System zur Synchronisation von Musik mit Tanzanimationen für Unterhaltungsroboter

Bachelor-Thesis zur Erlangung des akademischen Grades Bachelor of Science (B.Sc.) im Studienfach Informatik

Philipp Camphausen

4525520

Abgabedatum: 14.06.2024

Letztmöglicher Abgabetermin: 15.06.2024

Erstprüfer: Prof. Dr. Ing. Udo Frese

Zweitprüfer: Dr. Ing. Robert Porzel

Betreuer: Prof. Dr. Ing. Udo Frese

Eidesstattliche Versicherung

Ich versichere an Eides statt, diese Arbeit im Rahmen der am Arbeitsbereich üblichen Betreuung selbstständig angefertigt und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Bremen, den 14.05.2024

Philipp Camphausen

Abstract

Diese Arbeit beschäftigt sich mit der Frage, inwiefern die Integration einer Tanzanimation in den Unterhaltungsroboter Doggy2 die Sichtbarkeit des Roboters auf Veranstaltungen verbessern kann. Dazu wird ein System entwickelt, um Tanzanimationen mit einem Musikstück zu synchronisieren. Mithilfe eines Beat-Erkennungs-Algorithmus wird der Zeitpunkt des nächsten Beats bestimmt, um Tanzanimationen aus einer erstellten Auswahl mit diesem zu synchronisieren. Es wurde zusätzlich eine Software zum Erstellen der Animationen auf Basis von Bezier Splines umgesetzt. Anhand einer Feldstudie soll schließlich evaluiert werden, ob das entwickelte System mehr Aufmerksamkeit auf Doggy lenkt. Außerdem wird das System von Betrachtern qualitativ evaluiert. Die Studienergebnisse weisen auf einen positiven Effekt hin, durch Probleme bei der Durchführung konnten allerdings weniger Daten erhoben werden als geplant, sodass die Aussagekraft des Ergebnisses eingeschränkt ist.

Inhaltsverzeichnis

1. Einführung

Roboter, die in der Lage sind, mit Menschen zu interagieren und Emotionen auszudrücken, wecken Faszination in Menschen. Ein solches Beispiel ist der Unterhaltungsroboter Doggy, der in der Arbeitsgruppe Multisensorische Interaktive Systeme der Uni Bremen für den Einsatz auf Veranstaltungen wie Messen und Firmenveranstaltungen entwickelt wurde, um Besuchern eine kurze spannende Interaktion in Form eines Ballspiels zu bieten (Tim Laue, 2014).

Trotz seiner Fähigkeit, den Besuchern einen Ball zurückzuspielen und Emotionen durch Animationen auszudrücken, gibt es jedoch eine Herausforderung: Doggy kann erst interagieren, wenn Menschen aktiv auf ihn zugehen. Dies führt oft dazu, dass er bei Veranstaltungen passiv steht und potenzielle Interaktionsmöglichkeiten ungenutzt bleiben. Daher kam die Idee auf, Doggy eine Bewegung während er nicht beschäftigt ist („Idle-Animation“) hinzuzufügen, um seine Präsenz zu erhöhen. Da auf Veranstaltungen üblicherweise Musik vorhanden ist, wurde weiterhin überlegt die „Idle-Animation“ in Form eines zur Musik passenden Tanzes umzusetzen.

Peng et. al. beschreiben robotischen Tanz als „ein intelligentes motorisches Verhalten bestehend aus anspruchsvoller Wahrnehmung, fortgeschrittenem Denken und Lernen und autonomer Entscheidungsfindung“. Es sei daher „ein wichtiges Mittel zur Erforschung und Entwicklung verschiedener Fähigkeiten von sozialen Robotern“ (Peng *u. a.*, 2015)

Da Tanzen weiterhin eine passende Aktivität für einen sozialen Roboter wie Doggy ist, wird die „Idle-Animation“ als „Tanzanimation“ umgesetzt. Im Anschluss wird die Effektivität von Doggy2 hinsichtlich seiner Anziehungskraft und der Anzahl der Interaktionen mit Menschen in einer Feldstudie untersucht. Durch diese Untersuchung soll ein tieferes Verständnis dafür gewonnen werden, ob und wie Softwaresysteme genutzt werden können, um die soziale Akzeptanz und Sichtbarkeit von Unterhaltungsrobotern zu steigern.

1.1. Ziele der Arbeit

Im Rahmen der Arbeit soll eine Grundlage geschaffen werden um weitere Forschung in Richtung der Integration von Tanzanimationen in Doggy zu ermöglichen. Dies wird durch Umsetzung folgender Punkte erreicht:

- Aufbauend auf der Idee des Verfahrens aus (Jaime Gancedo, 2018) werden die Beats pro Minute (BPM) und die Position der Beats im Musikstück bestimmt um die Animationen damit zu synchronisieren, sodass Doggy die Bewegungen bestenfalls „On Beat“, mindestens aber im ungefähren Tempo des Musikstücks durchführt.
- Das von (Goto, 2006) beschriebenen Verfahrens wurde von (Jayaram, 2018a) teilweise implementiert. Diese Implementation, wird leicht modifiziert verwendet um das Musikstück in Abschnitte zu unterteilen. Diesen werden dann verschiedene Tanzanimationen zugewiesen, um etwas Abwechslung in Doggy's Tanz zu bringen.
- Zum Erstellen der Animationen für die verschiedenen Abschnitte wird aufgrund der spezifischen Anforderungen und zu diesem Zweck nicht ausreichenden bestehenden Werkzeugen, ein graphischer Editor zum definieren der Animationen mittels Bezier Splines programmiert.
- Die in den ersten beiden Punkten genannten Verfahren werden asynchron berechnet. Aus dem Ergebnis und den über den Editor definierten Animationen werden dann Motorbewegungen für Doggy erzeugt, welche dann synchron mit dem Musikstück abgespielt werden. Dadurch tanzt Doggy dann zur Musik.

- Mithilfe einer Feldstudie soll validiert werden, ob das umgesetzte System hilfreich ist um auf einer Veranstaltung Besucher dazu einzuladen mit dem Roboter zu interagieren. Außerdem soll mit einem kurzen Fragebogen die Qualität des umgesetzten Systems evaluiert werden.

2. Grundlagen und Ähnliche Arbeiten

2.1. Hintergrund von Doggy2: Doggy

Doggy ist ein Ballspielender Unterhatungsroboter, der in der AG Multisensorische Interaktive Systeme der Universität Bremen gebaut und durch vorangegangene Arbeiten stetig weiterentwickelt wurde (Tzeng, 2013; Bartsch, 2015).

Abbildung 1. Links: Kostümdesign von (Tzeng, 2013),
Rechts: Doggy im B-Human Labor der AG MSIS (2024)

Doggy entstand als Weiterentwicklung von Piggy, einem stationären Roboter mit zwei Bewegungsachsen (Tim Laue, 2014). Piggy sollte mit einer minimalen Anzahl an Bewegungsachsen auf Veranstaltungen ein kurzes Ballspiel mit Menschen spielen können und dabei „sicher, flexibel, bezahlbar, und reaktiv“(Tim Laue, 2014) sein.
Der Roboter erhielt eine weitere Bewegunsachse sowie im Rahmen von (Tzeng, 2013) ein neues Aussehen, welches einen anthropomorphen Stoffhund darstellt (vgl. Abb. 1 links) und wurde so zu Doggy. In der Arbeit wurde außerdem eine Software zum Produzieren von Animationen vorgestellt. Die Animationen wurden erstellt, indem die Achsen des Roboters auf einen Spielecontroller gemappt und die ausgeführten Bewegungen dann aufgenommen wurden. Es wurden zudem einige Animationen zum Darstellen der Gefühle „happy“ (glücklich) und „sad“ (traurig) entwickelt, sowie ein Zustandsautomat zum Abspielen eines Verhaltensablaufs entworfen (Tzeng, 2013).
Aufbauend darauf verbesserte Spillner den Zustandsautomaten und kreierte weitere Animationen. Diese konnten, wenn auch nur in einer sehr kleinen Studie festgestellt, die Interaktion mit Doggy verbessern (Spillner, 2018).
Diese Animationen wurden weiter verbessert und mit Sounds kombiniert, die während der Animation abgespielt werden. Es wurde zudem ein neuer Zustandsautomat mit mehr Gefühlen entworfen, um die neuen Animationen und Sounds zu integrieren und festgestellte Probleme mit der vorherigen zu beheben (Weidenbach, 2019) .

Die erwähnte Verhaltensroutine von Doggy stand allerdings zum Zeitpunkt dieser Arbeit noch nicht für Doggy2 zur Verfügung. Die Integration des Softwarestacks von Doggy in Doggy2 wurde in einer anderen Arbeit angegangen, welche allerdings während der Bearbeitungszeit dieser Arbeit abgebrochen wurde.

2.2. **Doggy2 im Detail**

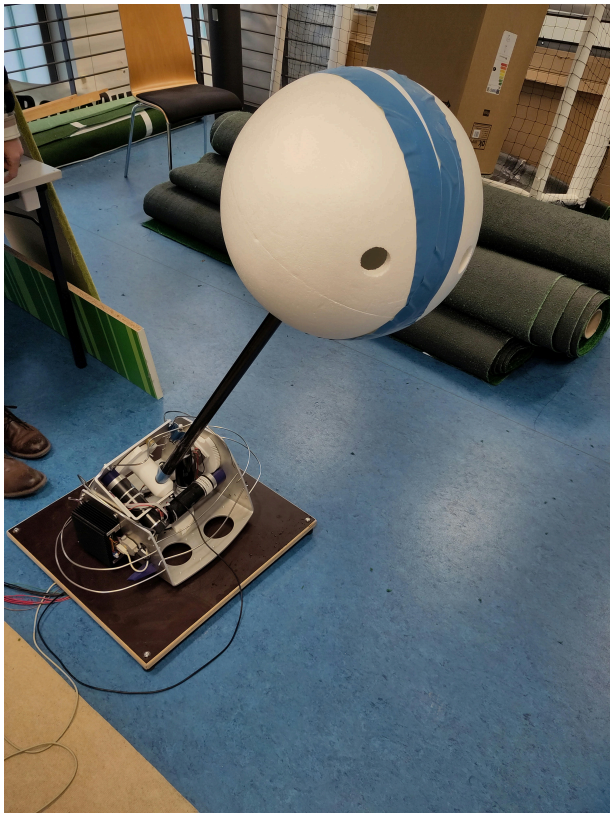

Abbildung 2. Doggy2 im B-Human Labor der AG MSIS (2024)

Doggy2 ist die neue Roboterhardware für Doggy. Dieser ist ca. 1,50m groß und besteht aus einer Stange mit einer Styroporkugel an der Spitze, welche den Kopf darstellt (vgl. Abb. 2). Über diese Konstruktion kann dann ein Kostüm gezogen werden, welches zum Zeitpunkt der Arbeit für Doggy2 allerdings noch nicht existierte. Die Herstellung eines passenden Kostüms konnte im Bearbeitungszeitraum auch nicht beauftragt werden. Daher musste die Studie im späteren Verlauf ohne Kostüm durchgeführt werden.
Ein Mikrocontroller steuert 3 Motoren, welche jeweils über einen Sensor die eigene Position kennen und zurückmelden können.
Der Mikrocontroller kann per USB mit einem Computer verbunden werden. Dies ermöglicht die Steuerung des Roboters über den verbundenen Computer.
Wörner implementierte Treibersoftware für den Mikrocontroller, die das Senden von Zielkoordinaten an den Roboter ermöglichen. Diese stellt die Grundlage zum Ansteuern des Roboters für diese Arbeit dar. Es war außerdem möglich, dem Roboter über eine grafische Benutzeroberfläche Zielkoordinaten in Form von Gradwerten zu senden, an welche sich der Roboter dann bewegt (Woerner, 2021).

2.3. Tanzende Roboter

Um einen geeigneten Ansatz für die genannte Problemstellung zu erarbeiten, wird nun ein Überblick über bestehende Systeme zur Integration von Tanzfähigkeiten in Roboter vorgestellt. Avrunin u. a. fanden heraus, dass „Einfache Mechanismen, wie Manipulation der Synchronität mit dem Rhythmus der Musik, Veränderung des Sets von Bewegungen und Synchronisation von Bewegungen des Roboters mit Ereignissen in der Musik die Wahrnehmung von Lebendigkeit, Tanzqualität und Unterhaltungswert beeinflussen."(Avrunin *u. a.*, 2011)

Einen guten Überblick über bereits existierende Systeme im Kontext von tanzenden Robotern stellen (Peng *u. a.*, 2015) dar. Die Metastudie kategorisiert verschiedenen Ansätze im Forschungsgebiet rund um tanzende Roboter und der aktuelle Stand in diesen Kategorien wird zusammengefasst. Es wird dabei grob in 4 Kategorien unterteilt:

- Kooperativer Tanz mit Menschen
- Imitation menschlicher Tanzbewegungen
- Synchronisation mit Musik
- Erstellung einer Choreografie

Die genannten Kategorien werden in weitere Unterkategorien unterteilt. Dabei wird jetzt auf die ersten beiden Kategorien, „Kooperativer Tanz mit Menschen" und „Imitation menschlicher Tanzbewegungen", nur kurz eingegangen, da sich herausstellen wird, dass diese im Weiteren weniger relevant für diese Arbeit sind.

Das liegt einerseits an den verwendeten Robotern: In diesen Kategorien wurde hauptsächlich mit humanoiden Robotern geforscht. Die einzige Ausnahme stellt der Roboter „Keepon" (Michalowski, Sabanovic und Kozima, 2007) dar. „Keepon" ist ein für soziale Interaktion konzipierter Roboter und hat mit vier Bewegungsachsen einen mit Doggy gut vergleichbaren Bewegungsspielraum, ist allerdings deutlich kleiner. Es wurde eine kooperative Tanzinteraktion für „Keepon" entwickelt, bei der der Roboter die Bewegungen des menschlichen Tanzpartners nachahmt (Michalowski, Sabanovic und Kozima, 2007). Da Doggy für kurze Interaktionen konzipiert ist (Tim Laue, 2014), wäre die Entwicklung eines weiteren Spiels in Form einer auf Tanz basierenden Interaktion denkbar und könnte Doggys Verhalten weiter bereichern. Die beiden Kategorien eignen sich allerdings nicht gut um auf Doggy aufmerksam zu machen, da eine bestehende Interaktion vorausgesetzt wird und werden deswegen nicht tiefergehend betrachtet.

2.4. Synchronisation mit Musik

Um Roboterbewegungen mit Musik zu synchronisieren bieten sich Beat-Erkennungs-Algorithmen an, da der Beat im Gegensatz zu anderen Eigenschaften von Musik relativ einfach bestimmt werden kann. Gute Beat-Erkennungs-Algorithmen, die mit vielen unterschiedlichen Musikgenres funktionieren, erfordern dennoch fortgeschrittenere Ansätze.

Oliviera, u. a. entwickelten einen Algorithmus zur prädiktiven Beat-Erkennung in Echtzeit für die Anwendung in interaktiv tanzenden Robotern. Dieser sollte „schnell auf Änderungen im Tempo der Musik reagieren, gegenüber Störgeräuschen robust laufen, sowie effizient berechnet werden können" (J. L. Oliveira *u. a.*, 2012). Dafür wurde ein auf konkurrierenden Programmteilen („Agenten") basierender Ansatz gewählt. Jeder der „Agenten" stellt eine Hypothese über die aktuelle BPM auf. Anhand verschiedener Bewertungskriterien bestimmt der „Agent" mit der wahrscheinlichsten Hypothese das Ergebnis (J. L. Oliveira *u. a.*, 2012).

Santiago u. a. integrieren den oben beschriebenen Algorithmus in einen humanoiden Roboter vom Typ „Robonova“, was dem Roboter ermöglichte, in Echtzeit und im Takt der Musik Animationen aus einer Bibliothek passend dazu auszuführen. (Santiago *u. a.*, 2011)

Ein ähnliches System wurde von Seo u. a. umgesetzt. Das System arbeitet ebenfalls in Echtzeit und bestimmt Position und zeitlichen Abstand zwischen zwei Beats. Das Verfahren zur Berechnung unterscheidet sich allerdings deutlich. Es wurde eine Kurzzeit Fast Fourier Transformation genutzt, um die Amplitude aller Frequenzen unter 200 Hz zu erhalten. Daraus werden dann Peaks erkannt und mit einem Mindestabstand zwischen Peaks sichergestellt, dass lokale Peaks zwischen Beats das Ergebnis nicht verfälschen. Durch die Länge des Fensters, auf dem die KFFT durchgeführt wird, variiert die Präzision des Algorithmus von ca. 5 BPM $(88 - 83)$ bis hin zu 14 BMP $(150 - 136)$ (Seo *u. a.*, 2013).

2.5. **Erstellung einer Choreografie**

Wie zu Beginn dargelegt, ist die Reaktion auf Ereignisse in der Musik relevant für die Wahrnehmung von Robotern. Nicht alle Ansätze, die in (Peng *u. a.*, 2015) betrachtet wurden, nutzen Informationen aus der Musik, um eine Choreografie zu erzeugen. Diese könnten dennoch hilfreiche Erkenntnisse enthalten, weshalb diese hier kurz zusammengefasst werden.

Shinazoki, Iwatani und Nakatsu konkatenieren Tanzschritte zufällig zu einer Tanzchoreografie. Die Idee stammte dabei aus Text to Speech Systemen, bei denen Text in Sprache umgewandelt wird, indem „Speech Units“ zu einer vollständigen Sprachausgabe zusammengesetzt werden. Es wurden in Zusammenarbeit mit einem Tänzer ca. 60 „Dance Units“ erstellt, aus denen der Roboter dann per Zufallsprinzip verschiedene Tänze generiert (Shinozaki, Iwatani und Nakatsu, 2008) .

Ein weiterer musikunabhängiger Ansatz ist Mapping Regeln zu verwenden, um eine dynamische Abfolge aus Bewegungen zu erstellen. Der Roboter nimmt dabei Ereignisse aus der Umgebung, wie beispielsweise Farben, Töne, Temperatur, etc. über Sensoren wahr. Diese sind dann mit Bewegungen verknüpft. Der Zustand der Umgebung bestimmt also über die Bewegung, die der Roboter ausführt. Ein Beispiel für einen solchen Ansatz setzten Oliviera u.a. um. Es wurde ein Lego Mindstorms NXT Roboter verwendet, der unter anderem über einen auf den Boden gerichteten Farbsensor seine Bewegung je nach Farbe des Untergrunds verändert (J. Oliveira *u. a.*, 2012).

Aucouturier, Ogai und Ikegami ließen die Bewegungsbahn eines fahrenden Roboters mithilfe eines FitzHugh-Nagumo Modells generieren. Dieses nutzt die Beats aus Musik als Eingabe (Aucouturier, Ogai und Ikegami, 2008). Dieser Ansatz generiert zwar interessante Bewegungen in Echtzeit, gibt allerdings die Kontrolle über die Bewegungen vollständig an das Modell ab, sodass gezielte Anpassungen nicht möglich sind.

3. Design des Systems

Ein Beat-Erkennungs-Algorithmus stellt die Basis vieler gezeigter Systeme dar. Die gezeigten Ansätze sind allerdings entweder für den Zeitrahmen zu kompliziert (Santiago *u. a.*, 2011; J. L. Oliveira *u. a.*, 2012) oder zu unpräzise (Seo *u. a.*, 2013), daher wurde nach einem Beat-Erkennungs-Algorithmus gesucht, der bei geringerem Implementierungsaufwand dennoch eine akzeptable Präzision bietet. Die Wahl fiel so auf den Ansatz von (Jaime Gancedo, 2018). Das System kann dann um Komponenten erweitert werden, die zusätzliche Eigenschaften des Musikstücks nutzen, um die resultierende Tanzanimation komplexer und interessanter zu gestalten. Die Animationen dann über ein Werkzeug zu erstellen, wie dies bereits in vorangegangenen Arbeiten zu Doggy der Fall war, überlässt dem Entwickler Kontrolle über das Ergebnis. Diese Verfahren so zu kombinieren, wurde zuvor noch nicht erforscht.

Das zu entwickelnde System soll nun also die Frequenz (BPM) und Position aller Beats ermitteln, damit vorgefertigte Animationen auf diese angepasst werden können. Zusätzlich wird das Musikstück in unterschiedliche Abschnitte unterteilt. Diesen Abschnitten können dann unterschiedliche Animationen zugewiesen werden. So entsteht eine „Tanzchoreografie“.

3.1. Rahmenbedingungen

Die Aufnahme von Sound über ein Mikrofon würde neue Probleme einführen, die spielende Musik müsste von anderen Umgebungsgeräuschen getrennt werden.

Daher war eine erste Idee, die Musikdaten per Audiokabel an den Roboter zu senden, welcher diese dann in Echtzeit verarbeitet. Aufgrund der Komplexität der vorgestellten Echtzeitalgorithmen zur Beat-Erkennung und Choreografieerzeugung, wurde stattdessen die Entscheidung getroffen, das System die Tanzchoreografie vor der Ausführung berechnen zu lassen und das Musikstück dann synchronisiert mit dem Tanz abzuspielen, da es für den Betrachter keinen Unterschied macht, ob die Berechnung in Echtzeit stattfindet oder nicht. Es entsteht so allerdings ein Nachteil für den praktischen Einsatz auf Veranstaltungen, da je nach den technischen Gegebenheiten der Veranstaltung nicht immer die Möglichkeit besteht, die Musik über den mit Doggy2 verbundenen Rechner einzuspielen (z.B. wenn die Veranstaltenden einen DJ o. ä. organisiert haben).

Das System benötigt also eine Komponente zur Beat-Erkennung, sowie eine Komponente, die die eingehenden Audiodaten basierend auf dem Inhalt in Abschnitte unterteilt. Dann können den verschiedenen Abschnitten jeweils Animationen zugeteilt werden, welche dann mit den Ergebnissen der Beat-Erkennung auf das Musikstück synchronisiert werden können.

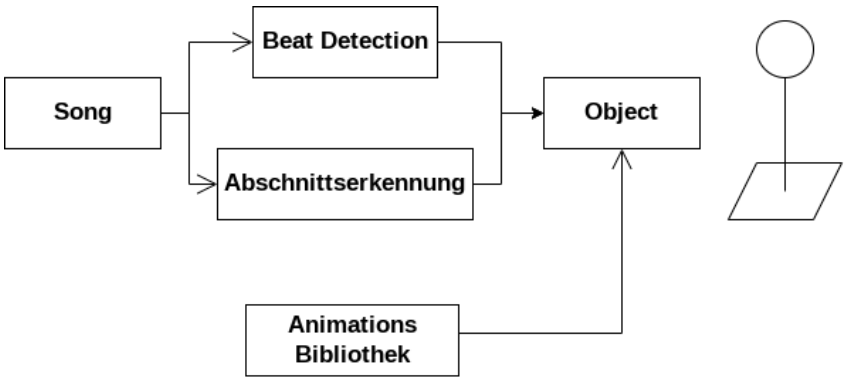

Abbildung 3. Aufbau des Systems

Im weiteren werden nun die Komponenten im Detail betrachtet.

3.2. **Komponenten**

3.2.1. *Beat Erkennung*

Eine Kernkomponente des Systems ist der Beat Detection Algorithmus. Dieser ist notwendig, um den Zeitpunkt des nächsten Beats zu bestimmen, damit die Bewegung des Roboters darauf abgestimmt werden kann. Der Beat wird dazu als lauteste wiederholende Frequenz in einem gewählten Zeitfenster betrachtet. Aufgrund des Einsatzgebiets des Roboters auf Veranstaltungen wird der Algorithmus darauf entwickelt, gut mit Musik aus den Genres Pop, Electronic und Techno zu funktionieren. Bei Musikstücken dieser Genres befindet sich der Beat meist im Bassbereich. Daher sollte der Einfluss von Frequenzen außerhalb des Bassbereichs auf das Ergebnis zu verringert werden.
Der im Weiteren genutzte Ansatz basiert auf (Jaime Gancedo, 2018). Da der dargestellte Ansatz allerdings nicht wie beschrieben repliziert werden konnte, wurden Anpassungen vorgenommen. Weiterhin ist der Algorithmus initial für die Anwendung in Echtzeit konzipiert, wobei sich Veränderungen der Beats pro Minute (BPM) nur mit einer Verzögerung auf das Ergebnis auswirken.
Die Kernidee des vorgestellten Algorithmus ist, die BPM zu berechnen, indem die Samples des Musikstücks in Blöcke unterteilt werden, deren Amplitude dann berechnet wird. Es entsteht so eine Historie aus der zusammengefassten Lautstärke im Verlauf des Musikstücks. Es bleibt dann ein Abschnitt der letzten 12 Sekunden dieser Historie gespeichert, zu welcher neue Blöcke nach der Aufnahme hinzugefügt und dafür der älteste Block im Abschnitt entfernt wird. Auf diesem Abschnitt wird dann nach jedem neuen Block eine Fast Fourier Transformation (FFT) durchgeführt (Jaime Gancedo, 2018). Aus dem neuen Block wird x_{avg} berechnet, dies stellt die summierte Amplitude des gesamten Blocks dar. Dann wird der nächste Wert für den Abschnitt bestimmt, indem x_{avg} mit dem Wert des letzten Blocks aus dem Abschnitt verrechnet wird. Es wurde $\alpha = 0.8$ verwendet. Die folgenden beiden Formeln sind direkt aus (Jaime Gancedo, 2018) entnommen.

$$x_{\text{avg}} = \sum_{n=0}^{B-1} [x]^2$$

$$P[n] = \alpha \cdot P[n-1] \cdot (\alpha - 1)x_{\text{avg}}$$

Betrachtet man die beiden Gleichungen in dieser Reihenfolge, funktioniert der Algorithmus allerdings nicht. $x_{\text{avg}} \cdot 0.2$ wird durch die Multiplikation mit $P[n-1]$ bei jedem neuen Block im Verlauf des Songs, insbesondere bei leisen Abschnitten im Song, immer kleiner. Dadurch tendiert das Ergebnis insbesondere bei Musikstücken, die leise starten, sehr schnell gegen 0, bis die untere Grenze der Fließkommapräzision erreicht ist. Dies macht weitere Verarbeitungsschritte unmöglich, da nach Berechnung der FFT alle Frequenzen bis auf 0 eine 0 als Ergebnis haben. Es gibt meiner Ansicht nach zwei Möglichkeiten, was hier schiefgelaufen ist:

- In der zweiten Gleichung steht fälschlicherweise $\cdot$ statt $+$. Mit einem $+$ stellt die Gleichung einen simplen Tiefpassfilter dar, was sinnvoll sein könnte. Allerdings würde dieser Tiefpassfilter auf die Amplitude und nicht das Audiosignal selbst angewendet und wirkt sich so eher nachteilig auf den folgenden Berechnungsschritt aus.
- Die zweite Gleichung soll vor der ersten auf den Block, statt auf x_{avg} angewendet werden. In diesem Fall stimmt auch der Wertebereich der im Paper gezeigten Graphen der Amplitude ungefähr mit meinen Ergebnissen überein. Das sorgt al-

lerdings nur dafür, dass der Unterschied in der Amplitude von lauten und leisen Stellen betont wird (vgl. Abb. 4).

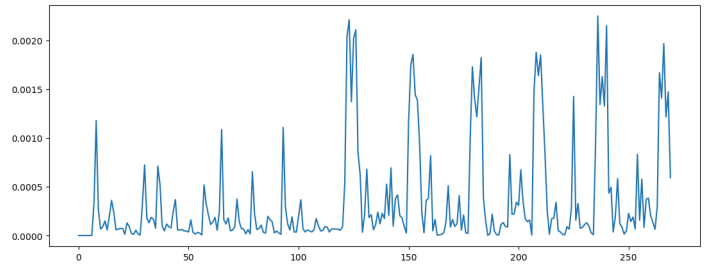

Abbildung 4. Amplitudenverlauf bei Anwendung der Formel auf den Block

Wegen der genannten Unklarheiten wurde letztendlich nur die Grundidee des Algorithmus übernommen, nämlich das Audiosignal in Blöcke zu unterteilen, die Amplitude zu berechnen und darauf eine FFT auszuführen.

Im vorgestellten Ansatz wurde ein Echtzeitsignal als Eingabe verwendet. Da wir stattdessen mit Audiodateien arbeiten, muss beachtet werden, dass das Signal in Stereo vorliegt. Zu Beginn der Verarbeitung wird ein Schritt hinzugefügt. Um tiefe Frequenzen, in denen sich der Beat meistens befindet, stärker zu betonen, wird ein Tiefpassfilter auf beide Kanäle angewendet.
Hierbei ist x ein Sample des Signals, i bezeichnet den Index

$$x[i] = x[i-1] * 0.95 + x[i] * 0.05$$

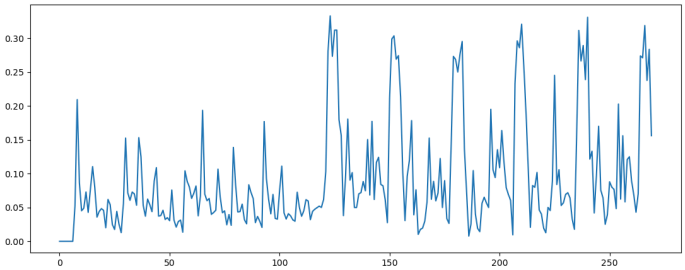

Abbildung 5. Amplitudenverlauf nach Anwendung des Tiefpassfilters

Da das Signal in Stereo vorliegt, müssen die beiden Kanäle zu einem zusammengefasst werden. Um dabei zu vermeiden, dass sich Frequenzen aus dem rechten und linken Kanal auslöschen, wird die Amplitude der Samples aus den Kanälen ausgerechnet, bevor diese zusammengenommen werden. Hierbei ist B die Blockgröße, l und r sind jeweils Samples aus dem linken, bzw. rechten Kanal.

$$\sum_{n=0}^{B-1} l^2 + r^2$$

Als Blockgröße wurden 980 Samples, ausgehend von einer Samplingrate des Musikstücks von 44100Hz, gewählt. Das entspricht 1/45 Sekunde und liegt darin begründet, dass bei

dieser Dauer der Beat einen über wenige Samples dauernden Peak im Amplitudenverlauf erzeugt und zudem $\frac{44100}{45}$ ohne Rest teilbar ist.

Die Samples der so summierten Blöcke der letzten 12 Sekunden bleiben nun als Window gespeichert. Sobald ein neuer Block verfügbar ist, wird der älteste entfernt und der neue hinzugefügt. Auf diesen Werten wird dann die FFT durchgeführt. Der Beat ist jetzt die lauteste Frequenz des Ergebnisses aus der FFT im Bereich zwischen 1-3Hz oder 60-180 BPM. Dies sind bei der genannten Abschnittsgröße von 12 Sekunden die Indizes zwischen 11 und 36 (Jaime Gancedo, 2018).

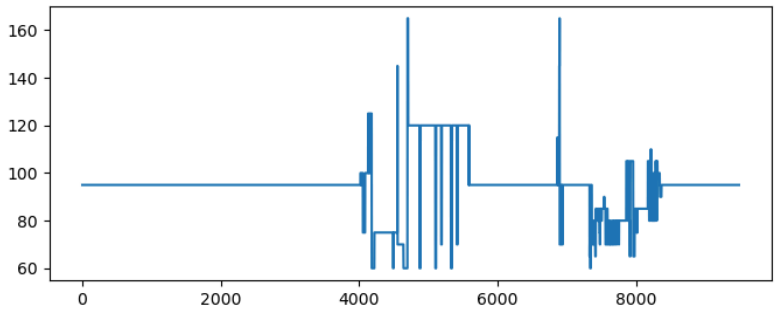

Abbildung 6. Verlauf der BPM des Songs: Madeon - The Prince

Die glatten Abschnitte im Ergebnisgraphen (vgl. Abb. 6) sind die Abschnitte des Musikstücks, an denen der Beat konsistent war. Die entstandenen ungleichmäßigen Abschnitte lassen sich direkt mit Sektionen im Musikstück in Verbindung bringen, bei denen entweder kein Beat vorhanden ist, oder dieser sich im Verlauf ändert. Damit die Animation durch diese Abschnitte nicht negativ beeinträchtigt wird, werden diese herausgefiltert. Dafür wird der längste zusammenhängende Abschnitt, für den der Beat korrekt bestimmt werden konnte, als Referenz betrachtet. Die zuvor und danach ermittelten Beats werden verworfen und die nun fehlenden Beats aus der BPM und dem ersten bzw. letzten Beat des Referenzabschnitts berechnet.

Da die FFT die Frequenzen des Eingabesignals „zerlegt“ beschreibt das erhaltene Ergebnis den Beat als wiederholende Frequenz. Der Phasenabstand definiert die Verschiebung dieser Frequenz und eignet sich daher, um den Zeitpunkt des Beats zu berechnen. Dies erfolgt mithilfe des atan2, mit dem realen und imaginären Teil der komplexen Zahl des Beats. Das Ergebnis ist dann die Phasenverschiebung als Radiant, welche mit der Frequenz verrechnet den zeitlichen Abstand zwischen dem Zeitpunkt der Berechnung und dem nächsten Beat ergibt.

Limitationen

Im Paper wird die BPM noch auf 0.0017Hz (0.1 BPM) präzisiert, dieser Schritt wurde aus Zeitgründen nicht mehr implementiert, sodass der Algorithmus auf 0,0833Hz (5 BPM) genau arbeitet. Das bedeutet, dass bei Songs, deren BPM zwischen einem 5er-Schritt liegt, die Beats nicht mehr genau getroffen werden. Das Tempo der Animation stimmt allerdings immer noch ungefähr überein.

Der Algorithmus wurde mit dem Datenset MUSDB18 (Rafii *u. a.*, 2019) getestet und hat dabei mit vielen Musikstücken nicht gut funktioniert. Das Datenset enthält allerdings wenig Musik aus den oben genannten Genres und eignete sich daher nicht gut zur Evaluation. Andere, besser geeignete Datensets konnten leider nicht gefunden werden.

Da die Präzision des Algorithmus von der Länge des betrachteten Zeitabschnitts abhängt, könnte dieser vergrößert werden. Das ist allerdings keine gute Idee, da für eine Genau-

igkeit von 1 BPM der Abschnitt dafür 60 Sekunden groß sein muss. Somit reagiert der Algorithmus dann wesentlich langsamer auf Veränderungen.

3.2.2. *Refrain Erkennung*

Diese Komponente nutzt das Verfahren aus (Goto, 2006), welches von (Jayaram, 2018a) als Bibliothek umgesetzt wurde, um Start und Ende von Vorkommnissen des Refrains im Song zu erkennen. Es nutzt aus, dass der Refrain in den meisten Songs der am häufigsten wiederholende Abschnitt ist.

Im ersten Schritt wird ein Überblick über die im Musikstück vorkommenden Noten zu verschiedenen Zeitpunkten verschafft. Es werden von 12 Noten die vorkommenden Frequenzen aus 6 Oktaven aufsummiert, so entsteht ein „Chromagram“ in dem jede Note je nachdem wie viel diese zu einem bestimmten Zeitpunkt präsent war, entsprechend heller oder dunkler dargestellt wird (Goto, 2006).
Das entstandene Chromagram wird dann in einem Zeit/Zeit Graphen gegenübergestellt. An übereinstimmenden Stellen entstehen dadurch dunkle Linien. Diese Linien sind paarweise Abschnitte im Musikstück, die sich wiederholen.
Da das Erkennen von diagonalen Linien aufwändig ist, wird der Graph mithilfe einer linearen Transformation in einen Time-Lag Graphen umgewandelt, sodass die Linien nun in der horizontalen verlaufen. Darauf wird dann ein Rauschentfernungsalgorithmus angewendet, wodurch der rechte Graph entsteht (Jayaram, 2018b).

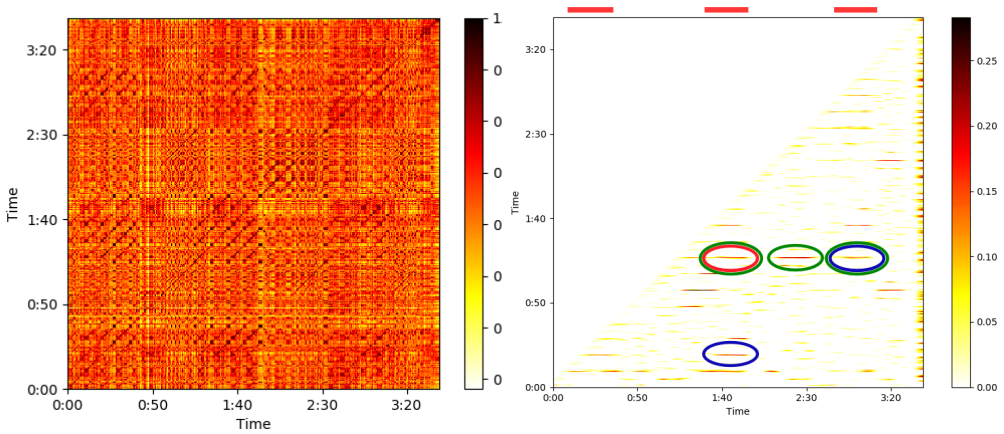

Abbildung 7. Zeit/Zeit Graph und Zeit/Lag Graph nach Rauschentfernung des Songs: Porter Robinson & Madeon - Shelter, erstellt mit PyChorus

Im Zeit/Lag Graphen werden dann zusammenhängende Linien ab einer festlegbaren Länge erkannt. Danach wird für jeden Abschnitt ein Score basierend auf der Anzahl an horizontal, sowie vertikal überlappenden Abschnitten berechnet. Der Abschnitt mit dem höchsten Score ist nun der wahrscheinlichste Refrain (Jayaram, 2018b).
Bis hier existiert der Code bereits, um nun mehr Vorkommnisse des Refrains zu erhalten. Die im Graph rot umrandete Linie ist das „passendste“ Segment, da sie mit den blau umrandeten Linien überlappt (vgl. Abb.7 rechts). Die weiteren Refrain-Segmente sind jetzt horizontal auf derselben Höhe zu finden (die grün umrandeten Linien).

Limitationen

Das Verfahren funktioniert nicht gut mit Songs, die nicht mit einem Metronom aufgenommen, bzw. am Computer produziert wurden, da in diesem Fall die Linien im Zeit/Lag

Graphen nicht mehr gerade sind, wodurch diese nicht mehr erkannt werden (Jayaram, 2018b). Aufgrund der gewählten Musikgenres stellt dies allerdings kein Problem dar.

3.2.3. *Einteilung des Songs in Abschnitte*

Die Refrain Erkennung teilt das Musikstück wie beschrieben in Abschnitte aus abwechselnd Refrain und kein Refrain.

Da die meisten Songs in ihrer Struktur meist einen von den restlichen Teilen des Songs unterschiedlichen Anfang, bzw. Ende haben, können zwei weitere Abschnitte definiert werden, indem der Abschnitt vor dem ersten Refrain als Intro und der Abschnitt nach dem letzten Refrain als Outro festgelegt wird. Dadurch entstehen mehr zum Song passende Abschnitte und somit mehr Animationswechsel, ohne das Musikstück tiefergehend analysieren zu müssen.

Abbildung 8. Beispielhafte Einteilung eines Songs

Damit die Übergänge zwischen den Abschnitten flüssig sind, werden Animationen immer abgeschlossen, bevor ein Wechsel stattfindet. Damit diese nahtlos ineinander übergehen, werden diese interpoliert.

3.2.4. *Berechnung der Bewegungswinkel*

Doggy2 kann neue Zielkoordinaten mit einer Rate von 50Hz erhalten. Ein Abtastratenzeitpunkt liegt dabei immer zwischen zwei Beats.

Abbildung 9. Beats (Rot) zwischen Abtastrate des Roboters (Schwarz)

Es wird daher der prozentuale Fortschritt zum Zeitpunkt der zu sendenden Koordinate berechnet, aus welchem dann die entsprechende Koordinate zum selben Fortschritt aus der definierten Animation errechnet wird. Mit dem bisher genutzten Verfahren zum Definieren der Animationen würde dies nicht ohne weiteres funktionieren. Daher wird im späteren Verlauf dieser Arbeit ein neues Verfahren eingeführt, welches zu einem prozentualen Animationsfortschritt die entsprechenden Roboterkoordinaten liefert.

3.3. Ein neues Tool zum definieren von Animationen

Wie bereits zuvor erwähnt konnten in vergangenen Arbeiten (Tzeng, 2013; Spillner, 2018; Weidenbach, 2019) zufriedenstellende Animationen durch Steuerung des Roboters mit einem Spielecontroller und Aufnahme dieser Bewegungen erreicht werden. Das Verfahren war auch durch den Programmcode von (Woerner, 2021) nutzbar. Dieses hier zum Erstellen der Tanzanimationen zu verwenden hätte allerdings zu Problemen geführt, da sich die Anforderungen im Vergleich zu den vorigen Animationen deutlich unterscheiden.

Die Aufnahme der Animationen fand über Sampling in festen Zeitabständen statt. Für eine Tanzanimation, welche je nach Tempo des Musikstücks schneller oder langsamer abgespielt wird, hätten Zwischenschritte interpoliert werden müssen, wobei die Qualität des Ergebnisses beim Erstellen der Animation, insbesondere im Hinblick auf die Variable Geschwindigkeit schwer abzuschätzen ist.

Außerdem bestand das Problem, dass durch die Steuerung via Controller teils ruckartige, robotisch wirkende Bewegungen entstanden sind, was in einigen Fällen eine Nachbearbeitung der Animationen notwendig machte (Weidenbach, 2019). Die Tatsache, dass in der Arbeit bereits angemerkt wurde, dass ein besseres Tool für die Erstellung der Animationen hilfreich wäre, unterstreicht den Nutzen ein solches Werkzeug zu erstellen.

Bezier Splines sind Freiformkurven, deren Verlauf mithilfe von Kontrollpunkten definiert werden kann. Sie sind je nach Berechnungsverfahren mathematisch leicht zu berechnen. Es gibt verschiedene Varianten von Splines, mit unterschiedlichen Eigenschaften. Hervorzuheben ist dabei die Eigenschaft der Kontinuität. Ein Spline kann Teil einer oder mehrerer Kontinuitätsklassen sein (Holmér, 2022).

Wenn alle Kontrollpunkte direkt in die Berechnung des Splines einbezogen werden, sorgt das dafür, dass sich Veränderungen der Position dieses Punktes auf die Form des gesamten Splines auswirken, wodurch das Definieren von spezifischen Formen erschwert wird. Um das zu vermeiden werden Splines in Segmente unterteilt, bzw. mehrere Kürzere Splines aneinander gehängt. Dazu werden meistens kubische Bezier Splines genutzt (Holmér, 2022).

Aufgrund dieser Eigenschaften sind Splines beispielsweise in der Spieleentwicklung zum Beschreiben von flüssigen Bewegungspfaden für Kamerafahrten, im Material Design zum Erstellen glatter Oberflächen und zum Beschreiben von Vektorgrafiken, wie etwa Schriftarten, weit verbreitet.

Die Kontinuitätsklasse C1 beschreibt die Kontinuität in der Veränderung der Geschwindigkeit der Bewegung und lässt sich daher nutzen, um zu garantieren, dass erstellte Animationen flüssig sind, d.h. keine abrupten Änderungen der Geschwindigkeit enthalten (Holmér, 2022). Ein einfacher Spline bestehend aus kubischen Beziers, bei dem Kontrollpunkte über Endpunkte der Segmente gespiegelt sind, erfüllt diese Eigenschaft, weshalb dies die Grundlage für das zu entwickelnde Softwarewerkzeug darstellt.

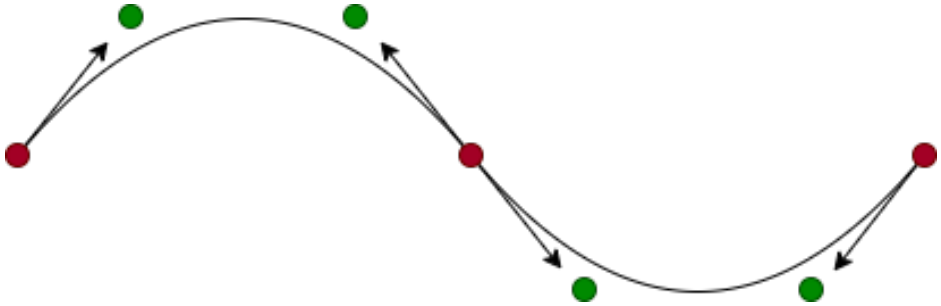

Abbildung 10. Veranschaulichung des Verwendeten Splines

In der Grafik (vgl. Abb. 10) Sind die Endpunkte der Spline Segmente Rot und die Kontrollpunkte Grün markiert. Um die Kontinuität zu gewährleisten, muss jeder Kontrollpunkt über den benachbarten Endpunkt gespiegelt sein. Damit die Kontinuität beim mehrfachen Abspielen derselben Animation nicht gebrochen wird, müssen die Kontrollpunkte auch an Start und Ende gespiegelt werden.

4. Integration in den Roboter (Implementierung)

Der bestehende Code von Doggy2 nutzt das „Robot Operating System“ (ROS), ein Framework zum Entwickeln von Robotersystemen. Es ist in der Forschung weit verbreitet. Es stellt im Kern eine Publisher/Subscriber Architektur zur Verfügung, über die „Knoten“ miteinander kommunizieren können, indem ein Knoten Nachrichten an ein „Topic“ sendet oder auf Nachrichten wartet. So ermöglicht ROS komplexe Robotersysteme zu entwickeln.

Die erwähnte Treibersoftware zum Ansteuern des Microcontrollers wurde mit ROS „Melodic“ umgesetzt, dies ist die vorletzte ROS Distribution und erhält bereits keine Updates mehr. Die neuere und letzte Distribution „Noetic“ wird noch bis Mitte 2025 unterstützt. Daher wurde über eine Portierung des Projektes auf ROS2 nachgedacht, da ROS2 Distributionen auch darüber hinaus aktualisiert werden. Aus Zeitgründen wurde dies allerdings wieder verworfen.
Das bestehende Projekte musste zunächst von ROS „Melodic“ auf „Noetic“ portiert werden. Das war notwendig, da insbesondere die „PyChorus“ Bibliothek Abhängigkeiten zu Paketen hat, die erst mit Python 3 zur Verfügung stehen. ROS Melodic wird allerdings mit Python 2 ausgeliefert. Um die Steuerungssoftware (Woerner, 2021) weiterhin verwenden zu können, mussten einige QT4 Abhängigkeiten manuell hinzugefügt werden, da diese in der neueren Ubuntu Distribution 20.04 nicht mehr in den offiziellen Repositories zur Verfügung standen.
Über das Topic `MicroconRequest` ist es nun möglich dem Roboter Zielkoordinaten zu senden. Da die Tanzbewegung vor der Ausführung vollständig berechnet wird, wurde das System innerhalb eines einzigen Python Skripts umgesetzt. Die im folgenden genauer beschriebenen Komponenten für Beat und Refrain Erkennung werden über Funktionen aufgerufen und die Ergebnisse in ein Array aus Koordinaten zum Senden an den Roboter umgewandelt.
Das Tool zum Erstellen der Animationen, sowie die Simulation sind in eigenen ROS Paketen zu finden.

4.1. Implementation Beat Erkennung

Die Entwicklung des Beat-Erkennungs-Algorithmus fand zuerst statt. Da zu Beginn noch nicht ausgeschlossen war den gewählten Algorithmus in Echtzeit zu nutzen, wurde Rust als Programmiersprache aufgrund der hohen Geschwindigkeit und Speichersicherheit (Bugden und Alahmar, 2022) gewählt. Für ROS2 wären auch entsprechende Bibliotheken zum Erstellen von ROS Knoten in Rust vorhanden gewesen.
Später stellte sich heraus, dass das bestehende Projekt noch ROS1 nutzte. Hier stellt sich die Integration mit Rust komplizierter dar, da diese nur über die „ROS Bridge“, ein separat zu startender Server, welcher die Nachrichten weiterleitet, möglich gewesen wäre. Da die Berechnung dann auch asynchron erfolgen sollte, war die hohe Performanz von Rust nicht mehr zwingend notwendig. Um die weitere Arbeit am Projekt durch Einführung einer neuen Programmiersprache nicht noch schwieriger zu gestalten, wurde schließlich die Entscheidung getroffen, den Algorithmus in Python neu zu schreiben.

Der Algorithmus benötigt eine Integer Wave Audiodatei in 44100 Hz Sampling Rate als Eingabe. Mit den in Kapitel 3 beschriebenen Schritten berechnet dieser Teil der Implementierung die Position aller Beats und speichert diese in einem Array, welches dann für weitere Schritte zur Verfügung steht.

4.2. **Implementation Refrain Erkennung**

Zur Chorus Detection existierte bereits die Python Bibliothek PyChorus (Jayaram, 2018a). Diese implementiert das beschriebene Verfahren aus (Goto, 2006), gibt allerdings nur einen Chorus mit der höchsten Übereinstimmung aus. Die Bibliothek dient als Basis und wird so erweitert, dass am besten alle, zumindest aber möglichst viele Vorkommnisse des Refrains ausgegeben werden.

Die PyChorus Bibliothek funktioniert grob wie folgt:
Zu Beginn wird mithilfe der `librosa` Bibliothek ein Chromagram erstellt. Aus diesem wird dann die TimeTimeMatrix erzeugt, welche dann in eine TimeLagMatrix umgewandelt wird. Aus dieser wird dann das Rauschen entfernt. Nun können Linien erkannt werden, welche dann die Refrain Abschnitte beschreiben. Das passiert, indem `local_maxima_rows` die „dunklen Stellen“ im Graphen bestimmt, diese werden dann von `detect_lines` zu Linien zusammengefügt. Den so erkannten Linien wird dann in `count_overlapping_lines` nach der Anzahl übereinstimmender Vorkommnisse ein Score zugewiesen. `best segment` liefert dann den Abschnitt mit dem besten Score als wahrscheinlichsten Refrain zurück.

Es wird daher in die Methode `best_segment` eingegriffen, damit diese statt einem einzigen Auftreten des Refrains alle gefundenen Refrain Abschnitte zurückliefert. Dazu wird der bereits bestimmte wahrscheinlichste Refrain Abschnitt als Referenz betrachtet. Der Lag Wert einer Line gibt die vertikale Position der Linie im Graphen an. Es werden dann alle weiteren Linien mit identischem Lag Wert in die Ergebnisse aufgenommen. Da eine Linie immer die zweite Stelle von paarweise gleichen Stellen im Song repräsentiert, wird zur zeitlich zuerst gefundenen Linie aus den Ergebnissen eine neue Linie mit gleicher Länge erzeugt, bei der der Startpunkt der Lag Wert ist. Da sich diese beiden Abschnitte überlagern, werden diese zu einem Abschnitt zusammengefügt.

4.3. Implementation des Animationstools

Das Tool zum Erstellen der Animationen wurde als GUI Anwendung mit der C++ Variante des GUI Frameworks QT5 umgesetzt. Somit ist eine Integration mit ROS problemlos möglich. Die Animationen können so direkt aus der GUI in der Simulation oder direkt am Roboter getestet werden. Die Nutzeroberfläche wurde mithilfe von QT Designer 5 definiert. Der Animationseditor selbst ist als Widget in QT umgesetzt und nutzt die Bibliothek SFML, welche es mit wenig Programmieraufwand ermöglicht, einfache geometrische Formen in einem Fenster zu zeichnen.

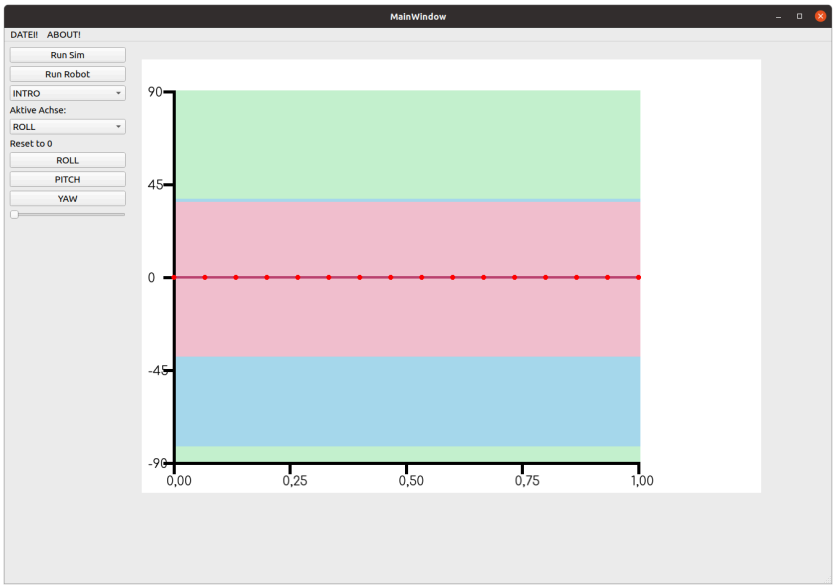

Abbildung 11. Der Animationseditor

Im Editor (vgl Abb. 11) sind drei Linien zu sehen, die jeweils den Gradwert der drei Bewegungsachsen von Doggy2 im Verlauf einer Zeiteinheit repräsentieren. Zu Beginn sind alle Achsen übereinander angeordnet, da alle auf 0 stehen. Die intensiver gefärbten Punkte stellen Kontrollpunkte der Splines dar. Durch Klicken und Ziehen kann die Position des Roboters zum Zeitpunkt des bearbeiteten Kontrollpunkts angepasst werden. Über das Drop-down-Menü **„Aktive Achse“** kann beschränkt werden, welche Achse manipuliert werden kann. Ohne diese Funktion könnte, wenn zwei oder mehr Punkte direkt übereinander liegen, nur der in der Darstellungshierarchie oberste Punkt ausgewählt und bewegt werden. Die farbigen Bereiche stellen jeweils den maximalen Bewegungsradius der jeweils gleich gefärbten Achse dar.

Ist eine Animation fertiggestellt, kann diese über **Datei -> Speichern** in einer Datei im JSON Format gespeichert werden. Dabei werden nur die Punkte der Animation gespeichert.

Der Animationseditor besteht aus 3 Klassen:

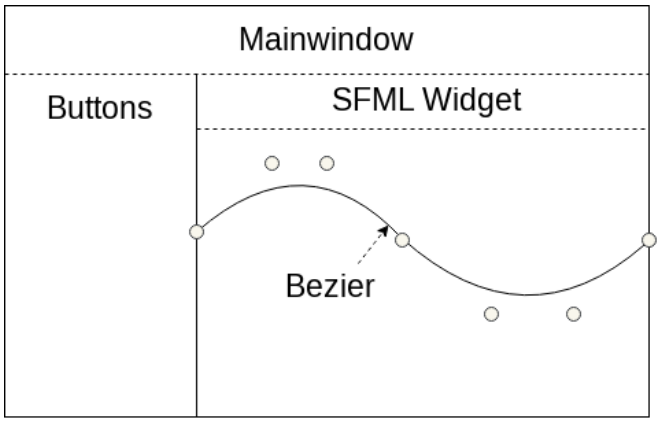

Abbildung 12. Klassen des Animationseditors

In der **Mainwindow** Klasse werden die QT Buttons (vgl Abb. 12, positioniert im linken Bereich) mit Funktionen verbunden, um Beispielsweise das Testen der Animation im Simulator zu ermöglichen.

Die **SFMLWidget** Klasse bindet die Bibliothek SFML2 ein und übernimmt die Darstellung des Graphen des Animationseditors. Hier sind außerdem alle Logiken implementiert, die mit der Manipulation des Graphen zu tun haben, auf welche später noch genauer eingegangen wird.

Die **Bezier** Klasse beschreibt eine Bewegung in 3D innerhalb einer Zeiteinheit t, welche aus beliebig vielen Kubischen Bezier Splines bestehen kann. Im Array **points** sind alle Punkte der Kurve gespeichert. Ein Punkt ist ein Objekt aus einer 3D-Koordinate und jeweils einer SFML CircleShape für jede Achse.

Weiterhin wurden in der Bezier Klasse drei Funktionen implementiert, welche die Darstellung der Splines ermöglichen. Jede dargestellte Kurve besteht dabei aus hunderten Punkten, die dann zu einer Kurve „verschwimmen“. Berechnet werden diese mit `lerp` , `sampleAt` und `sampleAtCubic`. `lerp` berechnet die lineare Interpolation zwischen zwei dreidimensionalen Punkten abhängig vom Wert `t`. Für jede Dimension wird folgende Formel angewendet:

$$d_{\text{interp}} = d_{\text{p1}} + \left(\left(\frac{d_{\text{p2}} - d_{\text{p1}}}{100} \right) * t \right)$$

`sampleAt` ist für die Berechnung der Position eines Samples zum Zeitpunkt t
Innerhalb von `sampleAt` wird bestimmt, in welchem Spline Segment sich der eingegebene t Wert befindet. Es wird dann ein neues Spline Objekt mit den 4 Punkten des Segments erzeugt. Auf diesem wird dann `sampleAtCubic` aufgerufen.
`sampleAtCubic` implementiert DeCasteljaus Algorithmus um die interpolierte Positon des Samples P zum Zeitpunkt t bestimmen, die Formel dafür sieht bei einem kubischen Bezier so aus (übernommen aus (Holmér, 2022)):

$$\begin{aligned} P &= \operatorname{lerp}(d, e, t) \\ e &= \operatorname{lerp}(c, d, t) \\ d &= \operatorname{lerp}(a, b, t) \\ c &= \operatorname{lerp}(p_2, p_3, t) \\ b &= \operatorname{lerp}(p_1, p_2, t) \\ a &= \operatorname{lerp}(p_0, p_1, t) \end{aligned}$$

Da die visuellen Kurven der Splines nun berechnet werden können, wird nun näher auf die Logik zum Verschieben der Kontrollpunkte eingegangen. Dies erforderte einiges an Programmieraufwand, um die Spiegelung der Kontrollpunkte und somit C1 Kontinuität der Splines zu gewährleisten

Handelt es sich beim angeklickten Punkt um einen Kontrollpunkt, so muss der gegenüberliegende Kontrollpunkt im nächsten Spline Segment in die entgegengesetzte Richtung bewegt werden. Ist es ein Endpunkt, so müssen die beiden benachbarten Kontrollpunkt mitbewegt werden. Bei Kontrollpunkten und Endpunkten am Anfang, bzw. Ende der Animation muss jeweils der Kontroll bzw. Endpunkt am anderen Ende des Graphen beeinflusst werden, damit die Animation die C1 Kontinuität auch beim wiederholten Abspielen beibehält.
Das Bewegen der Punkte wird ermöglicht, indem im `onMouseClickEvent` festgestellt wird, ob sich einer der Punkte an der angeklickten Position befindet. Es wird dann ein Pointer auf diesen und alle relevanten Nachbarpunkte gespeichert. Wenn mindestens ein Pointer gesetzt ist, folgen im `onMouseMoveEvent` die gespeicherten Punkte dann der Mausbewegung oder bei Nachbarpunkten der passenden Gegenbewegung, bis das `onMouseReleaseEvent` ausgelöst wird. Dieses entfernt dann alle gesetzten Pointer wieder.

Zum Speichern der Punkte müssen diese von den Bildschirmkoordinaten in die Gradwerte für Doggy2 umgerechnet werden, da diese nicht übereinstimmen. Der Graph innerhalb des Editors ist mit einer festen Größe definiert, d.h. er passt sich nicht der Größe des Fensters an, da dies einen hohen Entwicklungsaufwand bedeutet hätte. Daher kann die Übersetzung der Bildschirmkoordinaten mit einer hartkodierten Formel erfolgen.

Die Buttons „Reset to 0“ für die 3 Achsen, sowie der Button „Run on Robot“ haben aus Zeitgründen keine Funktionalität mehr erhalten.

Performance Problem durch anfänglich gewählte Datenstruktur
Zu Beginn wurde statt der reinen 2D Koordinaten jeweils ein SFML Circle Objekt gespeichert, da so die vom Nutzer manipulierbaren Kontrollpunkte nicht separat hätten gespeichert werden müssen. Durch die rekursive Berechnung der Kurven entstand so ein Performanceproblem: Ab ca. 1600 Samples pro Linie einer Achse beanspruchte die Anwendung die CPU stark und die Bildrate sank auf ca. 15 Bilder pro Sekunde, was sich durch Ruckeln beim Verschieben von Kontrollpunkten bemerkbar machte.
Das Problem wurde von den SFML Circle Objekten ausgelöst, da die Erzeugung eines neuen Circle Objekts einige rechenintensive Prozesse mit sich zieht und durch die rekursive Berechnung pro Sample $9 * 3$ neue Circle Objekte pro Frame erzeugt werden, die allerdings nie dargestellt werden müssen.
Gelöst wurde dies durch Erstellen einer neuen Bezier Klasse für die rekursiven Berechnungen. Diese speichert jeden Punkt als reine 3D Koordinate und spart so die nicht notwendigen Prozesse ein.

4.4. Erstellung der Animationen

Es stellte sich in der Praxis als sehr herausfordernd heraus, Animationen zu erstellen, die interessant, unterschiedlich und nicht zu schnell für die Roboterhardware sind. Zu Beginn wurden die Animationen auf den jeweils nächsten Beat abgestimmt, wodurch die Bewegungen bei höherer BPM (ca. 90) bereits teils abgehackt wirkten und die Gelenke und Motoren des Roboters stark beanspruchten. Somit wurden Animationen später für zwei Beat Intervalle definiert.

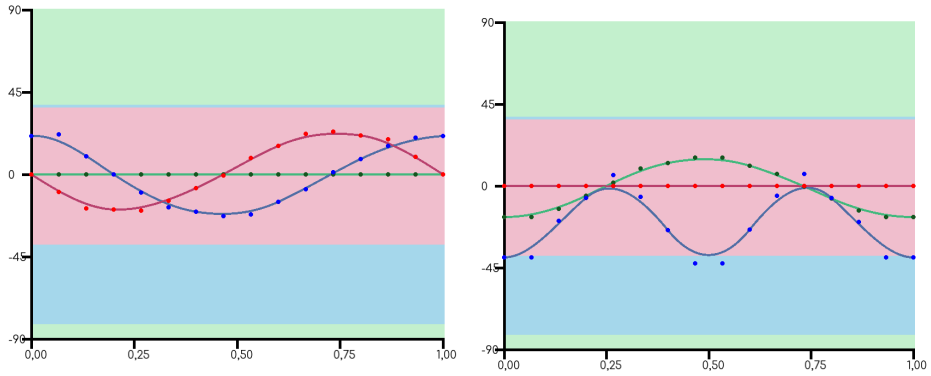

Abbildung 13. Animationsgraphen: Links Intro, Rechts Chorus

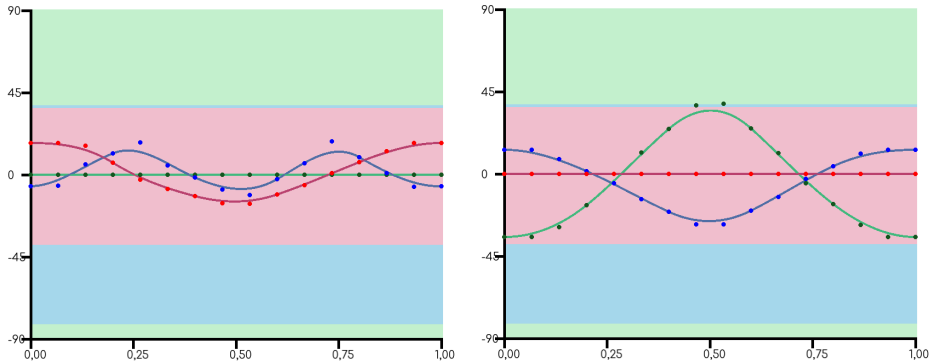

Abbildung 14. Animationsgraphen: Links Other, Rechts Outro

Die Linien der verschiedenen Achsen der erstellten Animationen ähneln Sinuskurven stark, weil diese in der Praxis am besten funktionierten. Somit stellt sich die Frage ob die Entwicklung des Animationstools überhaupt nötig gewesen wäre. Das entwickelte Animationstool ermöglichte allerdings erst das herauszufinden, da verschiedene Animationen schnell erstellt und getestet werden konnten. Außerdem besteht in Zukunft die Möglichkeit, die Animationen für längere Beat Intervalle zu definieren, wodurch die Möglichkeiten des Editors besser ausgeschöpft werden können und gleichzeitig interessantere Animationen entstehen.

4.5. Generierung und Abspielen der Tanzanimation

Die Ergebnisse der Beat und Refrain-Erkennung werden dann so zusammengefügt, dass bei einem Abschnittsübergang die zuletzt abgespielte Animation zu Ende läuft. Die Animationen werden aus in einem Verzeichnis gespeicherten JSON Dateien eingelesen. Daraus entsteht dann ein Array aus den Zielkoordinaten des Roboters, während der Song spielt, gesamplet in 50Hz.

Nachdem ein Tanz mithilfe der vorherigen Komponenten erstellt wurde, kann dieser nun synchron mit dem Musikstück abgespielt werden. Dies wurde mithilfe der Python Bibliothek PyAudio implementiert.

PyAudio bietet die Option, die Audiowiedergabe mit einem Callback nach einer selbst definierbaren Anzahl von abgespielten Samples durchzuführen. Es wurde dazu 882 als Größe gewählt, da dies bei 44100 Hz der Eingabedatei genau 50 Hz und somit der Samplingrate des Roboters entspricht.

4.6. Simulation

Um das System und die erstellten Animationen auch unabhängig vom echten Roboter testen zu können, wurde dieser mithilfe von RVIZ simuliert. RVIZ ist Bestandteil von ROS und daher sehr gut in dieses integriert. Daher eignet sich das Programm gut zur Simulation des Roboters.

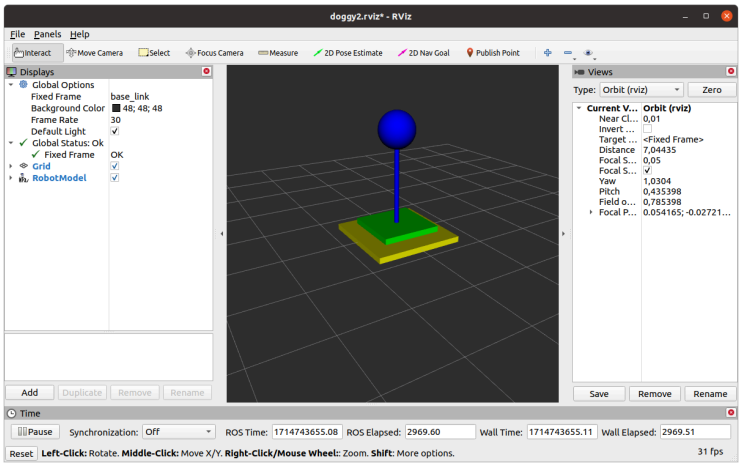

Abbildung 15. Doggy2 simuliert in RVIZ

Eine RVIZ Simulation wird dabei durch eine XML-Datei mit der Endung `.urdf` definiert. In dieser wird der Roboter mithilfe von **„links“** und **„joints“** modelliert.
Ein **link** repräsentiert ein Teil des Roboters, wie beispielsweise die Kugel, welche den Kopf darstellt. Innerhalb eines Links kann für diesen eine visuelle Repräsentation durch Beschreibung einer 3D-Form definiert werden. Die **links** werden dann durch **joints** miteinander verbunden. Diese repräsentieren die möglichen Bewegungsachsen. Für einen **joint** kann z.B. der mögliche Bewegungsradius auf die Werte des echten Roboters beschränkt werden.

Durch senden von ROS Nachrichten auf das topic `joint_states` kann das Modell bewegt werden. Dies funktioniert nach demselben Prinzip wie auch beim echten Roboter, sodass der mit der Simulation getestete Code gut auf den echten Roboter übertragbar ist.

4.7. **Reparatur des Roboters wegen Sensor Drift**

Beim Testen der Animationen fiel auf, dass sich der Nullpunkt des Roboters auf der X-Achse nach jeder Bewegung um ca. 10° nach rechts verschob. Es wurde vermutet, dass der entsprechende Sensor kaputtgegangen war. Zum Glück war ein passendes Ersatzteil in der Arbeitsgruppe vorhanden und konnte nach etwas Feinarbeit erfolgreich am Roboter angebracht werden. Somit funktionierte der Roboter wieder einwandfrei.

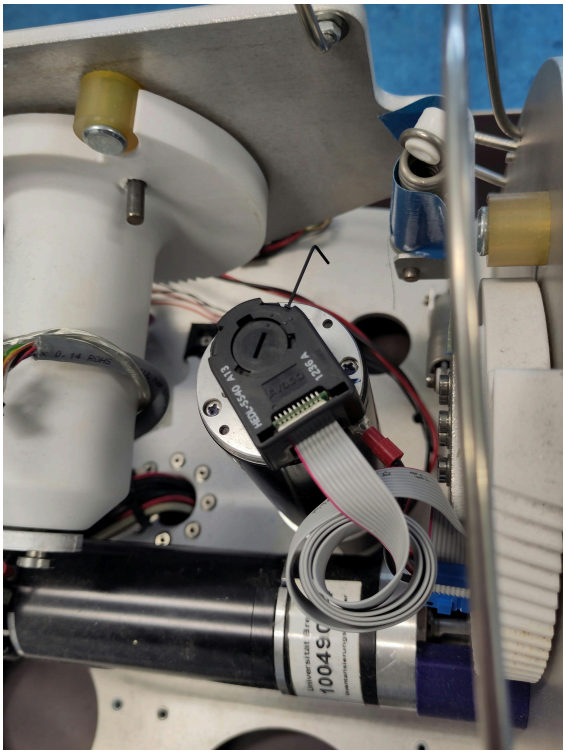

Abbildung 16. Neuer Sensor während des Anbringens

5. Nutzerstudie

Zur Evaluation des Systems wurde eine Studie geplant und durchgeführt.

5.1. Studiendesign

Als Ort zur Durchführung der Studie wurde ein Bereich vor dem Eingang eines zentral gelegenen Universitätsgebäudes (Mehrzweckhochhaus, kurz MZH) gewählt, da hier viele Menschen vorbeikommen und die notwendige Stromversorgung gegeben war. Geplant war die Studie in der Mittagszeit über einen Zeitraum von ca. 4-5 Stunden durchzuführen um ausreichend Daten sammeln zu können.

Um zu überprüfen, ob Doggy2 mit Tanzanimation mehr Aufmerksamkeit auf sich zieht, werden in Zeitblöcken von jeweils 15 Minuten alle vorbeikommenden Personen gezählt. Dabei wird in 2 Gruppen unterteilt: **Gruppe 1**: Roboter steht still; **Gruppe 2**: Roboter tanzt. Es wird in jedem Zeitblock Musik abgespielt, um Musik als beeinflussenden Faktor auszuschließen. Währenddessen werden in beiden Fällen alle vorbeikommenden Personen gezählt, sowie wie viele dieser Personen stehen bleiben, um sich den Roboter anzuschauen.
Die Demographie der Teilnehmer wurde nicht erfasst. Aufgrund des gewählten Durchführungsortes besteht die Demographie überwiegend aus Studenten, sowie Mitarbeitern der Universität. Im MZH sind die Informatik-Fakultäten der Universität untergebracht, sodass ein großer Teil der Personen vermutlich aus Mitgliedern des Fachbereichs Informatik besteht. In vergangenen Studien reagierten insbesondere Kinder sehr positiv auf Doggy (Weidenbach, 2019), da diese hier nicht anwesend waren, beeinflusst dies das Ergebnis vermutlich negativ.

Zusätzlich dazu soll mit einem kurzen Fragebogen die Qualität der Bewegung und die Wirkung des Roboters während des Tanzens evaluiert werden. Die Fragen basieren auf dem von Bartneck stammenden „Godspeed Questionaire“, welches eine standardisierte Evaluation von Mensch Roboter Interaktion ermöglichen soll (Bartneck *u. a.*, 2009), da „bereits kleine Unterschiede in der Wortwahl von Fragestellungen zu unterschiedlichen Ergebnissen führen können“ (Bartneck, 2023). Weidenbach nutzte in seiner Studie bereits ausgewählte Attribute aus dem „Godspeed Questionaire“, sodass Vergleichswerte vorliegen.
Dieser Fragebogen wird nur für **Gruppe 2** angeboten, da ein Vergleich mit dem sonst statischen Roboter aus meiner Sicht nicht sinnvoll erscheint. Aufgrund der Situation der Studie muss der Fragebogen schnell beantwortbar sein, da bei einem längeren Fragebogen die Bereitschaft diesen zu beantworten schnell sinkt. Es werden daher nur die mir am wichtigsten und passendsten erscheinenden Kategorien abgefragt. Außerdem sollte das Attribut bereits von (Weidenbach, 2019) abgefragt worden sein, damit ein Vergleich möglich ist. Die Entscheidung fiel auf folgende Kategorien:

Attribut	Von … Bis						
Lebendigkeit	künstlich	1	2	3	4	5	realistisch
Bewegung	bewegt sich steif	1	2	3	4	5	bewegt sich flüssig
Kompetenz	Inkompetent	1	2	3	4	5	Kompetent

Tabelle 1. Ausgewählte Attribute für den Fragebogen

Das Attribut **Lebendigkeit** soll Aufschluss über die generelle Wirkung von Doggy geben. **Bewegung** ist insbesondere deswegen interessant, da der Animationseditor mit dem Ziel programmiert wurde, flüssige Bewegungen einfach erstellen zu können. Das Attribut

Kompetenz wurde gewählt, um zu evaluieren, ob Betrachter Doggy2 als „guten Tänzer“ wahrnehmen.
Weiterhin soll mit der Frage: „Welche Aktivität würdest du dem Roboter zuschreiben?“ überprüft werden, ob die Bewegung überhaupt als Tanz wahrgenommen wird.
Der Fragebogen enthält außerdem ein Feld für freie Kommentare, um über die genannten Fragen hinaus Beobachtungen der Zuschauer zu erfassen.

5.2. **Durchführung**

Abbildung 17. Aufbau der Studie

5.2.1. *Studienaufbau*

Zum Beantworten des Fragebogens wurde dieser in Google Forms erstellt und an einem seperaten Laptop zum Ausfüllen bereitgestellt. Da das Zählen verschiedener Personengruppen, sowie die Überwachung des Roboters für eine Person gleichzeitig nur schwer möglich ist, fragte ich zwei Komillitonen um Hilfe. Einer zählte dabei alle vorbeigehenden Personen, ein weiterer alle stehenbleibenden Personen, die dritte Person konnte sich um den Roboter kümmern.

5.2.2. *Sicherheitsvorkehrungen*

Um den Roboter wurde ein Absperrband platziert, um zu verhindern, dass Menschen in den Bewegungsradius von Doggy2 gehen. Dies verhindert auch, dass Personen versehentlich in das Getriebe greifen, was zuvor durch das Kostüm verhindert wurde. Zu diesem Zweck wurden auch Warnsymbole an der Bodenplatte des Roboters angebracht.

5.2.3. *Limitationen und Probleme*

Für die Studie wurden Songs ausgewählt, bei denen die BPM des Songs mit der Präzision des Beat Detection Algorithmus übereinstimmt. Somit stellt sie einen Best-Case dar, was diese Komponente angeht.
Der Start der Studie verschob sich um ca. 2 Stunden, da ein geeignetes Transportmittel, um Doggy2 aus dem Cartesium zum MZH zu bringen, gefunden werden musste.
Weiterhin hatte Doggy2 beim Versuch den ersten Zeitblock mit Tanz zu starten, eine Fehlfunktion und ließ sich nicht mehr ansteuern, wodurch weitere Verzögerungen entstanden sind. So konnten nur 4 Blöcke zu je 15 Minuten durchgeführt werden. Der Roboter war ca. 30 Minuten aktiv.

Leider trat auch das Sensor Drift Problem 2 Tage vor Studientermin abgeschwächt an der Z-Achse von Doggy2 auf. Es war kein weiteres Ersatzteil vorhanden und eine Neubestellung hätte zu lange gedauert. Da das Problem nicht so ausgeprägt wie zuvor bei der X-Achse auftrat, konnte zumindest ein Song mit Tanz abgespielt werden, bevor der Anschlag der Z-Achse in der Bewegung erreicht wurde. Der Roboter musste somit allerdings nach jedem Song zurückgesetzt werden, was kurze Pausen zwischen den Songs notwendig machte und die Durchführung der Studie weiter erschwerte.

5.3. **Ergebnisse**

Die genannten Probleme mit der Roboterhardware während der Studiendurchführung mindern die Aussagekraft der Ergebnisse, da weniger Zeitblöcke als ursprünglich geplant durchgeführt werden konnten. In der Ergebnistabelle steht das **S** für Statisch und das **T** für Tanz, um die verschiedenen Testgruppen zu markieren.

Zeitslot	**Gezählte Personen**	**Davon Stehengeblieben**	**Anteil**
1 **S**	221	7	3.17%
2 **T**	162	16	9.88%
3 **S**	202	4	1.98%
4 **T**	88	3	3,41%
Summe **S**	423	11	2.60%
Summe **T**	250	19	7.60%

Tabelle 2. Studienergebnisse

Um die Aussagekraft der Ergebnisse bewerten zu können, wird nun ein rechtsseitiger Signifikanztest durchgeführt. Dabei stellen die statischen Zeitblöcke die Nullhypothese dar. Als Signifikanzniveau wurde 1% gewählt, in der Wissenschaft werden 5% als Standardwert betrachtet. Da der Anteil an Personen, die stehengeblieben sind, in den Zeitblöcken mit statischem Roboter sehr gering war, wurde das Signifikanzniveau und damit die Fehlerwahrscheinlichkeit niedriger angesetzt. Zwischen den Zeitslots 2T und 3S fand eine Pause statt, daher wird der Signifikanztest zunächst mit 1S und 2T, darauf mit 3S und 4T und zum Schluss mit der Summe der S Slots und der Summe der T Slots durchgeführt.

Zeitslots	H_0	**Grenze**	**Ergebnis**	**Signifikant?**
1 S - 2 T	3.17%	11	16	Ja
3 S - 4 T	1.98%	5	3	Nein
Summe S - Summe T	2.60%	13	19	Ja

Tabelle 3. Rechtsseitiger Signifikanztest

Da der Anteil an Personen, die stehengeblieben sind, in Slot 2 vergleichsweise hoch war, sorgt das dafür, dass auch der summierte Test signifikant ist. Es bleibt daher aufgrund der wenigen Zeitslots unklar, ob es sich beim Ergebnis aus Slot 2 um einen Ausreißer handelt.

Überraschend war für mich, wie wenig Interesse vorbeigehende Personen im Allgemeinen am Versuchsaufbau zeigten. Dies lässt sich zumindest zum Teil durch den Ort erklären. Viele der Personen sind vermutlich auf dem Weg zu einem anderen Universitätsgebäude zu Veranstaltungen, etc. gewesen und hatten daher vermutlich ein generell niedriges Interesse an ihrer Umgebung. Es könnte daher sein, dass die Ergebnisse auf einer Veranstaltung tendenziell etwas besser ausfallen, da dort Teilnehmer normalerweise Zeit

mitbringen und sich aktiv interessante Dinge anschauen wollen. Diese Vermutung müsste allerdings überprüft werden. Eine andere Erklärung könnte das Fehlen des Kostüms sein. Doggy2 hätte mit Kostüm vielleicht mehr Aufmerksamkeit auf sich gezogen.

Proband	**Lebendigkeit**	**Bewegung**	**Kompetenz**	**Aktivität**
1	1	4	2	Tanzen
2	2	5	4	Tanzen
3	1	4	2	Tanzübungen
4	3	4	4	Tanzen, Animierend/Stimmungsmachend
5	4	5	5	Tanzen
6	2	2	3	Tanzen
7	2	2	2	Tanzen
8	4	4	4	Punchingball/Ausweichtraining/Reaktionstest
9	1	2	5	bouncing
10	4	5	5	Sich im Rythmus bewegen
	2,4	3,7	3,6	

Tabelle 4. Ergebnisse des Fragebogens

Das „Godspeed Questionaire“ ist zwar standardisiert, ein Vergleich zu anderen Robotern, die auch mit diesem evaluiert wurden, war allerdings nicht möglich, da spezifisch tanzende Roboter die Auswahl bereits stark einschränken und bei diesen nur die Durchschnittswerte der übergeordneten Kategorien einsehbar waren. Ein Beispiel dafür ist (Berl, Pakrasi und LaViers, 2019). Daher werden die Ergebnisse dieser Studie mit den Ergebnissen von (Weidenbach, 2019) verglichen, um diese immerhin im Kontext von Doggy einordnen zu können.
Da nur 10 Beobachter den Fragebogen ausfüllten, ist die Aussagekraft der Ergebnisse begrenzt. Es lassen sich dennoch Tendenzen erkennen.

	Lebendigkeit	**Bewegung**	**Kompetenz**
Ergebnis Weidenbach	2,8	3,5	/
Ergebnis Studie	2,4	3,7	3,6

Tabelle 5. Vergleich mit einzelnen Ergebnissen der Studie von Weidenbach

Es ist sichtbar, dass die Ergebnisse in den Kategorien nah beieinander liegen. Dass **Lebendigkeit** hier etwas niedriger ausfällt, könnte durch das fehlende Kostüm erklärt werden, da so klar erkennbar ist, dass es sich um einen Roboter handelt.

Bewegung wurde ein wenig besser bewertet, was dafür spricht, dass die mit dem Animationseditor erzeugten Bewegungen ähnlich flüssig wahrgenommen werden, wie die in der Vergangenheit erstellten. Durch den Editor konnten die Animationen allerdings ohne

Nachbearbeitung und in sehr kurzer Zeit erzeugt werden, sodass sich die Entwicklung dieses Werkzeugs gelohnt hat.

Da es keinen Vergleichswert für **Kompetenz** gibt, lässt sich hier nur anmerken, dass das Ergebnis nahe bei dem für **Bewegung** liegt, was darauf hindeutet, dass die Tanzfähigkeiten des Roboters positiv eingeschätzt wurden.
8 von 10 Personen haben die Bewegung als Tanz wahrgenommen. Durch die spielende Musik liegt die Assoziation zwar nahe, dennoch ist es positiv zu beobachten, dass die Animationen größtenteils korrekt wahrgenommen werden.

Die Antworten aus dem Feld für freie Kommentare unterstützen, dass Doggy2 generell positiv ankam. Trotz des fehlenden Kostüms wurde der Roboter als „Super cute“ bezeichnet. Weitere Kommentare wie „Tanzt besser als manche Leute in lokalen Clubs“ und „Die Beats werden sehr gut getroffen“ unterstützen, dass Doggys Tanz positiv wahrgenommen wurde.

6. Fazit und Ausblick

Insgesamt lässt sich erkennen, dass sich die Integration einer Tanzanimation tendenziell positiv auf die Anziehungskraft und Wahrnehmung von Doggy2 auswirkt. Um darüber eine klarere Aussage treffen zu können, ist allerdings weitere Forschung notwendig. Es sind weitere Schritte nötig, um das System im Gesamtkontext vom restlichen Verhalten von Doggy evaluieren und nutzen zu können. Dazu muss dieses zunächst auf der neuen Roboterhardware lauffähig gemacht werden. Danach wird die Integration des hier beschriebenen Systems Änderungen am bestehenden Zustandsautomaten notwendig machen. Insbesondere der Übergang zwischen Tanzanimation und dem Ballspiel, bzw. den Emotionen wird dabei eine Herausforderung darstellen.

Da viele technische Probleme gelöst und Grundlagen zur Integration von Tanzanimationen in Doggys Verhalten geschaffen werden mussten, blieb keine Zeit mehr, um zu erforschen, wie sich verschiedene Varianten von Tanzanimationen auf Doggys Anziehungskraft und Wahrnehmung durch Beobachter auswirken. Dies könnte in Zukunft genauer betrachtet werden.

Aufgrund des zeitlichen Umfangs wurden bei den Beat und Refrain-Erkennungs-Komponenten Kompromisse gemacht, was die praktische Nutzbarkeit angeht. Hier gibt es Verbesserungspotential. Für den praktischen Einsatz ist ein Echtzeitalgorithmus zur Beat-Erkennung deutlich besser geeignet, da das Einspielen von Musik über den mit Doggy verbundenen Computer je nach Gegebenheiten der Veranstaltung beschränkt sein kann. Auch das Einlesen der Dateien macht das System unfreundlich für Nutzer, da die Musikstücke im Integer Wave Format mit 44100 Hz Samplingrate vorliegen müssen, damit der Beat-Erkennungs-Algorithmus ein brauchbares Ergebnis liefert. Dies bereitete mir auch beim Testen des Systems Probleme, da Musik im passenden Format und in guter Qualität schwer zu finden war. Der Algorithmus konnte so nur mit einer stark begrenzten Anzahl an Songs getestet werden.
Einen in Echtzeit arbeitenden Algorithmus zu verwenden ist daher notwendig, um das System außerhalb des Forschungskontexts praktikabel einsetzen zu können. Dabei könnte auch die Präzision des Algorithmus verbessert werden.

Eine Refrain-Erkennung zu nutzen, um passend zur Musik die Animation zu wechseln, hat in der Praxis nur mäßig funktioniert. Obwohl bei fast allen genutzten Songs mindestens ein Refrain erkannt wurde, waren diese nicht immer gleichmäßig im Song verteilt, sodass Abschnitte mit derselben Animation teils sehr lange abgespielt wurden. Während der Durchführung der Studie fiel dies besonders auf. Für einen vorbeigehenden Beobachter, der dem Roboter vielleicht nur wenige Sekunden Aufmerksamkeit schenkt, dauert das zu lange. Dazu kommt, dass der Algorithmus nicht in Echtzeit funktioniert, was mögliche Verbesserungen an anderen Komponenten des Systems dahingehend einschränkt.
Hier lohnt es sich nach einem neuen Ansatz, wie beispielsweise leicht unterschiedlichen Variationen derselben Animation oder der Verwendung von längeren, komplexeren Bewegungen zu suchen. Dies könnte besser geeignet sein um die Aufmerksamkeit von Interessierten zu bekommen und zu halten.
Eine Frage zum Animationswechsel hätte den Fragebogen dahingehend gut ergänzen können, da so genauer festgestellt werden könnte, inwiefern es die Antworten zu den anderen Fragen beeinflusst, ob die betrachtende Person einen Animationswechsel miterlebt hat oder nicht.

Es ist außerdem denkbar, weitere Eigenschaften aus dem Musikstück in die Tanzbewegung einfließen zu lassen. Ein recht leicht umsetzbares Beispiel könnte sein, die Intensität der Bewegung an die Lautstärke zu koppeln.

Das Animationstool war im Verlauf der Arbeit sehr hilfreich und einfach zu bedienen. Nach dem initial recht hohen Entwicklungsaufwand ließen sich die Animationen selbst sehr schnell damit kreieren, was das Experimentieren mit verschiedenen Animationen auch im knappen Zeitrahmen der Arbeit ermöglichte. Es könnte dennoch erweitert werden. So wäre es hilfreich, die Anzahl an Zeiteinheiten sowie Spline Segmenten anpassen zu können, um noch mehr Flexibilität und Kontrolle beim Erstellen der Animationen zu ermöglichen.

Da Doggy2 nur über drei Bewegungsachsen verfügt, ist die Anzahl an möglichen, ausreichend unterschiedlichen Bewegungen, die zudem noch erahnen lassen, dass die Bewegung auf den Beat abgestimmt stattfindet und im Zeitraum von zwei Beat Intervallen von Doggy2 flüssig ausführbar sind, begrenzt. Die Dauer einer Animation auf beispielsweise vier oder mehr Beat Intervalle zu verlängern, würde mehr kreativen Spielraum für komplexere und interessantere Animationen schaffen.

Abbildungsverzeichnis

Tabellenverzeichnis

Quellen

Aucouturier, J.-J., Ogai, Y. und Ikegami, T. (2008) „Making a Robot Dance to Music Using Chaotic Itinerancy in a Network of FitzHugh-Nagumo Neurons“, in M. Ishikawa u. a. (Hrsg.) *Neural Information Processing*. Berlin, Heidelberg: Springer, S. 647–656. Verfügbar unter: https://doi.org/10.1007/978-3-540-69162-4_67

Avrunin, E. *u. a.* (2011) „Effects related to synchrony and repertoire in perceptions of robot dance“, in *2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI). 2011 6th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, S. 93–100. Verfügbar unter: https://doi.org/10.1145/1957656.1957678

Bartneck, C. (2023) „Godspeed Questionnaire Series: Translations and Usage“, *International Handbook of Behavioral Health Assessment*. Herausgegeben von C. U. Krägeloh, M. Alyami, und O. N. Medvedev. Cham: Springer International Publishing. Verfügbar unter: https://doi.org/10.1007/978-3-030-89738-3_24-1

Bartneck, C. *u. a.* (2009) „Measurement Instruments for the Anthropomorphism, Animacy, Likeability, Perceived Intelligence, and Perceived Safety of Robots“, *International Journal of Social Robotics*, 1(1), S. 71–81. Verfügbar unter: https://doi.org/10.1007/s12369-008-0001-3

Bartsch, M. (2015) *Sound of Interest - Ein Ballspielroboter hört stereo*. Verfügbar unter: https://www.informatik.uni-bremen.de/agebv2/downloads/published/bartsch_thesis_15.pdf (Zugegriffen: 3 Juni 2024)

Berl, E., Pakrasi, I. und LaViers, A. (2019) „Creating Context Through Performance: Perception of the ‘Dancing Droid’ Robotic Platform in Variable Valence Interactions in Distinct Office Environments“. Verfügbar unter: https://doi.org/10.1007/978-3-030-35888-4_27

Bugden, W. und Alahmar, A. (2022) *Rust: The Programming Language for Safety and Performance*. arXiv. Verfügbar unter: https://doi.org/10.48550/arXiv.2206.05503

Goto, M. (2006) „A chorus section detection method for musical audio signals and its application to a music listening station“, *IEEE Transactions on Audio, Speech, and Language Processing*, 14(5), S. 1783–1794. Verfügbar unter: https://doi.org/10.1109/TSA.2005.863204

Holmér, F. (2022) *The Continuity of Splines*. Youtube [Online Video]. Verfügbar unter: https://www.youtube.com/watch?v=jvPPXbo87ds&t=2643s (Zugegriffen: 3 Juni 2024)

Jaime Gancedo, W. S., Sakif Hossain (2018) *Design and implementation of a Beat Detector algorithm*. Verfügbar unter: https://www.eit.lth.se/fileadmin/eit/courses/etin80/2018/reports/beat-detector.pdf (Zugegriffen: 15 Februar 2024)

Jayaram, V. (2018b) *Finding Choruses in Songs with Python*. Verfügbar unter: https://towardsdatascience.com/finding-choruses-in-songs-with-python-a925165f94a8s (Zugegriffen: 15 Februar 2024)

Jayaram, V. (2018a) *PyChorus: Python module for detecting musical choruses*. Verfügbar unter: https://github.com/vivjay30/pychorus (Zugegriffen: 15 Februar 2024)

Michalowski, M. P., Sabanovic, S. und Kozima, H. (2007) „A dancing robot for rhythmic social interaction“, in *2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI). 2007 2nd ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, S. 89–96. Verfügbar unter: https://doi.org/10.1145/1228716.1228729

Oliveira, J. L. *u. a.* (2012) „Beat Tracking for Multiple Applications: A Multi-Agent System Architecture With State Recovery“, *IEEE Transactions on Audio, Speech, and Language Processing*, 20(10), S. 2696–2706. Verfügbar unter: https://doi.org/10.1109/TASL.2012.2210878

Oliveira, J. *u. a.* (2012) „An Empiric Evaluation of a Real-Time Robot Dancing Framework based on Multi-Modal Events“, *TELKOMNIKA Indonesian Journal of Electrical Engineering*, 10. Verfügbar unter: https://doi.org/10.11591/telkomnika.v10i8.1327

Peng, H. *u. a.* (2015) „Robotic Dance in Social Robotics—A Taxonomy“, *IEEE Transactions on Human-Machine Systems*, 45(3), S. 281–293. Verfügbar unter: https://doi.org/10.1109/THMS.2015.2393558

Rafii, Z. *u. a.* (2019) *MUSDB18-HQ - an uncompressed version of MUSDB18*. Zenodo. Verfügbar unter: https://doi.org/10.5281/ZENODO.3338373

Santiago, C. B. *u. a.* (2011) „Autonomous robot dancing synchronized to musical rhythmic stimuli“, in *6th Iberian Conference on Information Systems and Technologies (CISTI 2011). 6th Iberian Conference on Information Systems and Technologies (CISTI 2011)*, S. 1–6. Verfügbar unter: https://ieeexplore.ieee.org/document/5974339/references#references (Zugegriffen: 4 Februar 2024)

Seo, J.-H. *u. a.* (2013) „Autonomous Humanoid Robot Dance Generation System based on real-time music input“, in *2013 IEEE RO-MAN. 2013 IEEE RO-MAN*, S. 204–209. Verfügbar unter: https://doi.org/10.1109/ROMAN.2013.6628446

Shinozaki, K., Iwatani, A. und Nakatsu, R. (2008) „Construction and evaluation of a robot dance system“, in *RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication. RO-MAN 2008 - The 17th IEEE International Symposium on Robot and Human Interactive Communication*, S. 366–370. Verfügbar unter: https://doi.org/10.1109/ROMAN.2008.4600693

Spillner, L. (2018) *Interacting With a Ball-Playing Entertainment Robot*. Verfügbar unter: https://www.informatik.uni-bremen.de/agebv2/downloads/published/spillner_thesis_18_with_appendix.pdf (Zugegriffen: 3 Juni 2024)

Tim Laue, T. H. U. F., Oliver Birbach (2014) „An Entertainment Robot for Playing Interactive Ball Games“. Verfügbar unter: https://www.informatik.uni-bremen.de/agebv2/downloads/published/laue_robocup_13.pdf (Zugegriffen: 3 Februar 2024)

Tzeng, Y.-r. (2013) *New Dog New Tricks - Human-Robot Interaction and Appearance Design of a Ball Playing Robot*. Verfügbar unter: https://www.informatik.uni-bremen.de/agebv2/downloads/published/tzeng_thesis_13.pdf (Zugegriffen: 3 Juni 2024)

Weidenbach, P. (2019) *Integration and Evaluation of an Interactive Ball-Playing Robot*. Verfügbar unter: https://www.informatik.uni-bremen.de/agebv2/downloads/published/weidenbach_thesis_19.pdf (Zugegriffen: 3 Februar 2024)

Woerner, T. (2021) *Synthese eines geeigneten Reglers für einen Ballspielroboter mit redundanter Achsansteuerung*. Verfügbar unter: https://www.informatik.uni-bremen.de/agebv2/downloads/published/woerner_thesis_21.pdf (Zugegriffen: 3 Februar 2024)