte läuft das Programm weiter (`except`) und es wird ein Hinweis im Terminal ausgegeben.

Nachdem die Musik gestartet wurde (`fnr > 200`) wird im folgenden versucht (`try`) die Geschwindigkeit der Musik zu verändern. Dies geschieht, indem das Player-Attribut `tick_duration` verändert wird. Diese Veränderung wird jedoch nur vorgenommen, wenn die Geschwindigkeit (repräsentiert als `bpm`) größer als Null ist, da im folgenden durch diesen Wert geteilt wird und ein Teilen durch Null zu einem Fehler führt. Sind die Voraussetzungen jedoch gegeben wird die neue Ticklänge über die `tick_laenge_ermitteln`-Funktion bestimmt (Code 5.22) und im Objekt verändert. Die Musik ändert so ihre Geschwindigkeit, dar der Tick die elementare Zeitlänge beim abspielen von Midi-Datien ist, von welchem Abhängt, wie weit die einzelnen Midi-Events voneinander entfernt sind. Solche Midi-Events geben beispielsweise an, dass ein bestimmter Ton begonnen oder beendet wird.

Sollte eine Anpassung der Geschwindigkeit zu einem Fehler führen, wird auch hier der Benutzer wieder darauf hingewiesen.

Die folgenden Funktionen innerhalb der Schleife sind nun noch für die Steuerung des Programms gedacht. Die Tastenbelegung kann aus dem Menü entnommen werden. Die `cv2.waitKey`-Funktion wartet, ob ein Tastenereignis eintritt. Sie bekommt dabei eine Länge in Millisekunden übergeben, welche gewartet wird. In unserem Fall ist dies eine Millisekunde. Die Funktion gibt die Nummer des Buchstaben zurück. Um diese Nummer mit dem ASCII-Code aus der `ord`-Funktion zu vergleichen werden allerdings nur die letzten acht Stellen benötigt. Also wird ein bitweises `AND` mit `0xFF = 0b11111111` durchgeführt und mit der ASCII-Nummer des Buchstabens verglichen. Kommt dieser Vergleich zu einem wahren Ergebnis werden die Anweisungen nach der Bedingung ausgeführt.

Im ersten Abschnitt wird die Funktion `get_values` ausgeführt, sobald ein »k« gedrückt wird. Diese Funktion (Code 5.10) ermöglicht das wählen neuer Farbeinstellungen für die Staberkennung.

Der zweite Block beendet das Programm. Wird die Leertaste gedrückt wird über das `break` die Schleife verlassen und das Programm beendet sich, nach dem über `cap.release` der Stream zur Kamera geschlossen wird und über `cv2.destroyAllWindows` alle Fenster geschlossen werden. Bevor dies geschieht wird jedoch die Musik beendet, sollte sie bereits gestartet worden sein (`fnr > 200`). Dies geschieht über die `stopit`-Methode des Player-Objektes. Beim Herunterfahren wird der Benutzer auch über eine Ausgabe informiert.

Im folgenden finden sich nun nähere Erklärungen zu den verwendeten Funktionen und Methoden.

5.3. Funktionen

5.3.1. Player

Bei dem Player-Objekt handelt es sich um eines, welches von Stefan geschrieben wurde. Es wurde erstellt, um Midi-Dateien abspielen zu können. Die besondere Herausforderung dabei war dies so zu gestalten, das während des Abspielens die Geschwindigkeit der Musik veränderbar ist. Diese Eigenschaft wird über das `tick_duration`-Attribut (Zeile 556) gesteuert. Es gibt an, wie lange eine Grundeinheit der Midi-Operationen dauert. Dieses Attribut kann man auch ändern, während Musik abgespielt wird und somit ist eine Änderung der Musikgeschwindigkeit möglich.

init-Methode

Die `init`-Methode initialisiert das Player-Objekt. Bei `fluidsynth` handelt es sich um einen Synthesizer. Dieser arbeitet mit sogenannten Soundfonts. Ein solches, welches hier verwendet wird ist `FluidR3_GM.sf2`. In dieser Methode wird es geladen. Der entsprechenden Funktion wird dabei der Pfad zum Soundfont übergeben. Dementsprechend befindet es sich im vorliegenden Fall im gleichen Verzeichnis. Der dritte Punkt auf den hingewiesen werden sollte ist der Treiber. Dieser wird in Zeile 44 gesetzt. Im vorliegenden Programm ist `alsa` gewählt. Sollte das Programm unter Windows laufen sollte an dieser Stelle ein anderer Treiber, wie beispielsweise `dsound` gewählt werden. Synthesizer, Soundfont und Treiber spielen bei der Installation eine wichtige Rolle. Näheres ist dazu im Anhang unter Installationshinweise zu finden.

```
41 def __init__(self):
42   self.d=[]
43   self.fs = fluidsynth.Synth()
44   self.fs.start(driver="alsa")
45
46   self.sfid = self.fs.sfload("FluidR3_GM.sf2")
47   self.fs.program_select(0, self.sfid, 0, 0)
48
49   self.tick_duration = 0.01
```

Code 5.3: init-Funktion

load-Methode

Die `load`-Methode lädt eine Midi-Datei zum abspielen. Übergeben wird der Methode der Pfad zur Datei, sowie in der Anwendung der Methode auf ein Objekt, das Objekt selbst.

```
52 def load(self, filename):
53   '''laedt ein Midi-File mit Namen filename.'''
54   with open(filename,'rb') as f:
55     self.d=midi.fileio.read_midifile(f)
56   self.d.make_ticks_abs()
```

Code 5.4: load-Funktion

play-Methode

Die `play`-Methode spielt die geladene Midi-Datei ab. Sie öffnet mehrere Threads, welche dann parallel zum Hauptprogramm laufen. In diesen Threads spielt die `play_voice`-Funktion die entscheidende Rolle, indem sie jeweils eine Stimme analysiert.

```
58 def play(self):
59   '''spielt die Datei in Echtzeit ab, die zugrundeliegende
60   Zeiteinheit self.tick_duration laesst sich waehrend des Abspielens
61   aendern.'''
62   self.stop = False
63   for voice in self.d:
64     voice_thread = threading.Thread(target=self.play_voice, args=(voice,))
65     voice_thread.start()
```

Code 5.5: play-Funktion

stopit-Methode

Die `stopit`-Methode beendet das Abspielen der Musik. In der praktischen Anwendung kommt es hier allerdings immer wieder zu Problemen. Der letzte Ton wird oft ausgehalten, bis das Terminal geschlossen ist.

```
67 def stopit(self):
68   self.stop = True
```

Code 5.6: stopit-Funktion

play_voice-Methode

Die `play_voice`-Methode ist für die Analyse einzelner Stimmen zuständig und wird von der `play`-Methode aufgerufen. Von dieser Methode aus wird über die `send`-Methode mit dem Synthesizer kommuniziert.

```
70 def play_voice(self,voice):
71   voice.sort(key=lambda e: -e.tick)
72
73   tick_now = 0
74
75   time_now = time.time()
76
77   while len(voice)>0 and not self.stop :
78     event=voice.pop()
79     while event.tick>tick_now:
80       #print event
81       n_ticks= max(int((time.time()-time_now)//self.tick_duration)+1,event.
     tick-tick_now)
82       #print n_ticks, n_ticks*self.tick_duration-time.time()+time_now
83       time.sleep(n_ticks*self.tick_duration-time.time()+time_now)
84       tick_now +=n_ticks
85       time_now=time.time()
86
```

```
87      self.send(event)
```

Code 5.7: play_voice-Funktion

send-Methode

Die `send`-Methode ermöglicht die Kommunikation mit dem Synthesizer.

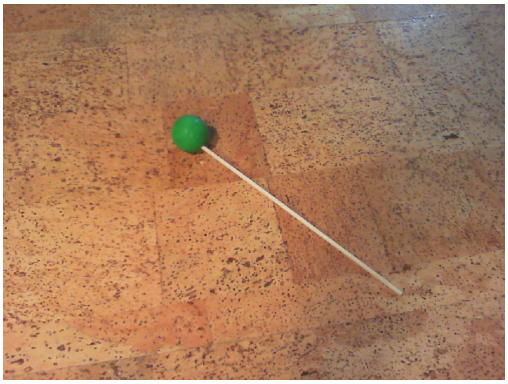
```
89 def send(self, event):
90   '''Diese Methode kommuniziert mit dem Synthesizer. Bisher sind
91   nur drei Ereignistypen implementiert.'''
92   if type(event) is midi.NoteOnEvent:
93     self.fs.noteon(event.channel,event.data[0],event.data[1])
94   elif type(event) is midi.NoteOffEvent:
95     self.fs.noteoff(event.channel, event.data[0])
96   elif type(event) is midi.ProgramChangeEvent:
97     self.fs.program_change(event.channel, event.data[0])
98   elif type(event) is midi.ControlChangeEvent:
99     self.fs.cc(event.channel, event.data[0],event.data[1])
```

Code 5.8: send-Funktion

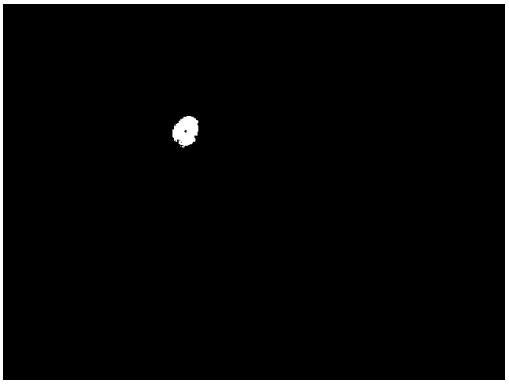
5.3.2. detect_color-Funktion

Der Funktion wird das Array des aktuellen Frames übergeben. Hieraus ermittelt sie mithilfe durch andere Funktionen festgelegter Farbwerte die Position des zu verfolgenden Objektes. Hierbei spielt die Funktion `cv2.inRange` eine wesentliche Rolle. Sie hilft uns eine Maske zu erstellen, die allen Werten der Quelle, welche innerhalb festgelegter Werte liegen, ein `True` zuordnet und allen Werten, die außerhalb liegen ein `False`. Hierbei ist es weitaus einfacher das zu analysierende Bild nicht im BGR-Farbraum zu betrachten, sondern im HSV-Farbraum. Dieser besteht ebenfalls aus drei Werten, dem Hue, der Saturation und dem Value. Hiermit ist es uns einfacher möglich, die `inRange`-Funktion zu verwenden. Die verwendeten Werte müssen regelmäßig an die Umgebungsbedingungen angepasst werden, da das Programm sonst nicht korrekt funktioniert. Die `bitwise_and`-Funktion ermöglicht es uns dann letztlich aus dem Originalbild nur die Pixelwerte anzeigen zu lassen, die auf der gleichen Position, wie ein `True` der Maske liegen. Um die Position des verfolgten Objektes in einer Koordinate zu bestimmen nutzen wir noch `numpy.mean`, da die Koordinate annähernd immer dem Mittelwert des Arrays der `True`-Postitionen entspricht. Hierbei spielt die Genauigkeit der zu Beginn gewählten Range-Werte eine nicht zu vernachlässigende Rolle. Nach allen Berechnungen gibt die Funktion die ermittelte Koordinate in Form eines Tupels zurück.

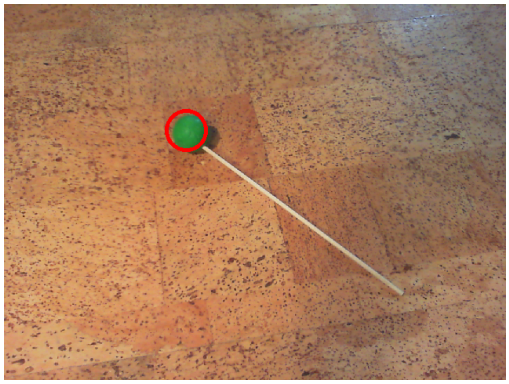
```
103 def detect_color(frame): #Funktion, welche eine nach Farbe definierte
        Stelle ermittelt
104
105   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)   #konvertiert frame in hsv
106   lower = np.array([hue_min, sat_min, val_min])  #legt untere schranke fest
107   upper = np.array([hue_max, sat_max, val_max])  #legt obere schranke fest
108
```



(a) Ursprüngliches Bild



(b) Maske nach Farbeingrenzung



(c) Erkannte Stabposition

Abbildung 5.1.: Farberkennung

```
109  #Die Wert aus den Schranken werden aus globalen Variablen gezogen
110
111  mask =  cv2.inRange(hsv, lower, upper)                    #erstellt maske,
       welche fuer werte innerhalb des intervalls
112                                    #den wert 1 annimmt, sonst 0
113  y_werte, x_werte = np.where(mask == 255)                  #Es werden die x-
       und y-Werte ausgelesen, welche ein True (255) bekomen haben
114  if len(x_werte) > 20:                                  ###es reicht eine bedingung
       zu erfuellen + schwellenwert
115    y_mittel = int(np.mean(y_werte))                        #Es wird der Mittelwert
        aus allen y-Werten gebildet
116    x_mittel = int(np.mean(x_werte))                        #Es wird der Mittelwert
        aus allen x-Werten gebildet
117    position = (x_mittel, y_mittel)                         #Die Mittlere Position
       aller Trues entspricht dem Tupel beider Mittelwerte
118
119  else:                                          ###if-bedingung unnoetig, mittelpunkt
       erhaelt man durch tausch von x und y
120    y_shape, x_shape, _ = frame.shape                       #Es werden die
       Bildmasse ausgelesen
```

```
121     position = (int(x_shape//2),int(y_shape//2))          #Als Position
        wird der Bildmittelpunkt gewaehlt
122
123   return position                                  #Ergebnis wird zurueckgegeben
```

Code 5.9: detect_color-Funktion

5.3.3. get_values-Funktion

Diese Funktion ist für die Kalibrierung der Farberkennung zuständig. Die Schleifen von Zeile 127 bis Zeile 137 gibt dem Nutzer Zeit den Stab in die Richtige Position zu bringen. Die Position wird dabei durch einen Kreis gekennzeichnet (Z. 129), der Mittig im Bild angeordnet ist (`(x_shape//2, y_shape//2)`). Wird bestätigt das der Stab in der entsprechenden Position ist (`cv2.waitKey(1) & 0xFF == ord('b')`) wird die Schleife verlassen. Es wird ein neuer Frame ohne Kreis geholt und analysiert. Die einzelnen Analyseschritte sind wohl am besten den Kommentaren des Codes zu entnehmen. Zur erwähnen ist noch, dass die ermittelten Werte in einer externen Datei abgespeichert werden, um beim nächsten Programmstart wiederverwendet werden zu können.

```
125 def get_values(cap):
126   while(True):
127     _, frame = cap.read()
128     y_shape, x_shape, _ = frame.shape
129     cv2.circle(frame,(x_shape//2, y_shape//2),25,(0,0,255),4)
130     frame = cv2.flip(frame,1) #Spiegelung des frames an der Horizontalen
131     cv2.putText(frame, "Bereit? [B]", (200,450), 2, 1, (255,255,255), 0)
132     cv2.imshow('frame', frame)
133     if cv2.waitKey(1) & 0xFF == ord('b'): #Signal, dass in Position, ueber
        das Druecken von k
134       break
135     if cv2.waitKey(1) & 0xFF == ord(' '):
136       cap.release()
137       sys.exit()
138
139   ret, frame = cap.read() #frame ohne Kreis wird geholt
140   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Konvertierung in hsv
141   radius = 25 #Radius des zu untersuchenden Kreises
142   kreisliste = [] #diese Liste soll alle Pixel enthalten, die innerhalb des
        Kreises liegen
143   for i in range(x_shape): #das Bild wird pixelweise durchgegangen. Wenn
        der Pixel im Bild liegt, wir dieser registriert
144     for j in range(y_shape):
145       if np.sqrt((x_shape//2-i)**2+(y_shape//2-j)**2) < radius: #Hier Satz
        des Pythagoras zur Entfernungsbestimmung Pixel--Kreismittelpunkt
146         kreisliste.append(hsv[j][i])
147   npkreisliste = np.array(kreisliste) #Konvertierung in Array zur weiteren
        Verarbeitung
148
149   hue = npkreisliste[:,0] #Array mit allen hue-Werten wird angelegt
150   sat = npkreisliste[:,1] #Array mit allen sat-Werten wird angelegt
151   val = npkreisliste[:,2] #Array mit allen val-Werten wird angelegt
152
```

```
153     #Berechung der Grenzen erfolgt, wie folgt. Zuerst wird der Mittelwert
        aller Werte einer Eigenschaft gebildet.
154     #Als naechstes wird jeweils die Standartabweichung ermittelt.
155     #Die untere Schranke ist dann einfach der Mittelwert - die
        Standartabweichung
156     #Die obere Schranke ist einfach der Mittelwert + die Standartabweichung
        .
157
158     #Die Ergebnisse werden zur direkten Weiterverwendung in globale
        Variablen geschrieben.
159     #Zur verwendung nach einem Programmneustart werden sie in eine datei
        ausgelagert.
160
161   global hue_min #globale Variable wird angelegt
162   hue_min = int(np.mean(hue) - np.std(hue)) #Wert wird ermittelt
163   global hue_max
164   hue_max = int(np.mean(hue) + np.std(hue))
165   global sat_min
166   sat_min = int(np.mean(sat) - np.std(sat))
167   global sat_max
168   sat_max = int(np.mean(sat) + np.std(sat))
169   global val_min
170   val_min = int(np.mean(val) - np.std(val))
171   global val_max
172   val_max = int(np.mean(val) + np.std(val))
173
174   sicherung = (hue_min,hue_max,sat_min,sat_max,val_min,val_max) #erzeuge
        Datensatztupel zur Abspeicherung fuer Pickle
175   output = open('hsv_werte.pkl', 'w') #die Ausgabedatei wird vorbereitet
176   pickle.dump(sicherung, output) #die Daten werden geschrieben
177   output.close() #der Output wird geschlossen
```

Code 5.10: get_values-Funktion

5.3.4. schwellenwerte_einlesen-Funktion

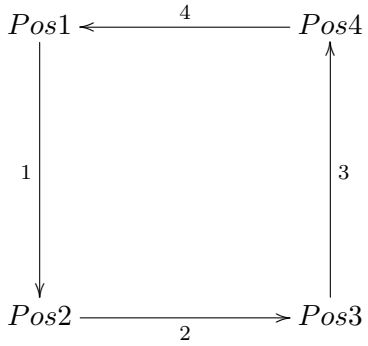
Diese Funktion wird beim Programmstart ausgeführt und legt erstmalig die Schwellenwerte für die Farberkennung fest. Dies sind globale Variablen. Es wird getestet, ob bereits eine Datei im Verzeichnis liegt, in der die Werte zwischengespeichert wurden. Ist dies der Fall werden die entsprechenden Wert in die Variablen geladen. Ist dies nicht der Fall werden beliebige Werte (1 und 0) den Variablen zugeordnet. Dies geschieht, damit das Programm erst einmal normal weiterlaufen kann und der Nutzer dann merkt, dass eine Kalibrierung einzuleiten ist.

```
179 def schwellenwerte_einlesen():
180   global hue_min #die Schwellenwerte werden als global definiert
181   global hue_max
182   global sat_min
183   global sat_max
184   global val_min
185   global val_max
186
```

```
187   try: #es wird versucht / festgestellt, ob es bereits eine Datei mit
        gespeicherten Schwellenwerten gibt
188     f = open("hsv_werte.pkl") #falls ja wird diese geoeffnet
189     sicherung = pickle.load(f)
190     hue_min, hue_max, sat_min, sat_max, val_min, val_max = sicherung #und
        die Daten entpackt
191
192   except: #falls nein, werden sehr komische Werte festgelegt, damit der
        Benutzter das Programm kalibriert
193     val_max = 1
194     val_min = 0
195     sat_max = 1
196     sat_min = 0
197     hue_max = 1
198     hue_min = 0
```

Code 5.11: schwellenwerte_einlesen-Funktion

5.3.5. detect_gesture-Funktion



Start	vorvor-letzter Vektor	vorletzter Vektor	letzter Vektor	aktueller Vektor
Pos1	1	2	3	4
Pos2	2	3	4	1
Pos3	3	4	1	2
Pos4	4	1	2	3

Abbildung 5.2.: Vektorumordnung

Die `detect_gesture`-Funktion wird von der `check_gestures`-Funktion aufgerufen, und soll testen, ob eine bestimmte Figur gerade ausgeführt wird. Zu diesem Zweck bekommt sie die Figur in Form einer Liste von Vektoren übergeben, sowie die zu überprüfende Liste an Vektoren. Zu Beginn werden aus der zu überprüfenden Liste die letzten entscheidenden Vektoren herausgelesen und in einer neuen Liste gespeichert. Die Anzahl der übernommenen Vektoren hängt dabei von der Figurlänge, also der Anzahl der übergebenen Listeneinträge der Figur ab.

Die beiden gleichlangen Listen die jetzt vorliegen (`figur`, `zu_untersuchen_kurz`) können nun jedoch nicht gleich verglichen werden, da es verschiedene Kombinationen der Figurvektoren geben kann. Die Startposition der zu untersuchenden Liste steht nicht fest. Also wird im folgenden (Z.208–215) ein Array erzeugt, bei dem die Zeilen für die verschiedenen Kombinationen stehen. Abbildung 5.2 verdeutlicht das Prinzip. Ist ein solches Array erstellt, werden nun die jeweiligen Zeilen mit der zu untersuchenden gekürzten Liste verglichen (Z.217–219). Kommt der Vergleich zu einem positiven Ergebnis wurde

die Figur detektiert und es wird ein `True` zurückgegeben. Kommt der Vergleich in keinem Fall zu einem positiven Ergebnis wird ein `False` zurückgegeben und so wurde gezeigt, das die Figur nicht die letzte Periode über ausgeführt wurde.

```
200 def detect_gesture(zu_untersuchen, figur):
201   figurlaenge = len(figur)
202
203   zu_untersuchen_kurz = [] #Diese Liste soll die letzten dirigierten
        Vektoren enthalten, die wichtig sind
204   for i in range(figurlaenge,0,-1): #von index -1 bis index 0
205     zu_untersuchen_kurz.append(zu_untersuchen[-i]) #letzten Eintraege
        werden ermittelt
206   zu_untersuchen_kurz = np.array(zu_untersuchen_kurz) #zur weiteren
        Verarbeitung Konvertierung in Array
207
208   figurensammlung = [] #verschiedene Kombinationen werden angelegt,
        schwierig sich das ohne zeichnung vorzustellen. Bsp.: aus [1,2,3] wird
        [[1,2,3],[2,3,1],[3,2,1]]
209   tmp = figur
210   for i in range(figurlaenge):
211     tmp.append(tmp[0])
212     del(tmp[0])
213     anhaengen = deepcopy(tmp)
214     figurensammlung.append(anhaengen)
215   figurensammlung = np.array(figurensammlung) #zur weiteren Verarbeitung
        Konvertierung in Array
216
217   for i in range(figurlaenge): #Der Vergleich findet statt
218     if np.array_equiv(figurensammlung[i],zu_untersuchen_kurz): #
        elementweises Vergleichen von letztem Dirigat und verschiedenen Figur-
        Kombinationen
219       return True #falls Figur gefunden
220
221   return False #falls Figur nicht gefunden
```

Code 5.12: detect_gesture-Funktion

5.3.6. detect_bpm-Funktion

Der Funktion wird die Länge der Liste, welche die Abfolge der Richtungsvektoren speichert, übergeben. In dieser Liste stehen nie zwei gleiche Richtungsvektoren hintereinander, sondern es wird immer nur dann ein neuer hinzugefügt, wenn er ungleich dem vorgehenden ist. In der Funktion werden zu Beginn drei Variablen als global Definiert: `a`, `b` und `bpm`. Wenn die Vektorenliste ihren ersten Richtungsvektor erhält, hat sie die Länge 2, da das erste Element standardmäßig bei uns ein None ist. Der Variable `a` wird nun die aktuelle Systemzeit übergeben (bei Windows und Linux unterschiedliche Werte, das mindert aber nicht die Funktionalität). Bei der nächsten festgestellten Richtungsänderung wird der Vektorenliste ein neuer Eintrag angehängt und die Länge beträgt nun 3. Folglich wird in `b` die abermals aktuelle Systemzeit gespeichert. Die BPM werden nun berechnet, indem 60 durch die Differenz von `b` und `a` geteilt wird. Da die Variable `bpm`

global ist, kann sie von anderen Programmteilen problemlos verwendet werden. Anschließend bekommt `a` den Wert von `b`, damit die Berechnungen bei dem nächsten Aufruf der Funktion wieder korrekt sind.

```
223 def detect_bpm(l):
224   global a
225   global b
226   global bpm
227   if l > 2:
228     b = time.clock()
229     bpm = 60/(b-a)
230     a = b
231   else:
232     a = time.clock()
```

Code 5.13: detect_bpm-Funktion

5.3.7. get_vec-Funktion

Übergeben bekommt diese Funktion sowohl ob sich der Dirigierstab aktuell in einem Kreis befindet, wenn ja in welchem, und in welchem Kreis er sich als letztes befunden hat. Für jede Möglichkeit wurde der entsprechende Richtungsvektor festgelegt, der zurückgegeben wird.

```
234 def get_vec(old, new):
235   if new == None:
236     return None
237   elif old == new:
238     return None
239   elif old == 1:
240     if new == 2:
241       return (1,0)
242     elif new == 3:
243       return (1,-1)
244     elif new == 4:
245       return (0,-1)
246   elif old == 2:
247     if new == 1:
248       return (-1,0)
249     elif new == 3:
250       return  (0,-1)
251     elif new == 4:
252       return (-1,-1)
253   elif old == 3:
254     if new == 1:
255       return (-1,1)
256     elif new == 2:
257       return (0,1)
258     elif new == 4:
259       return (-1,0)
260   elif old == 4:
261     if new == 1:
262       return (0,1)
```

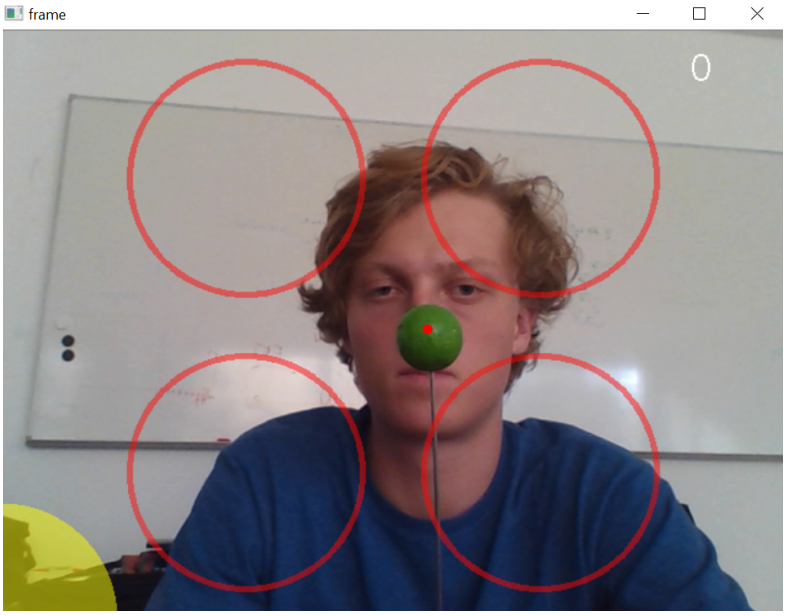


Abbildung 5.3.: Benutzerinterface

```
263     elif new == 2:
264       return (1,1)
265     elif new == 3:
266       return (1,0)
```

Code 5.14: get_vec-Funktion

5.3.8. get_visual-Funktion

Die Visualisierungsfunktion beinhaltet die meisten cv2-Befehle, welche beispielsweise Kreise auf dem Bildschirm zeichnen. Davon gibt es neben den großen Dirigierkreisen und dem Menükreis beispielsweise noch den kleinen ausgefüllten Kreis, der die aktuelle Position des Dirigierstabes darstellt.

```
268 def get_visual(pos):
269   if new == 1:
270     cv2.circle(frame, c1, cthresh, color, 3)
271     cv2.circle(clean, c1, cthresh, color, 3)
272   if new == 2:
273     cv2.circle(frame, c2, cthresh, color, 3)
274     cv2.circle(clean, c2, cthresh, color, 3)
275   if new == 3:
276     cv2.circle(frame, c3, cthresh, color, 3)
277     cv2.circle(clean, c3, cthresh, color, 3)
278   if new == 4:
279     cv2.circle(frame, c4, cthresh, color, 3)
280     cv2.circle(clean, c4, cthresh, color, 3)
281
```

```
282   cv2.circle(frame, c1, cthresh, color, 3)
283   cv2.circle(frame, c2, cthresh, color, 3)
284   cv2.circle(frame, c3, cthresh, color, 3)
285   cv2.circle(frame, c4, cthresh, color, 3)
286   cv2.circle(frame, pos, 4, (0,0,255), -1)
287   cv2.circle(clean, pos, 4, (0,0,255), -1)
288   cv2.circle(frame, (640,480), cthresh, (0,255,255), -1)
```

Code 5.15: get_visual-Funktion

5.3.9. get_length-Funktion

Der Funktion werden zwei Positionen in Form von Tupeln übergeben und Sie berechnet mithilfe eines Algorithmus basierend auf dem Satz des Pythagoras die skalare Länge des Verbindungsvektors der beiden Koordinaten.

```
290 def get_length(a,b):       #berechnet laege eines vektors von 2 punkten
291   ax, ay = a
292   bx, by = b
293   cx = ax - bx
294   cy = ay - by
295   l = np.sqrt(cx**2+cy**2)
296   return l
```

Code 5.16: get_length-Funktion

5.3.10. in_circle-Funktion

Übergeben bekommt diese Funktion eine Position bestehend aus einem Tupel. Sie prüft, ob die Position innerhalb eines relevanten Kreises liegt oder nicht. Dafür werden zunächst mithilfe von `get_length` die Entfernungen der zu Beginn des Programms hinterlegten Kreismittelpunkte zu der übergebenen Position berechnet. Anschließend wird geprüft, ob eine Entfernung kleiner als der ebenfalls hinterlegte Kreisradius ist, der für alle Kreise der selbe ist. Ist dies der Fall wird die Nummer des Kreises zurückgegeben, andernfalls `None`.

```
298 def in_circle(pos):
299
300   vec1 = get_length(c1, pos)    #sammlung der laengen der vektoren von den
        mittelpunkten zur position
301   vec2 = get_length(c2, pos)
302   vec3 = get_length(c3, pos)
303   vec4 = get_length(c4, pos)
304   vecmenu = get_length(cmenu, pos)
305
306   lengths = [cthresh+1, vec1, vec2, vec3, vec4, vecmenu]
307
308   for x in lengths:
309     if x < cthresh:
310       ret = lengths.index(x)
311       if ret == 5:
```

```
312             return "menu"
313         return ret
```

Code 5.17: in_circle-Funktion

5.3.11. beat_number-Funktion

Aufgabe dieser Funktion ist es im Falle einer erkannten Geste die aktuelle Taktposition zurückzugeben. Sie bekommt den aktuellen Richtungsvektor und die erkannte Geste übergeben. Für jede Geste ist hier bei dem entsprechenden Richtungsvektor eine beliebig gewählte Taktposition hinterlegt. Dennoch funktioniert sie einwandfrei.

```
315 def beat_number(vec, figure):
316   a,b = vec
317 
318   if figure == 'triangleA':
319     if vec == (0,-1):
320       return 1
321     if vec == (-1,0):
322       return 2
323     else:
324       return 3
325   if figure == 'triangleB':
326     if vec == (1,0):
327       return 1
328     if vec == (0,1):
329       return 2
330     else:
331       return 3
332   if figure == 'triangleC':
333     if vec == (-1,0):
334       return 1
335     if vec == (0,1):
336       return 2
337     else:
338       return 3
339   if figure == 'triangleD':
340     if vec == (0,-1):
341       return 1
342     if vec == (1,0):
343       return 2
344     else:
345       return 3
346 
347   if figure == 'rectangleCCW':
348     if vec == (0,-1):
349       return 1
350     if vec == (1,0):
351       return 2
352     if vec == (0,1):
353       return 3
354     if vec == (-1,0):
355       return 4
```

```
356   if figure == 'rectangleCW':
357     if vec == (0,-1):
358       return 4
359     if vec == (1,0):
360       return 3
361     if vec == (0,1):
362       return 2
363     if vec == (-1,0):
364       return 1
365
366   if figure == 'line':
367     if vec == (0,-1):
368       return 1
369     else:
370       return 2
371   else:
372     return False
```

Code 5.18: beat_number-Funktion

5.3.12. check_gestures-Funktion

Diese Funktion ist dafür zuständig nacheinander zu prüfen, ob eine hinterlegte Geste entdeckt wird oder nicht. Dabei ruft sie, sobald mehr als vier Richtungsvektoren zur Verfügung stehen, für jede hinterlegte Geste `detect_gesture` auf und anschließend zur Visualisierung `counter`. Für elegantere Lösungen mit `for`-Schleifen war am Ende keine Zeit übrig.

```
374 def check_gestures():
375   global figure
376   if len(veclist) > 4:
377     if detect_gesture(veclist, rectangleCW):
378         figure = "rectangleCW"
379         counter()
380     if detect_gesture(veclist, rectangleCCW):
381         figure = "rectangleCCW"
382         counter()
383     if detect_gesture(veclist, triangleA):
384         figure = "triangleA"
385         counter()
386     if detect_gesture(veclist, triangleB):
387         figure = "triangleB"
388         counter()
389     if detect_gesture(veclist, triangleC):
390         figure = "triangleC"
391         counter()
392     if detect_gesture(veclist, triangleD):
393         figure = "triangleD"
394         counter()
395     if detect_gesture(veclist, line):
396         figure = "line"
397         counter()
```

Code 5.19: check_gestures-Funktion

5.3.13. counter-Funktion

Diese Funktion zeigt oben links im Bild die aktuelle Taktposition an. Hier wurde empirisch ein Verfahren ermittelt, welches die etwas undurchschaubaren Strukturen der Figurenliste umgeht. Dies ist definitiv keine schöne Art zu programmieren, aber in unserem Fall hat es die Improvisationslösung in das endgültige Programm geschafft.

```
399 def counter():
400   f = len(figure)
401   if f > 10:
402     r = 4
403   else:
404     r = int(np.sqrt(f))
405 
406   cv2.putText(mirror, str(beat_number(veclist[-1],figure)) + "/" + str(r),
      (40,40), 2, 1, (255,255,255), 0)
407   return r
```

Code 5.20: counter-Funktion

5.3.14. bpm_mean-Funktion

Der Funktion wird die aktuelle Geschwindigkeit übergeben. Ist diese größer als 40, wird sie als relevant betrachtet und der Liste aller relevanten BPM-Werte angehängt. Hat diese Liste weniger als 10 Einträge, wird der Mittelwert aller Einträge gebildet und in Schritten im Abstand von 20 gerundet. Hat die Liste mehr als 10 Einträge, werden für selbige Berechnung nur die letzten 10 Einträge berücksichtigt. Der neue Wert wird zurückgegeben.

```
409 def bpm_mean(bpm):
410   if bpm > 40:
411     bpmlist.append(bpm)
412     if len(bpmlist) < 10:
413       npbpm = np.array(bpmlist)
414       mean = int(np.mean(npbpm))/10
415       mean = int(mean)*20
416 
417     else:
418       rel = bpmlist[-10:]
419       npbpm = np.array(rel)
420       mean = int(np.mean(npbpm))/10
421       mean = int(mean)*10
422     return mean
423   return bpm
```

Code 5.21: bpm_mean-Funktion

5.3.15. tick_laenge_ermitteln-Funktion

Diese Funktion ermittelt aus der Geschwindigkeit in **B**eats **P**er **M**inute die Tickdauer, für das Player-Attribut `tick_duration`. Das Finden einiger geeigneten Umrechnungsformel

gestaltete sich schwierig. Wir wussten nicht, wieviel Ticks ein Grundschlag hat. Wir legten also zugrunde, das wir es ersteinmal mit Taktarten zu tun hätten, die auf Vierteln basieren. Im folgenden bestimmten wir die Menge an Ticks, welche eine Viertel hat.

Um dies zu machen stellen wir die `tick_duration` auf 0,1 Sekunden und starteten eine Midi-Datei mit einer Tonleiter in Vierteln. Parallel dazu maßen wir die Länge jeder Viertel und bildeten aus unseren Messungen den Mittelwert. Die Messung ergab, dass eine Viertel im Schnitt eine Länge von 19,2 Sekunden hat. Da die Tickdauer auf eine zehntel Sekunde gesetzt war wussten wir entsprechend, das eine Sekunde zehn Ticks und 19,2 Sekunden demnach 192 Ticks hatten. Daraus folgte, das eine Viertel im Schnitt 192 Ticks hat. Die Grüßenordnung dieses Werts bestätige sich bei späteren Test, allerdings ist er wohl noch nicht ganz genau, da die Musik beim Dirigieren schleppt, und somit etwas langsamer ist, als dirigiert wird. Hier ist noch Korrekturbedarf.

Um die Frequenz von $\frac{1}{min}$ in $Hz = \frac{1}{s}$ umzurechnen wurde zunächst durch 60 geteilt ($\frac{bpm}{60}$). Da allerdings keine Frequenz, sondern die Periodenlänge für die weiteren Berechnungen verwendet wird, wird als nächstes der Kehrwert genommen ($\frac{60}{bpm}$). Nun ist die Länge eines Schlages bekannt. Da wir wissen, das ein Schlag 192 Ticks hat müssen wir also um die Ticklänge zu bestimmen lediglich noch durch 192 teilen:

$$\text{Ticklänge} = \frac{\frac{60}{bpm}}{192}$$

Diese Formel wird nun in der Funktion angewandt. Vorher wird jedoch getestet, ob das Ergebnis größer Null ist, um im folgenden einen Fehler bei der Division durch Null zu vermeiden.

```
425 def tick_laenge_ermitteln(bpm):
426   if (60/ bpm)/192 > 0:        #1 tick 0,1s# 1 viertel 19,2s#
427     l=(60/ bpm)/192
428     return l
```

Code 5.22: tick_laenge_ermitteln-Funktion

5.3.16. menu_visual-Funktion

Hier wird das Menü beschrieben, welches auf dem Bildschirm angezeigt wird, sobald man den Dirigierstab in das kleine gelbe Kuchenstück am linken unteren Bildrand bewegt. Die Funktion liest zuerst den aktuellen Frame der angeschlossenen Webcam ein. Mit `detect_color` wird die aktuelle Position bestimmt. Befindet sich diese nicht mehr im gelben Kuchenstück, wird die Schleife abgebrochen und das ursprüngliche Programm wird weitergeführt. Der Frame wird gespiegelt und anschließend werden die Menüpunkte eingefügt und das Bild angezeigt.

```
430 def menu_visual(cap):
431   while(True):
432     _, frame = cap.read()
433     position = detect_color(frame)
434
435     new = in_circle(position)
```

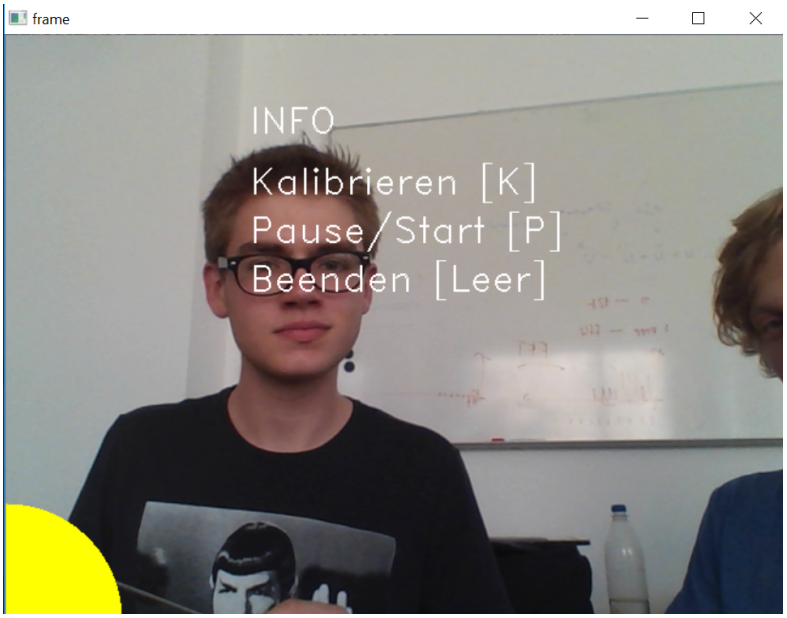


Abbildung 5.4.: Menü erscheint bei Berührung des Viertelkreises

```
436
437     if new != "menu":
438       break
439
440     frame = cv2.flip(frame,1)
441
442     #===Menue===#
443     cv2.circle(frame, (0,480), cthresh, (0,255,255), -1)
444     cv2.putText(frame, "INFO", (200,80), 2, 1, (255,255,255), 0)
445     cv2.putText(frame, "Kalibrieren [K]", (200,130), 2, 1, (255,255,255),
        0)
446     cv2.putText(frame, "Pause/Start [P]", (200,170  ), 2, 1, (255,255,255),
         0)
447     cv2.putText(frame, "Beenden [Leer]", (200,210  ), 2, 1, (255,255,255),
        0)
448     #===Ende===#
449
450     cv2.imshow('frame', frame)
451
452     if cv2.waitKey(1) & 0xFF == ord(' '):
453       cap.release()
454       sys.exit()
```

Code 5.23: menu_visual-Funktion

5.3.17. waehle_stueck-Funktion

Der Funktion wird sowohl die aktuelle Taktart, als auch der Pfad des Ordners, in dem sich die Midi-Dateien befinden übergeben. In dem Ordner gibt es drei Unterverzeichnisse, eins für jede Taktart. Abhängig von der aktuellen Taktart navigiert die Funktion in das entsprechende Unterverzeichnis. Dort wählt sie zufällig ein Stück aus. Dessen Pfad gibt sie als String zurück.

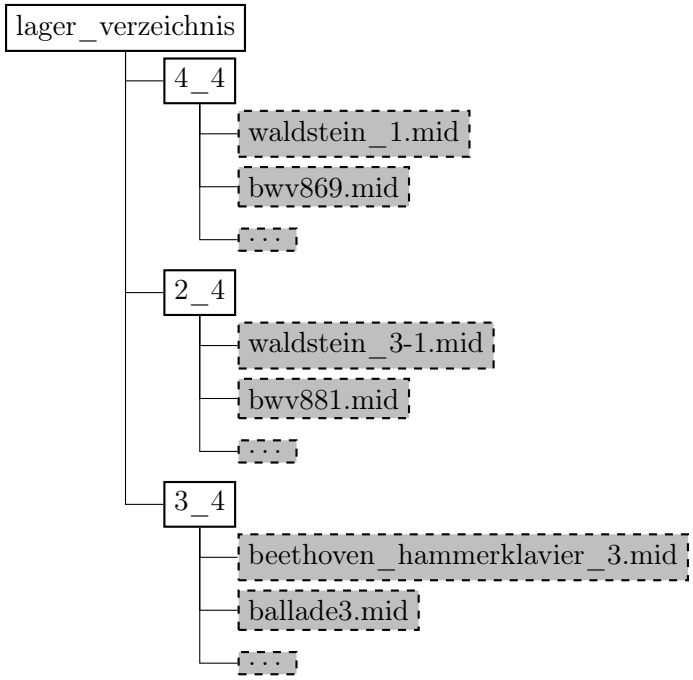


Abbildung 5.5.: Dateiorganisation

```
455 def waehle_stueck(taktart, lager_verzeichnis):
456   if taktart == 4:
457     stuecke = os.listdir(lager_verzeichnis + "/4_4")
458     art = "4_4"
459   elif taktart == 2:
460     stuecke = os.listdir(lager_verzeichnis + "/2_2")
461     art = "2_2"
462   elif taktart == 3:
463     stuecke = os.listdir(lager_verzeichnis + "/3_4")
464     art = "3_4"
465   else:
466     print "Ungueltige Taktart"
467   auswahl = random.choice(stuecke)
468   return  lager_verzeichnis +  "/" + art + "/" + auswahl
```

Code 5.24: waehle_stueck-Funktion

6. Reflektion

6.1. Lernerfahrung

Neben dem fachlichen Wissen, welches sich sicherlich jeder von uns angeeignet hat, haben wir auch einige Dinge über die Arbeit in der Gruppe und darüber, wie es ist selbstständig und längerfristig an einem Projekt zu arbeiten, aus diesem Semester mitgenommen.

6.2. Schwierigkeiten

Im Laufe des Semesters gab es selbstverständlich auch diverse Schwierigkeiten, mit denen wir umgehen mussten. Neben Schwierigkeiten beim Schreiben komplexer Programme, auf die im Kapitel »Projektverlauf« genauer eingegangen wird, gab es des öfteren Probleme dabei, die verschiedenen Erweiterungen für Python auf allen Rechnern zum Laufen zu bringen. In der Gruppe existierte über weite Strecken außerdem kein einheitliches Programm, welches von allen Gruppenmitgliedern als Arbeitsgrundlage verwendet wurde. Grund hierfür waren unterschiedliche Vorlieben beim Programmieren. Im Endeffekt hat diese Tatsache zu viel Verwirrung gestiftet im Vergleich zu dem Nutzen, den sie kurzfristig lieferte. Hierbei würden wir aus der Retrospektive anders vorgehen. Eine Versionsverwaltung über *git* wäre wohl eine Alternative gewesen.

6.3. Ausblick

6.3.1. und jetzt – wie geht es weiter?

Glücklicherweise sind wir mit unserem Projekt soweit gekommen, dass man es auch als abgeschlossen betrachten kann. Sicherlich gibt es einige Punkte, die verbesserungswürdig sind, aber das zu Beginn formulierte Ziel wurde durchaus erreicht. Diese Dokumentation und unsere Programme wollen wir so veröffentlichen, dass sich jeder Interessierte alles anschauen, verstehen und bei Interesse eigenständig weiterarbeiten kann. Als verknüpfende Plattform wurde GitHub gewählt. Wir sind sehr gespannt, ob das Projekt weitergeführt werden wird, ob von uns oder von Fremden.

6.3.2. weitere Projektideen

Im Laufe der Projektarbeit sind uns einige weitere Ideen für Programme gekommen. Vielleicht sind diese eine Inspiration für andere Programmierfreudige. Auch wir werden uns an manchen Stellen an diesen Programmen versuchen.

Flugradar

Bei unserem ersten Treffen zur Dokumentation saßen wir auf einem Balkon einer Dachwohnung an der Startschneise des Flughafen Tegels. Die Flugzeuge flogen recht nah an uns vorbei und schnell haben wir es uns zur Gewohnheit gemacht, auf `https://www.flightradar24.com/` zu schauen welches Flugzeug, welche Airline und welches Ziel an uns gleich vorbeifliegt. Wir haben die Flugzeuge früher gehört als gesehen. Um nicht immer wieder das Browserfenster wechseln zu müssen dachten wir, ein kleines Python-Programm, welches bei Annäherung eines Flugzeuges eine Nachricht aufpoppen lässt wäre praktisch.

Tatsächlich stellt `https://opensky-network.org/` sogar eine Python-API zu Verfügung, bei der man sich alle 10 Sekunden die Daten aller Flugzeuge weltweit laden kann. Diese könnten dann auf sich nähernde Flugzeuge gescannt werden. Leider gelang es noch nicht diese Bibliothek korrekt zum laufen zu bekommen.

Aquarellbild

Ein Programm, welches von selbst ein ästhetisches Aquarellbild malt wäre sicher eine schöne Sache...

Schreiberkennung

Diese Programmidee ist tatsächliche eine, wie wir finden ziemlich coole und hängt auch mit unserem Programm zusammen. Wir bewegen unseren Stab auf dem Bildschirm und die Postion wird erkannt. Wenn man mit unserem Programm eine Weile arbeitet kann man das Gefühl bekommen, man würde vor einem Spiegel stehen und auf ihm zeichnen. Und das ist die Grundidee. Man nutzt die Luft als Tafel und gleichzeitig kann der, dem man etwas aufschreibt das geschriebene »richtigherum« lesen. Man könnte solch ein Programm wohl recht einfach mit dem bereits erworbenen Wissen programmieren. Die Funktionen finden schon im hier vorgestellten Programm Verwendung. Das Gezeichnete könnte man in einem dreidimensionalen Array Speichern – Farbe und vielleicht auch Transparenz des Geschriebenen. In der Weiterentwicklung wäre dann die Videospeicherung und vielleicht auch die Tonaufnahme möglich. Ein Erklärvideo bei YouTube, bei dem dich der Lehrer anschaut hätte doch etwas...

interaktives Snakespiel

Punkte oder eine Schlange, die dem Stab folgen und sich nicht in den Schwanz beißen dürfen – für zwischendurch vielleicht unterhaltsam...

Anhang

A. Codeverzeichnis

B. Abbildungsverzeichnis

Programm

C.1. Installationshinweise

C.1.1. benötigte Programme

1. Python
2. Fluidsynth

C.1.2. besondere benötigte Bibliotheken

1. numpy
2. cv2
3. pickle
4. midi
5. fluidsynth
6. threading

C.2. Einrichtung

C.2.1. Allgemein

Folgende Variablen sollten vor Programmausführung richtig gesetzt werden:

1. für das Verwenden der Kamera: `cap = cv2.VideoCapture(0)` für webcam: 0 für externe webcam: 1 (Z. 508)
2. für die Verwendung der Foundfont-Datei:

   `self.sfid = self.fs.sfload("../FluidR3_GM.sf2")` hier muss auch der Pfad in welchem die `FluidR3_GM.sf2` liegt angepasst werden. (Z. 46)
3. um eine entsprechende Midi-Datei auswählen zu können muss eine Verzeichnisstruktur, wie in Abbildung 5.5 angelegt werden und mit Musik gefüllt werden. Das Verzeichnis in welchem dies geschieht muss unter

   `p.load(waehle_stueck(counter(), "../Midi"))` angegeben werden. (Anstelle von `"../Midi"`) (Z. 549)

Die letzten Beiden Punkte sind bereits in einem nichtöffentlichen Dateisystem unter `https://tubcloud.tu-berlin.de/index.php/s/tY9kTveBZcMhYw8` geschehen.

C.2.2. unter Windows

Unter Windows sollte folgendes beachtet werden:

1. in der Datei sollte der Audiotreiber geändert werden von

   `self.fs.start(driver="alsa")` auf `self.fs.start(driver="dsound")`. (Z. 44)

2. damit pyfluidsynth verwendet werden kann muss das `libfluidsynth.dll` für die entsprechende Vindowsversion (64/32-Bit) heruntergeladen werden und in das Verzeichnis, in welchem auch die pyfluidsynth-Bibliothek liegt, gespeichert werden. Zusätzlich muss in der Datei `FLUIDSYNTH.PY` aus der pyfluidsynth-Bibliothek der Code zu Beginn wie folgt geändert werden:

```
# A short circuited or expression to find the FluidSynth library
# (mostly needed for Windows distributions of libfluidsynth supplied
    with QSynth)

lib = find_library('fluidsynth') or find_library('libfluidsynth') or
    find_library('libfluidsynth-1')

if lib is None:
  #raise ImportError, "Couldn't find the FluidSynth library."
   fl=WinDLL("C:\Users\User\Anaconda2\Lib\site-packages\libfluidsynth64
    .dll")
   lib="ok"
else:
   _fl=CDLL(lib)
```

Code C.1: FLUIDSYNTH.PY

Selbstverständlich muss dabei der Pfad, in welchem die `libfluidsynth64.dll` Datei liegt entsprechend angepasst werden.

C.3. Code komplett

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3  #
4  #  Das_Orchester_ist_Programm-Programm.py
5  #
6  #  This program was created as part of the laboratory Mathesis at the
       Technical University Berlin .
7  #  Copyright 2016 Henriette Behr, Henriette Rilling, Max Wehner, Robin
       Krueger <das_orchester_ist_programm@web.de>
8  #
9  #  This program is free software; you can redistribute it and/or modify
10 #  it under the terms of the GNU General Public License as published by
11 #  the Free Software Foundation; either version 2 of the License, or
12 #  (at your option) any later version.
13 #
14 #  This program is distributed in the hope that it will be useful,
```

```
15 #  but WITHOUT ANY WARRANTY; without even the implied warranty of
16 #  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
17 #  GNU General Public License for more details.
18 #
19 #  You should have received a copy of the GNU General Public License
20 #  along with this program; if not, write to the Free Software
21 #  Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
22 #  MA 02110-1301, USA.
23
24 from __future__ import division
25 import time
26 import random
27 import numpy as np
28 import cv2
29 import sys
30 from copy import deepcopy
31 import pickle
32 import midi
33 import fluidsynth
34 import threading
35 import os
36
37 #=====Object from Stefan=====#
38
39 class Player(object):
40
41   def __init__(self):
42     self.d=[]
43     self.fs = fluidsynth.Synth()
44     self.fs.start(driver="alsa")
45
46     self.sfid = self.fs.sfload("FluidR3_GM.sf2")
47     self.fs.program_select(0, self.sfid, 0, 0)
48
49     self.tick_duration = 0.01
50
51
52   def load(self, filename):
53     '''lädt ein Midi-File mit Namen filename.'''
54     with open(filename,'rb') as f:
55       self.d=midi.fileio.read_midifile(f)
56     self.d.make_ticks_abs()
57
58   def play(self):
59     '''spielt die Datei in Echtzeit ab, die zugrundeliegende
60     Zeiteinheit self.tick_duration lässt sich während des Abspielens
61     ändern.'''
62     self.stop = False
63     for voice in self.d:
64       voice_thread = threading.Thread(target=self.play_voice, args=(voice,)
     )
65       voice_thread.start()
66
67   def stopit(self):
```

```
68      self.stop = True
69
70    def play_voice(self,voice):
71      voice.sort(key=lambda e: -e.tick)
72
73      tick_now = 0
74
75      time_now = time.time()
76
77      while len(voice)>0 and not self.stop :
78        event=voice.pop()
79        while event.tick>tick_now:
80          #print event
81          n_ticks= max(int((time.time()-time_now)//self.tick_duration)+1,
      event.tick-tick_now)
82          #print n_ticks, n_ticks*self.tick_duration-time.time()+time_now
83          time.sleep(n_ticks*self.tick_duration-time.time()+time_now)
84          tick_now +=n_ticks
85          time_now=time.time()
86
87        self.send(event)
88
89    def send(self, event):
90      '''Diese Methode kommuniziert mit dem Synthesizer. Bisher sind
91      nur drei Ereignistypen implementiert.'''
92      if type(event) is midi.NoteOnEvent:
93        self.fs.noteon(event.channel,event.data[0],event.data[1])
94      elif type(event) is midi.NoteOffEvent:
95        self.fs.noteoff(event.channel, event.data[0])
96      elif type(event) is midi.ProgramChangeEvent:
97        self.fs.program_change(event.channel, event.data[0])
98      elif type(event) is midi.ControlChangeEvent:
99        self.fs.cc(event.channel, event.data[0],event.data[1])
100
101 #=====Functions=====#
102
103 def detect_color(frame): #Funktion, welche eine nach Farbe definierte
      Stelle ermittelt
104
105   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)   #konvertiert frame in hsv
106   lower = np.array([hue_min, sat_min, val_min])  #legt untere schranke fest
107   upper = np.array([hue_max, sat_max, val_max])  #legt obere schranke fest
108
109   #Die Wert aus den Schranken werden aus globalen Variablen gezogen
110
111   mask =  cv2.inRange(hsv, lower, upper)                #erstellt maske,
        welche für werte innerhalb des intervalls
112                                     #den wert 1 annimmt, sonst 0
113   y_werte, x_werte = np.where(mask == 255)              #Es werden die x-
        und y-Werte ausgelesen, welche ein True (255) bekomen haben
114   if len(x_werte) > 20:                           ###es reicht eine bedingung
        zu erfüllen + schwellenwert
115     y_mittel = int(np.mean(y_werte))                #Es wird der Mittelwert
         aus allen y-Werten gebildet
```

```
116     x_mittel = int(np.mean(x_werte))                #Es wird der Mittelwert
        aus allen x-Werten gebildet
117     position = (x_mittel, y_mittel)                 #Die Mittlere Position
        aller Trues entspricht dem Tupel beider Mittelwerte
118
119   else:                                   ###if-bedingung unnötig, mittelpunkt
        erhält man durch tausch von x und y
120     y_shape, x_shape, _ = frame.shape               #Es werden die Bildmaße
        ausgelesen
121     position = (int(x_shape//2),int(y_shape//2))          #Als Position
        wird der Bildmittelpunkt gewählt
122
123   return position                                 #Ergebnis wird zurückgegeben
124
125 def get_values(cap):
126   while(True):
127     _, frame = cap.read()
128     y_shape, x_shape, _ = frame.shape
129     cv2.circle(frame,(x_shape//2, y_shape//2),25,(0,0,255),4)
130     frame = cv2.flip(frame,1) #Spiegelung des frames an der Horizontalen
131     cv2.putText(frame, "Bereit? [B]", (200,450), 2, 1, (255,255,255), 0)
132     cv2.imshow('frame', frame)
133     if cv2.waitKey(1) & 0xFF == ord('b'): #Signal, dass in Position, über
        das Drücken von k
134       break
135     if cv2.waitKey(1) & 0xFF == ord(' '):
136       cap.release()
137       sys.exit()
138
139   ret, frame = cap.read() #frame ohne Kreis wird geholt
140   hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV) #Konvertierung in hsv
141   radius = 25 #Radius des zu untersuchenden Kreises
142   kreisliste = [] #diese Liste soll alle Pixel enthalten, die innerhalb des
        Kreises liegen
143   for i in range(x_shape): #das Bild wird pixelweise durchgegangen. Wenn
        der Pixel im Bild liegt, wir dieser registriert
144     for j in range(y_shape):
145       if np.sqrt((x_shape//2-i)**2+(y_shape//2-j)**2) < radius: #Hier Satz
        des Pythagoras zur Entfernungsbestimmung Pixel—Kreismittelpunkt
146         kreisliste.append(hsv[j][i])
147   npkreisliste = np.array(kreisliste) #Konvertierung in Array zur weiteren
        Verarbeitung
148
149   hue = npkreisliste[:,0] #Array mit allen hue-Werten wird angelegt
150   sat = npkreisliste[:,1] #Array mit allen sat-Werten wird angelegt
151   val = npkreisliste[:,2] #Array mit allen val-Werten wird angelegt
152
153     #Berechung der Grenzen erfolgt, wie folgt. Zuerst wird der Mittelwert
        aller Werte einer Eigenschaft gebildet.
154     #Als nächstes wird jeweils die Standartabweichung ermittelt.
155     #Die untere Schranke ist dann einfach der Mittelwert - die
        Standartabweichun
156     #Die obere Schranke ist einfach der Mittelwert + die Standartabweichung
        .
```

```
157
158     #Die Ergebnisse werden zur direkten Weiterverwendung in globale
        Variablen geschrieben.
159     #Zur verwendung nach einem Programmneustart werden sie in eine datei
        ausgelagert.
160
161   global hue_min #globale Variable wird angelegt
162   hue_min = int(np.mean(hue) - np.std(hue)) #Wert wird ermittelt
163   global hue_max
164   hue_max = int(np.mean(hue) + np.std(hue))
165   global sat_min
166   sat_min = int(np.mean(sat) - np.std(sat))
167   global sat_max
168   sat_max = int(np.mean(sat) + np.std(sat))
169   global val_min
170   val_min = int(np.mean(val) - np.std(val))
171   global val_max
172   val_max = int(np.mean(val) + np.std(val))
173
174   sicherung = (hue_min,hue_max,sat_min,sat_max,val_min,val_max) #erzeuge
        Datensatztupel zur Abspeicherung für Pickle
175   output = open('hsv_werte.pkl', 'w') #die Ausgabedatei wird vorbereitet
176   pickle.dump(sicherung, output) #die Daten werden geschrieben
177   output.close() #der Output wird geschlossen
178
179 def schwellenwerte_einlesen():
180   global hue_min #die Schwellenwerte werden als global definiert
181   global hue_max
182   global sat_min
183   global sat_max
184   global val_min
185   global val_max
186
187   try: #es wird versucht / festgestellt, ob es bereits eine Datei mit
        gespeicherten Schwellenwerten gibt
188     f = open("hsv_werte.pkl") #falls ja wird diese geoeffnet
189     sicherung = pickle.load(f)
190     hue_min, hue_max, sat_min, sat_max, val_min, val_max = sicherung #und
        die Daten entpackt
191
192   except: #falls nein, werden sehr komische Werte festgelegt, damit der
        Benutzter das Programm kalibriert
193     val_max = 1
194     val_min = 0
195     sat_max = 1
196     sat_min = 0
197     hue_max = 1
198     hue_min = 0
199
200 def detect_gesture(zu_untersuchen, figur):
201   figurlaenge = len(figur)
202
203   zu_untersuchen_kurz = [] #Diese Liste soll die letzten dirigierten
        Vektoren enthalten, die wichtig sind
```

```
204   for i in range(figurlaenge,0,-1): #von index -1 bis index 0
205     zu_untersuchen_kurz.append(zu_untersuchen[-i]) #letzten Eintraege
        werden ermittelt
206   zu_untersuchen_kurz = np.array(zu_untersuchen_kurz) #zur weiteren
        Verarbeitung Konvertierung in Array
207
208   figurensammlung = [] #verschiedene Kombinationen werden angelegt,
        schwierig sich das ohne zeichnung vorzustellen. Bsp.: aus [1,2,3] wird
        [[1,2,3],[2,3,1],[3,2,1]]
209   tmp = figur
210   for i in range(figurlaenge):
211     tmp.append(tmp[0])
212     del(tmp[0])
213     anhaengen = deepcopy(tmp)
214     figurensammlung.append(anhaengen)
215   figurensammlung = np.array(figurensammlung) #zur weiteren Verarbeitung
        Konvertierung in Array
216
217   for i in range(figurlaenge): #Der Vergleich findet statt
218     if np.array_equiv(figurensammlung[i],zu_untersuchen_kurz): #
        elementweises Vergleichen von letztem Dirigat und verschiedenen Figur-
        Kombinationen
219       return True #falls Figur gefunden
220
221   return False #falls Figur nicht gefunden
222
223 def detect_bpm(l):
224   global a
225   global b
226   global bpm
227   if l > 2:
228     b = time.clock()
229     bpm = 60/(b-a)
230     a = b
231   else:
232     a = time.clock()
233
234 def get_vec(old, new):
235   if new == None:
236     return None
237   elif old == new:
238     return None
239   elif old == 1:
240     if new == 2:
241       return (1,0)
242     elif new == 3:
243       return (1,-1)
244     elif new == 4:
245       return (0,-1)
246   elif old == 2:
247     if new == 1:
248       return (-1,0)
249     elif new == 3:
250       return  (0,-1)
```

```
251     elif new == 4:
252       return (-1,-1)
253   elif old == 3:
254     if new == 1:
255       return (-1,1)
256     elif new == 2:
257       return (0,1)
258     elif new == 4:
259       return (-1,0)
260   elif old == 4:
261     if new == 1:
262       return (0,1)
263     elif new == 2:
264       return (1,1)
265     elif new == 3:
266       return (1,0)
267
268 def get_visual(pos):
269   if new == 1:
270     cv2.circle(frame, c1, cthresh, color, 3)
271     cv2.circle(clean, c1, cthresh, color, 3)
272   if new == 2:
273     cv2.circle(frame, c2, cthresh, color, 3)
274     cv2.circle(clean, c2, cthresh, color, 3)
275   if new == 3:
276     cv2.circle(frame, c3, cthresh, color, 3)
277     cv2.circle(clean, c3, cthresh, color, 3)
278   if new == 4:
279     cv2.circle(frame, c4, cthresh, color, 3)
280     cv2.circle(clean, c4, cthresh, color, 3)
281
282   cv2.circle(frame, c1, cthresh, color, 3)
283   cv2.circle(frame, c2, cthresh, color, 3)
284   cv2.circle(frame, c3, cthresh, color, 3)
285   cv2.circle(frame, c4, cthresh, color, 3)
286   cv2.circle(frame, pos, 4, (0,0,255), -1)
287   cv2.circle(clean, pos, 4, (0,0,255), -1)
288   cv2.circle(frame, (640,480), cthresh, (0,255,255), -1)
289
290 def get_length(a,b):       #berechnet länge eines vektors von 2 punkten
291   ax, ay = a
292   bx, by = b
293   cx = ax - bx
294   cy = ay - by
295   l = np.sqrt(cx**2+cy**2)
296   return l
297
298 def in_circle(pos):
299
300   vec1 = get_length(c1, pos)    #sammlung der längen der vektoren von den
        mittelpunkten zur position
301   vec2 = get_length(c2, pos)
302   vec3 = get_length(c3, pos)
303   vec4 = get_length(c4, pos)
```

```
304   vecmenu = get_length(cmenu, pos)
305
306   lengths = [cthresh+1, vec1, vec2, vec3, vec4, vecmenu]
307
308   for x in lengths:
309     if x < cthresh:
310       ret = lengths.index(x)
311       if ret == 5:
312         return "menu"
313       return ret
314
315 def beat_number(vec,figure):
316   a,b = vec
317
318   if figure == 'triangleA':
319     if vec == (0,-1):
320       return 1
321     if vec == (-1,0):
322       return 2
323     else:
324       return 3
325   if figure == 'triangleB':
326     if vec == (1,0):
327       return 1
328     if vec == (0,1):
329       return 2
330     else:
331       return 3
332   if figure == 'triangleC':
333     if vec == (-1,0):
334       return 1
335     if vec == (0,1):
336       return 2
337     else:
338       return 3
339   if figure == 'triangleD':
340     if vec == (0,-1):
341       return 1
342     if vec == (1,0):
343       return 2
344     else:
345       return 3
346
347   if figure == 'rectangleCCW':
348     if vec == (0,-1):
349       return 1
350     if vec == (1,0):
351       return 2
352     if vec == (0,1):
353       return 3
354     if vec == (-1,0):
355       return 4
356   if figure == 'rectangleCW':
357     if vec == (0,-1):
```

```
358         return 4
359       if vec == (1,0):
360         return 3
361       if vec == (0,1):
362         return 2
363       if vec == (-1,0):
364         return 1
365
366     if figure == 'line':
367       if vec == (0,-1):
368         return 1
369       else:
370         return 2
371     else:
372       return False
373
374 def check_gestures():
375     global figure
376     if len(veclist) > 4:
377       if detect_gesture(veclist, rectangleCW):
378           figure = "rectangleCW"
379           counter()
380       if detect_gesture(veclist, rectangleCCW):
381           figure = "rectangleCCW"
382           counter()
383       if detect_gesture(veclist, triangleA):
384           figure = "triangleA"
385           counter()
386       if detect_gesture(veclist, triangleB):
387           figure = "triangleB"
388           counter()
389       if detect_gesture(veclist, triangleC):
390           figure = "triangleC"
391           counter()
392       if detect_gesture(veclist, triangleD):
393           figure = "triangleD"
394           counter()
395       if detect_gesture(veclist, line):
396           figure = "line"
397           counter()
398
399 def counter():
400     f = len(figure)
401     if f > 10:
402       r = 4
403     else:
404       r = int(np.sqrt(f))
405
406     cv2.putText(mirror, str(beat_number(veclist[-1],figure)) + "/" + str(r),
        (40,40), 2, 1, (255,255,255), 0)
407     return r
408
409 def bpm_mean(bpm):
410     if bpm > 40:
```

```
411       bpmlist.append(bpm)
412       if len(bpmlist) < 10:
413         npbpm = np.array(bpmlist)
414         mean = int(np.mean(npbpm))/10
415         mean = int(mean)*20
416 
417       else:
418         rel = bpmlist[-10:]
419         npbpm = np.array(rel)
420         mean = int(np.mean(npbpm))/10
421         mean = int(mean)*10
422       return mean
423     return bpm
424 
425 def tick_laenge_ermitteln(bpm):
426     if (60/ bpm)/192 > 0:        #1 tick 0,1s# 1 viertel 19,2s#
427       l=(60/ bpm)/192
428       return l
429 
430 def menu_visual(cap):
431     while(True):
432       _, frame = cap.read()
433       position = detect_color(frame)
434 
435       new = in_circle(position)
436 
437       if new != "menu":
438         break
439 
440       frame = cv2.flip(frame,1)
441 
442       #===Menü===#
443       cv2.circle(frame, (0,480), cthresh, (0,255,255), -1)
444       cv2.putText(frame, "INFO", (200,80), 2, 1, (255,255,255), 0)
445       cv2.putText(frame, "Kalibrieren [K]", (200,130), 2, 1, (255,255,255),
          0)
446       cv2.putText(frame, "Beenden [Leer]", (200,170  ), 2, 1, (255,255,255),
          0)
447       #===Ende===#
448 
449       cv2.imshow('frame', frame)
450 
451       if cv2.waitKey(1) & 0xFF == ord(' '):
452         cap.release()
453         sys.exit()
454 
455 def waehle_stueck(taktart, lager_verzeichnis):
456     if taktart == 4:
457       stuecke = os.listdir(lager_verzeichnis + "/4_4")
458       art = "4_4"
459     elif taktart == 2:
460       stuecke = os.listdir(lager_verzeichnis + "/2_2")
461       art = "2_2"
462     elif taktart == 3:
```

```
463       stuecke = os.listdir(lager_verzeichnis + "/3_4")
464       art = "3_4"
465     else:
466       print "Ungueltige Taktart"
467     auswahl = random.choice(stuecke)
468     return  lager_verzeichnis +  "/" + art + "/" + auswahl
469
470 #=======gestures=======#
471
472 rectangleCW = [(1,0),(0,-1),(-1,0),(0,1)]
473 rectangleCCW = [(1,0),(0,1),(-1,0),(0,-1)]
474 triangleA = [(1,1),(0,-1),(-1,0)]
475 triangleB = [(-1,-1),(1,0),(0,1)]
476 triangleC = [(1,-1),(-1,0),(0,1)]
477 triangleD = [(-1,1),(0,-1),(1,0)]
478 line = [(0,1),(0,-1)]
479 gestures = [rectangleCW, rectangleCCW, triangleA, triangleB, triangleC,
        triangleD, line]
480
481 #=======circles=======#
482
483 c1 = ((320+120),(240-120))
484 c3 = ((320-120),(240+120))
485 c2 = ((320-120),(240-120))
486 c4 = ((320+120),(240+120))
487 cmenu = (640,480)
488 color = (10,10,255)
489
490 #=======variables=======#
491
492 global fnr, bpm, new
493
494 fnr = 0
495 bpm = 0
496 cthresh = 95
497 ref = (320,240)
498 old = 0
499 new = 5
500
501 #=======lists=======#
502
503 veclist = [None]              #liste beinhaltet richtungsänderungen
504 bpmlist = []
505
506 #=======program=======#
507
508 cap = cv2.VideoCapture(0)
509 schwellenwerte_einlesen()
510
511 while True:
512   _, frame = cap.read()
513   clean = np.copy(frame)
514
515   position = detect_color(frame)
```

```
516
517     get_visual(position)
518
519     new = in_circle(position)
520
521     if new == "menu":
522       menu_visual(cap)
523       continue
524
525     vec = get_vec(old, new)
526
527     if vec != None and vec != veclist[-1]:
528       veclist.append(vec)
529       detect_bpm(len(veclist))
530
531     if new != None and new != old:
532       old = new
533
534     bpm = bpm_mean(bpm)
535
536     frame = cv2.addWeighted(frame, 0.5, clean, 0.5, 0)
537     mirror = cv2.flip(frame,1)
538
539     check_gestures()
540
541     cv2.putText(mirror, str(int(bpm)), (560,40), 2, 1, (255,255,255), 0)
542
543     cv2.imshow("frame", mirror)
544     fnr += 1
545
546     if fnr == 200:
547       try:
548         p = Player()
549         p.load(waehle_stueck(counter(), "../Midi"))
550         p.play()
551       except:
552         print "Bitte neu starten und Dirigieren."
553     if fnr > 200:
554       try:
555         if bpm > 0:
556           p.tick_duration = tick_laenge_ermitteln(bpm)
557       except:
558         print "Fehler. Bitte neu starten."
559
560     if cv2.waitKey(1) & 0xFF == ord('k'):
561       get_values(cap)
562
563     if cv2.waitKey(1) & 0xFF == ord(' '):
564       if fnr > 200:
565         p.stopit()
566       print "herunterfahren"
567       break
568
569 cap.release()
```

```
570 cv2.destroyAllWindows()
```

Code C.2: Das komplette Programm

Das Programm kann als Datei aus dem GitHub-Repository der Gruppe heruntergeladen werden. Dieses ist zu finden unter `https://github.com/rbnkrgr/das_orchester_ist_programm`. Das Endprodukt trägt den Namen `Das_Orchester_ist_Programm-Programm.py`.