MUSE: A Music Conducting Recognition System

Chase D. Carthen*
Richard Kelley

Cris Ruggieri
Sergiu M. Dascalu

Justice Colby
Frederick C. Harris, Jr.

Dept. of Computer Science and Engineering
University of Nevada
Reno, Nevada 89557
chase@nevada.unr.edu, organicjustice@gmail.com, cris.ruggieri@gmail.com
fred.harris@cse.unr.edu, dascalus@cse.unr.edu, richard.kelley@gmail.com

***Abstract*—In this paper, we introduce Music in a Universal Sound Environment(MUSE), a system for gesture recognition in the domain of musical conducting. Our system captures conductors' musical gestures to drive a MIDI-based music generation system allowing a human user to conduct a fully synthetic orchestra. Moreover, our system also aims to further improve a conductor's technique in a fun and interactive environment. We describe how our system facilitates learning through a intuitive graphical interface, and describe how we utilized techniques from machine learning and Conga, a finite state machine, to process inputs from a low cost Leap Motion sensor in which estimates the beats patterns that a conductor is suggesting through interpreting hand motions. To explore other beat detection algorithms, we also include a machine learning module that utilizes Hidden Markov Models (HMM) in order to detect the beat patterns of a conductor. An additional experiment was also conducted for future expansion of the machine learning module with Recurrent Neural Networks(rnn) and the results prove to be better than a set of HMMs. MUSE allows users to control the tempo of a virtual orchestra through basic conducting patterns used by conductors in real time. Finally, we discuss a number of ways in which our system can be used for educational and professional purposes.**

***Index Terms*—pattern recognition, machine learning, music, hidden markov models, education**

1 Introduction

Although music is fundamentally an aural phenomenon, much of the communication required for an ensemble to produce music is visual in nature. In particular, musical conductors use hand gestures to communicate with musicians. Musical conductors only form of feedback for improving their conducting is mainly received from a band and is very limited without the aid of a band. Beginning conductors lack an efficient way to practice conducting without a band present to help them hone their skills and lack the skill of experienced conductors that have worked many hours with bands. There are currently many gesture recognition software that have been built for the purpose of making music easier and enjoyable.

Recent advances in sensor technology are making such gesture recognition feasible and economical; cameras such as Microsoft's Kinect and the Leap Motion Controller are able to estimate the kinematics and dynamics of arms and hands in real time, meaning that conductors' gestures can now be accurately and cheaply recorded by the computer. There have been a few projects that have investigated and implemented different methods of capturing conducting. These methods have utilized the Leap Motion and Kinect in order to control music with gestures in different ways. Projects such as those from the developers WhiteVoid that utilizes a single Leap Motion, baton, and speakers while allowing users to control the tempo and individual instruments [1]. Another project [2], utilizes the Kinect in order to capture conducting gestures for manipulating tempo and the volume of individual instruments. There have been some old systems in the past that have been used outside of conducting but more for creating a coordinated performance such as [3] where many cellphones were synchronized for playing a symphony. These systems can provide a way for experienced or new users to learn conducting with feedback in a new way.

MUSE has been designed for beginning conductors to learn and enjoy conducting. MUSE allows users to learn simple conducting patterns and utilizes the Leap motion sensor's API [4], and algorithms such as: a finite state machine recognizer called Conga [5] and HMMs. These algorithms allow MUSE to effectively detect a user's conducting basic conducting patterns and control the tempo of a given song. An additional experiment was also conducted with rnn to determine if it is better than a HMM and to be later Incorporated into MUSE. The rnn experiment is included since not many papers seem experiment with using an rnn. MUSE provides a graphical user interface (GUI) for users to control and select different songs and even record previous performances of songs. Unlike other previous conducting projects MUSE has been designed as an architecture that is expandable.

In this paper we first describe the related work. Secondly, we then outline the architecture of our system and some of its functionality. We then give details on the algorithms used to capture conducting by the system and their accuracy. In the next section, We discuss our system's applications more throughly. Finally, we conclude with a discussion of further applications and future work. Throughout the rest of this paper we will refer to the Leap Motion as the Leap.

2 Background

The task of building software catered to beginner conductors requires an understanding of basic conducting patterns. As mentioned earlier there have been other software designed to recognize conducting with different degrees of complexity. Some software simply captures the change in velocity and alters the tempo, while other software may attempt to capture a gesture formation in conjunction with the change in velocity. MUSE is able to capture both the change in velocity and a gesture pattern. The gesture patterns of conducting is described in the Conducting Patterning and Ictus section.

2.1 Conducting Patterns and the Ictus

MUSE has been designed to recognize basic conducting patterns that are taught to beginning conductors. Figure 1 gives an example of these basic patterns. These conducting patterns consist of ictus's or where the beat occurs typically when the conductor's hand switches in a major direction. Figure 1 demonstrates three conducting patterns where the left most has two ictus's, the middle has three ictus's, and the rightmost has four ictus's. Basic conducting patterns are typically used to help beginning conductors to gain a grasp on keeping tempo with a band and to lead into more complex conducting styles.

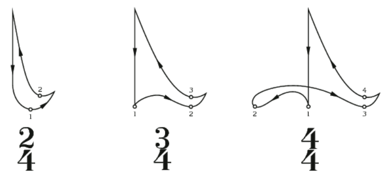

Fig. 1. Three basic conducting patterns that are taught are shown above. On the left is a two beat pattern, in the middle is a right three beat, and on the right is a four beat pattern. The beats or ictus's are located at the ends of the arrows. [6]

2.2 Sensors for Gesture Recognition

In order to perform gesture recognition, the first obvious requirement is a system of sensors to record raw data from a person's motions. Previous systems for capturing gestures can be broken into two broad approaches: camera- based systems and controller based systems. A camera-based system utilizes camera's to capture a person's hand or baton motions, while a controller based system utilizes some kind of controller to capture a user's hand motion. Both types of recognition make use of gesture recognition algorithms to detect the beats of a conducting pattern.

An example of a controller used for gesture recognition, in particular, is the controller for Nintendo's Wii known as the Wiimote. The Wiimote has been successfully used to recognize a number of gestures with a fairly small training set [7]. The Wiimote has accelerometers that make it possible to keep track of orientation and makes use of a infrared sensor bar to get position information. The WiiMote is a great tool for creating a conducting application. Despite the WiiMote and other controller based systems being useful for capturing gestures for the use of conducting, they introduce a new interaction that may seem foreign to a conductor.

There are primarily three types of camera based systems: traditional intensity cameras, stereo camera systems, and depth cameras. In this paper our system uses depth based system camera like the Kinect. The Kinect has been used to build a commercially-successful pose recognition system for humans in indoor settings [8]. In this paper, we use a system similar to the Kinect to track hand motions specifically.

There has been some work on gesture recognition in conducting including work that was previously discussed in the Introduction section. In particular, the Conga framework [5] that uses finite state machines to recognize conducting, dynamic time warping (DTW) which has been used in [9], [10] to improve the overall accuracy of detecting conducting patterns, and HMMs [11]. The systems in previous apporaches with HMMs, DTW, and others are very accurate and have 95% or above accuracy. However, finite state machines such as Conga are constructed by hand and manual construction obviates the need to perform a lengthy training process, the use of a non-statistical approach can lack the robustness required to deal with noisy sensors and inexperienced conductors. Despite the accuracy of these apporaches, our system uses the Conga frame to recognize different conducting patterns and extends the Conga framework by using HMMs to perform the necessary sequence analysis to deal with noise. However, beats are still captured by the Conga framework exclusively.

3 System Architecture and Functionality

MUSE system architecture has been designed with a culmination of multiple dependencies which are: QT to be used as a Graphical User Interface (GUI)[12], rtmidi to handle communication to other midi input and output devices[13], libjdksmidi to parse midi files and handle midi events[14], the Leap Motion API to capture motion information from the Leap Motion device[4], Nick Gillian's Gesture Recognition Toolkit (GRT) to create a Hidden Markov HMM for recognizing conducting gestures, implementing the Conga framework in C++ to capture a majority of conducting gestures, and OpenGL to visualize the conductor's hand [15], and zlib for compressing records generated by user's performances [16]. The rnn is currently not part of MUSE and will be added later. These dependencies have made it possible to build MUSE.

3.1 Overview of the Architecture

The overall picture of MUSE's architecture can be seen in Figure 2. MUSE can be broken done into four simple layers being its presentation layer, midi processor layer, hardware layer, and pattern recognition layer. The presentation layer consists of the GUI and everything necessary to create a virtual view of the user's hand. The hardware layer of MUSE is simply the Leap Motion itself and the Leap Motion API required to use the Leap. The Pattern Recognition consists of two different beat pattern recognizers being the Conga finite state machine and the use of GRT to create a HMM. The Midi

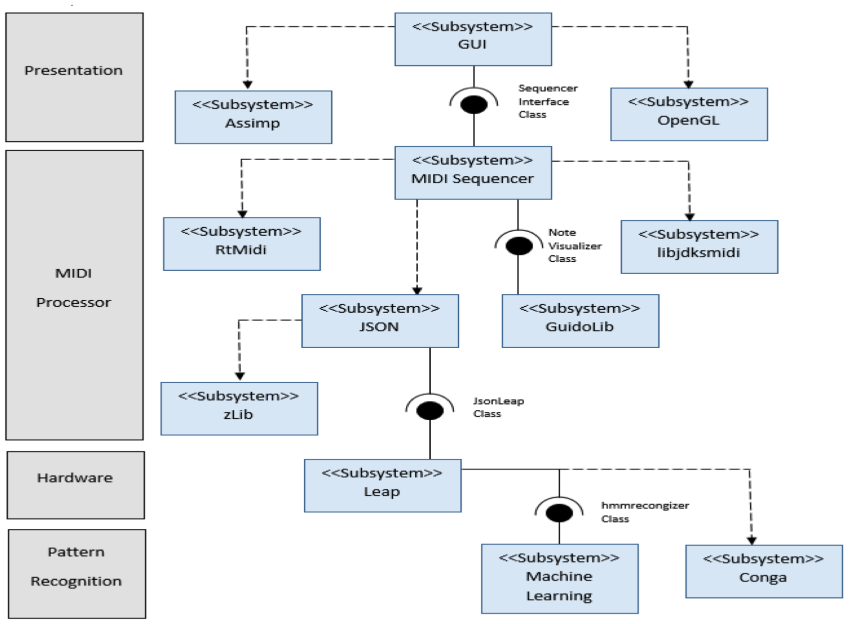

Fig. 2. The system architecture is demonstrated in this figure. The system consists of a presentation layer that shows information to the user, a midi processor for manipulating the musical output of the system, a pattern recognition layer where all code for detecting conducting patterns is placed. All interfaces within MUSE demonstrated in this figure as well.

Processor layer handles processing midi files by controlling midi playback speed in the case of changing conductor input and has the capability of serializing and recording a user's hand motions for future playback. In the next following sections the presentation layer and MIDI processor layer are explained in detail. The pattern recognition layer is explained later in Section 4.

3.1.1 Presentation Layer: The GUI of MUSE is designed in a similar fashion to any music player in the sense that it allows the user to select and play songs at their leisure. As the song plays, the user may provide gestural input via a Leap Motion controller to guide musical playback. Also, they can alter the play-style or even the volume of the instruments which are provided as controls in the GUI. MUSE has components for choosing specific models of conducting and visualization as well as opening MIDI files. A screen shot of MUSE's GUI can be seen in Figure 3 and the individual components of the screen are detailed as follows:

- OpenGL Visualizer: The OpenGL visualizer displays the motions of a user's hand and allows the user to see what the Leap is detecting. This interface is provided so that the user may be able to determine what is the best area to conduct over the Leap.
- Channel Manipulation: The channel manipulation which is the sixteen slider bars next located next to the OpenGL Visualizer. In this section of the GUI the user is able to manipulate different 16 different midi channels of MIDI files that are playing. Each channel correlates to a different instrument that is playing on the MIDI file. The volume sliders will also generate specific volumes based on the configuration of each MIDI file that is opened. The sliders operate and provide information in real-time because the variables for volume are changing constantly.
- Tempo Manipulation: A tempo manipulation toggle allows users to change the tempo of each song (for the song to play faster or slower). When a MIDI file is loaded the tempo within the midi file is automatically loaded and can be changed with the plus and minus below the tempo next to the OpenGL visualizer. It will also change in real-time as a user moves their hand or conducting instrument over the Leap depending on the velocities of their movements.
- Beat Pattern Detection Menu: Located above the tempo selection and to the left of the OpenGL visualizer are the options to choose a 2-pattern, 3-pattern, and 4-pattern. Above these options are the displays for the desired beat pattern and the current detected beat for that pattern. We introduced this feature in order to make detecting beat patterns easier and more accurate. We moving towards a system where it expects certain beats patterns and to grade the user.
- Note Manipulation Section: Located next to the next below the channel manipulation is the note manipulation section. This section is dedicated to manipulating different play styles of the midi which we have denoted as attack, staccato, legato, and marcato. Only one of these states can be on at any given time. This functionality was added as a precursor for detecting the type of play attached to different types of conducting that are usually captured in a conductors motion.
- Files Menu: The user interface has drop down menus that will be utilized in order to open files, exit the program, record conducting, and to provide information

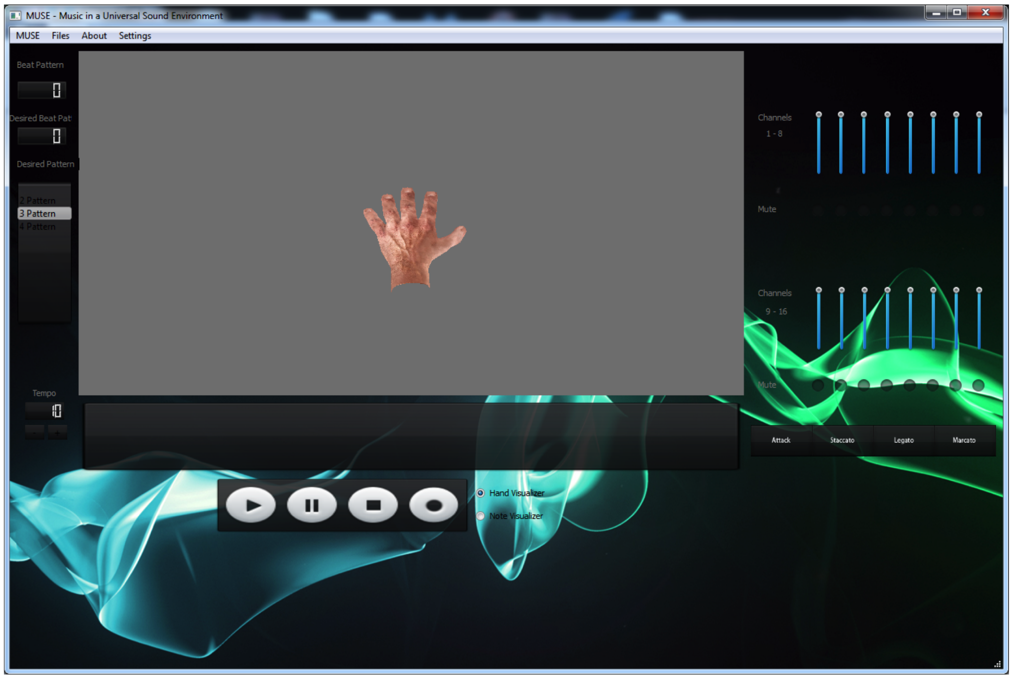

Fig. 3. An example screenshot of MUSE that demonstrates MUSE's GUI.

on the program itself. The recording functionality will be explained later in Section 3.1.2.

- Musical Note Visualizer: Demonstrated below the below the OpenGL Visualizer is small view of the musical note visualizer that the user can see what notes are currently playing. An example of the note visualizer output can be seen in Figure 4.

3.1.2 Midi Processor Layer: The midi processor consists of a midi sequencer, an rtmidi module, libjdksmidi module, Guidolib module, and a JSON module. Each of these modules has a unique independent/dependent responsibility to the midi processor. The libjdks module is responsible for reading and parsing a midi file for all midi events existent in the midi. The sequencer is responsible for determining what midi events to feed to the Rtmidi module for playback dependant upon input given from the pattern recognition level. The rtmidi module allows users to have the option to send midi signals to other midi output devices that could be connected to other software or synthesizers.

Also, MUSE allows a user to record their performance. This requires the midi processor layer to keep track of all motion created by the user made during the playback of a song. Every frame of information that is acquired from the Leap is recorded and timestamped. When a user has finally finished recording a song and their gesture motions, the motions are serialized into a JSON format with the necessary information. The JSON string generated is then compressed with zlib and saved at a location specified by the user. When a user goes to select this recorded file again for playback, the user's recording will be played back in the same fashion as before. This functionality

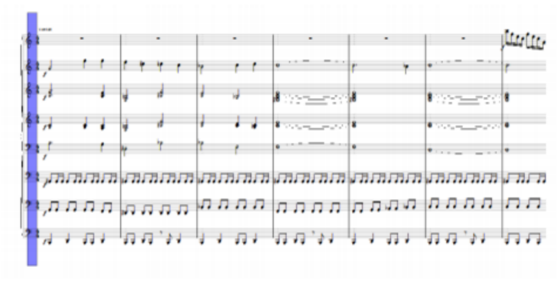

Fig. 4. A close up of the Note Visualizer. The visualizer updates in real time, according to the gestures of the user and displays notes of a selected instrument to the user.

could be used a feed back to give the user insight on how well they are doing with their conducting. We choose to serialize our data as JSON in order to account for future expandability for other user data that may be beneficial.

4 Machine Learning Module Theory and Results

In this section the rnn and HMMs used for predicting beat patterns are discussed as a part of the Pattern Recognition layer. Both of these classifiers were strictly made to classify the patterns themselves, but not the individual beats inside of the pattern. Also the HMMs have so far only been implemented for the machine learning module. The machine learning module was built to automatically determine which pattern is being detected without the user selecting a preselected pattern. Right now MUSE primarily relies upon the user's selection and uses Conga to determine the change in tempo based on Conga primarily.

Based on the details from the Conga paper [5], we have implemented a finite state machine to be used with MUSE. We have spent time analyzing the Leap's output when someone is conducting in order to figure out what is needed to capture an ictus or down beat in a conducting pattern. Our finite state machine keeps track of the x and y position, velocity, and acceleration. Our finite state machine effectively looks for zero crossings in the velocity and acceleration that denotes a major change in direction, which is an ictus in conducting. A unique finite state machine was created for a two beat pattern, three beat pattern where the second beat is on the left, three beat right pattern where the second beat is on the right, and four beat pattern. These four patterns, that are recognized by Conga, are used for MUSE and for training both the HMMs and rnn.

The HMMs in the Pattern Recognition layer uses any velocity measurements received and computed by the Leap Motion API as input. Inside the Pattern Recognition layer of MUSE, the velocity measurements are preprocessed into a discrete alphabet and then passed into a HMM for each pattern supported by MUSE. There are four unique HMMs representing each beat pattern supported by MUSE. Predicting the beat pattern is then selected based on the model with the greatest likelihood. This classification is not accepted as the ground truth until the pattern is completed. Conga is used to determine if a pattern has truly been completed. In order to capture the ordering and complexity of beat patterns a left to right Hidden Markov model structure was chosen to model the data captured from the Leap. The left to right Hidden Markov model was chosen because a beat pattern is a cyclic in nature. An alternative model type is an ergodic HMM, which does not model a beat pattern well because it does not have a strict path for state sequences, which makes the ergodic model a bad choice.

In creating the discrete alphabet for the HMM only the x and y components of the velocity data are used.

The velocity vector captured from the Leap is encoded with the following function:

$$D = \lfloor vec.roll + \frac{1}{2} \rfloor \bmod 16 \qquad (1)$$

Where vec.roll is the roll of the current velocity with respect to the z axis, which can be computed by the Leap Motion API. The roll was chosen for the decoding function because it points in the direction that the hand is currently going. The above formula in Equation 1 that uses Equation **??** was inspired by another paper[17] and the equation encodes acceleration vectors into 16 different quadrants to be passed into the HMMs and rnn.

An rnn was built with keras[18] to test how well it would work in comparison to the HMMs. The rnn's architecture is a single long short term memory (lstm) layer follow by a regular feed-forward layer and the output is squashed with a softmax layer. The rnn's objective function is a categorical cross entropy function. The rnn has four outputs were each output indicates the most probable beat pattern.

TABLE I
COMPARISON OF THE RNN VERSUS THE HMMS ON THE CONGA LABELED DATASET AND MANUALLY LABELED DATASET.

	Conga Labeled	Manually labeled
rnn	**98.1%**	**87.3%**
HMMs	78.9%	50.7%

The MUSE system utilizes data collected from the Leap Motion API to train HMMs and a RNN. The Leap Motion API provides data at rates of 100 FPS to 200 FPS enough to predict conducting gestures. A dataset constructed from 140 data samples from six different people who conducted for 30 seconds was used to train both HMMs and the RNN. The conga finite machine was used to label this dataset and it was split into 80% training and 20% testing. Another dataset was labeled manually by pressing every time a beat occurs from one person with 40 datasets that were 10 seconds each. Table I demonstrates the overall results of the rnn and HMMs over both datasets. The rnn performs better than the HMMs on both datasets with 98.1% on the Conga labeled and 87.3% on the manually labeled dataset.

The difference in performance between the Conga and Manually labeled datasets can be explained by the way the two datasets were labelled. In processing both these datasets the individual patterns in each recording were split apart and labeled. For example, if one recording had four patterns within it, then those four patterns would be broken done into four individual sequences and labelled. In labeling the Conga finite state machine will have some delay in determining the end of a beat pattern and is limited by the design of the finite state machine created for the pattern. In the manually labelled dataset the end of a pattern is limited by the skill and delay of the key press by the one labeling the dataset. The difference in the results demonstrates that the end of patterns were very different between the two datasets and actually impacts the two classifiers.

5 Discussion

MUSE has been designed for students and teachers to use this system for learning basic conducting patterns. A lot of the features have been tailored for constructing future games and assignments for teachers and students. However, MUSE is need of improvements overall, especially in more robust beat detection algorithms and the inclusion of other sensors. A lot of other projects have focused on building systems that are able to effectively capture conducting accurately. Unlike these systems, MUSE's overarching goal is to capture the motions of a conductor and provides users feedback in a scalable system. MUSE has incorporated a way to save past performances that is independent of other sensors that could allow for the addition of new sensors.

MUSE has several different features that makes it easily adaptable for teaching purposes and scalable for other sensors that may be added further on. MUSE is capable of taking in midi and producing output to different output sources. Users are able to save and record performances, alter the playing style of music, and choose different conducting patterns that they wish to be recorded. Having the capability to select different beat patterns allows for different conducting scenarios to be constructed in the future. Having the ability to select different beat styles allows for MUSE to have the extensibility for algorithms that detect the style of conducting. Using midi as an output source is more flexible then audio as it can be controlled robustly. The user is also able to view how notes are changing through time, effectively allowing a user to examine a visual representation of the music. Despite these features, MUSE has several improvements that could be made.

MUSE's HMMs were created for the purposes of capturing four basic conducting patterns. These algorithms are accurate enough to perceive the conducting patterns and even give the user feedback. The feedback from MUSE can be severely impacted by any inaccuracies from the limited range and sensitivity of the sensor. Despite these inaccuracies, the rnn demonstrated in the previous may be beneficial for improving the overall accuracy of MUSE's current use of HMMs.

The rnn discussed in the previous section was found to be more accurate that the hmms. There have been very few experiments with music conducting and determining a beat pattern. It was found that the rnn outperforms classifying individual beat pattern segments. However, it has not been added to MUSE yet. These results require further investigation and a real time classifier needs to be made for different types of rnns.

6 Conclusions and Future Work

The MUSE application coalesces a variety of different tools and concepts to provide a versatile set of functionality behind an intuitive interface. Having incorporated a large amount of research as to the conventions of conducting as well as the needs of the music community, MUSE offers a solution to a ubiquitous problem in a unique way. Through the inclusion of a finite state machine beat recognizer using the Conga framework, the application provides an accurate method to not only keep track of a users conducting patterns but also record their movements. There are some future plans to improve the accuracy of the conducting recognition system by using techniques such as DTW mentioned in the introduction and the rnn. We also have plans to incorporate different sensors such as the Kinect. Another goal is to implement the system as a web application and to incorporate MUSE into a virtual reality application, such as a six-sided cave or an Occulus Rift. Besides tracking movements, the machine learning within MUSE demonstrates the ability to be adaptable for other algorithms. With a dynamic visualization component incorporated into the GUI as well as a robust manipulation of sound output, the technical back-end is well hidden to the user as they utilize MUSE to its full potential.

Acknowledgment

This material is based in part upon work supported by: The National Science Foundation under grant number(s) IIA-1329469. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] L. Stinson. (2014, may) Conduct a virtual symphony with touchscreens and an interactive baton. [Online]. Available: http://www.wired.com/2014/05/interactive-baton/

[2] L.-W. Toh, W. Chao, and Y.-S. Chen, "An interactive conducting system using kinect," in *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, July 2013, pp. 1–6.

[3] G. Levin, G. Shakar, S. Gibbons, Y. Sohrawardy, J. Gruber, E. Semlak, G. Schmidl, J. Lehner, and J. Feinberg. (2001, may) Dialtones (a telesymphony). [Online]. Available: http://www.flong.com/projects/telesymphony/

[4] "Api reference." [Online]. Available: https://developer.leapmotion.com/documentation/index.html

[5] E. Lee, I. Grüll, H. Kiel, and J. Borchers, "conga: A framework for adaptive conducting gesture analysis," in *Proceedings of the 2006 conference on New interfaces for musical expression*. IRCAMCentre Pompidou, 2006, pp. 260–265.

[6] B. Smus, "Gestural music direction," May 2013. [Online]. Available: http://smus.com/gestural-music-direction/

[7] T. Schlömer, B. Poppinga, N. Henze, and S. Boll, "Gesture recognition with a wii controller," in *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, ser. TEI '08. New York, NY, USA: ACM, 2008, pp. 11–14. [Online]. Available: http://doi.acm.org/10.1145/1347390.1347395

[8] J. Shotton, T. Sharp, A. Kipman, A. Fitzgibbon, M. Finocchio, A. Blake, M. Cook, and R. Moore, "Real-time human pose recognition in parts from single depth images," *Commun. ACM*, vol. 56, no. 1, pp. 116–124, Jan. 2013. [Online]. Available: http://doi.acm.org/10.1145/2398356.2398381

[9] R. Schramm, C. Rosito Jung, and E. Reck Miranda, "Dynamic time warping for music conducting gestures evaluation," *Multimedia, IEEE Transactions on*, vol. 17, no. 2, pp. 243–255, Feb 2015.

[10] R. Schramm, C. R. Jung, and E. R. Miranda, "Dynamic time warping for music conducting gestures evaluation," *IEEE Transactions on Multimedia*, vol. 17, no. 2, pp. 243–255, 2015.

[11] L. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb 1989.

[12] "Qt - home," 2016. [Online]. Available: http://www.qt.io/

[13] G. P. Scavone, "Introduction." [Online]. Available: https://www.music.mcgill.ca/~gary/rtmidi/

[14] J. Koftinoff, "jdksmidi." [Online]. Available: https://github.com/jdkoftinoff/jdksmidi

[15] "The industry's foundation for high performance graphics." [Online]. Available: https://www.opengl.org/

[16] G. Roelofs and M. Adler, "zlib home site," Apr 2013. [Online]. Available: http://www.zlib.net/

[17] D. Schmidt, "Acceleration-based gesture recognition for conducting with hidden markov models," Ph.D. dissertation, Ludwig-Maximilians-Universitaet, 2007.

[18] F. Chollet, "keras," https://github.com/fchollet/keras, 2016.