Music in Motion: Gestural Music Generation Using Microsoft Kinect v2 and Ableton 9 Live

Emily Abele, Matthew Cox, Andrew Hollenbach, Ameya Lonkar, Vishwanath Raman

Rochester Institute of Technology, 2014

Abstract

This work experiments with initiating music generation in real-time using human movements. We present a system that discovers the human skeleton and then translates that into music controls. The stage parameters are defined and, within that, the skeletal position is tracked using the Kinect's real-time skeletal tracking system. The resulting position and gestures of the skeleton are translated into cues that are relayed to Ableton and then into musical cues. This is continually relayed and regulated by the user, the person who is giving the gestural cues.

1. Introduction

With the Kinect, we have obtained skeletal tracking data that can be translated into both programmed and learned gestures using Microsoft's own SDK, the Open Sound Control Library. The movements of the user are then passed as an OSC message into Ableton 9 Live. This signal is then registered as either a musical note that is subject to change with specific gestural movements of the user. The user gestures can be tracked in a fixed area within the Kinect's limits, which shall be referred to as the "staging area". The user also has the ability to create and delete loops in the different specified sections of the staging area, which has been partitioned to allow for a variety of different instruments. We have worked with a musician (Matthew Cox) to create an intuitive method of gesture-based music generation using the Kinect and Ableton 9 Live.

2. Background

2.1 Current Motion Capture Technology

Current systems utilize electromagnetic, electromechanical, fiber optic, or optical methods of 3D rotoscoping, which measures the 3D positions, orientations, velocities, and accelerations automatically. Electromagnetic motion capture uses sensors placed on the joints of a moving object to measure the positions and orientations of joints without occlusion and can capture multiple subjects; however, there is a small capture volume and it is not as accurate as other systems. Electromechanical motion capture uses each sensor to measure the 3D orientations and has a large capture volume, while being portable and allowing for outdoor capture; the information, though, it hard to gather and it lacks accuracy. Fiber optic motion capture uses tape bound to the body of the subject being captured and the orientation of the entire tape is measured and while it has all the pros of electromechanical motion capture, it is

highly intrusive. Optical motion capture utilizes multiple calibrated cameras, usually eight or more, to capture different angles and the subject wears retro-reflective markers. It is highly accurate, and can capture a lot of detail from the subject, but there are occlusion issues, it has a limited capture volume, and is expensive. [1]

2.2 The Kinect

To begin, the Kinect uses depth sensors and markerless motion capture by detecting the skeleton of a user using x, y, and z coordinates in a 3D space. The inferring of the body is a two-stage process that is completed by first creating a depth map using structured light and then inferring the body position using machine learning. The Microsoft Kinect's depth computation is done by PrimeSense hardware that is built directly into the Kinect; the depth map is constructed when the PrimeSense hardware analyzes patterns of infrared light [2]. Once this happens, the position of the skeleton is inferred from training examples that are then mapped to the depth images. When a game is played using the Kinect and a Microsoft Xbox system, a similar problem is being approached: the user is performing a series of gestures that must translate into game-based motions in order for the user to interact in real time with the system. Instead of music, however, these are gestures that relate to bowling, dancing, or possibly sword fighting. Either way, these gestures are recognized and relayed in real time as the Kinect identifies and relays the most likely position of the tracked skeleton that is the user's position

2.3 Previous Approaches

This problem of harnessing the Kinect's motion capture abilities for musical purposes has been approached in similar ways; the simplicity of the Kinect's markerless motion capture is appealing. A previous study on this topic used a similar tracking method. Joints were tracked and the information was used as a building block of their gesture-to-music application. In our case, the gesture-to-music application is Ableton 9 Live. The skeletal joint information was sent over OSC and the information unpacked as x, y, and z coordinates without orientation; this information is then placed into the range of 0 to 127 which is standard MIDI range. The x, y, and z values were assigned to carrier frequency, modulator frequency, and amplitude, respectively in a manner that was scalable [3]. Another approach took the quaternion representation, as determined the magnitude of the angle between the bones. This translated into the orientation of the bones, and the angle between them, made possible by the Kinect's reference of connect joints relative to sensor position [4]. While this is already, in a way, used by

games that require a Kinect, we wanted to create an experience that was more controlled by the individual user rather than governed by a predetermined set of motions.

3. System Overview

3.1 System Architecture

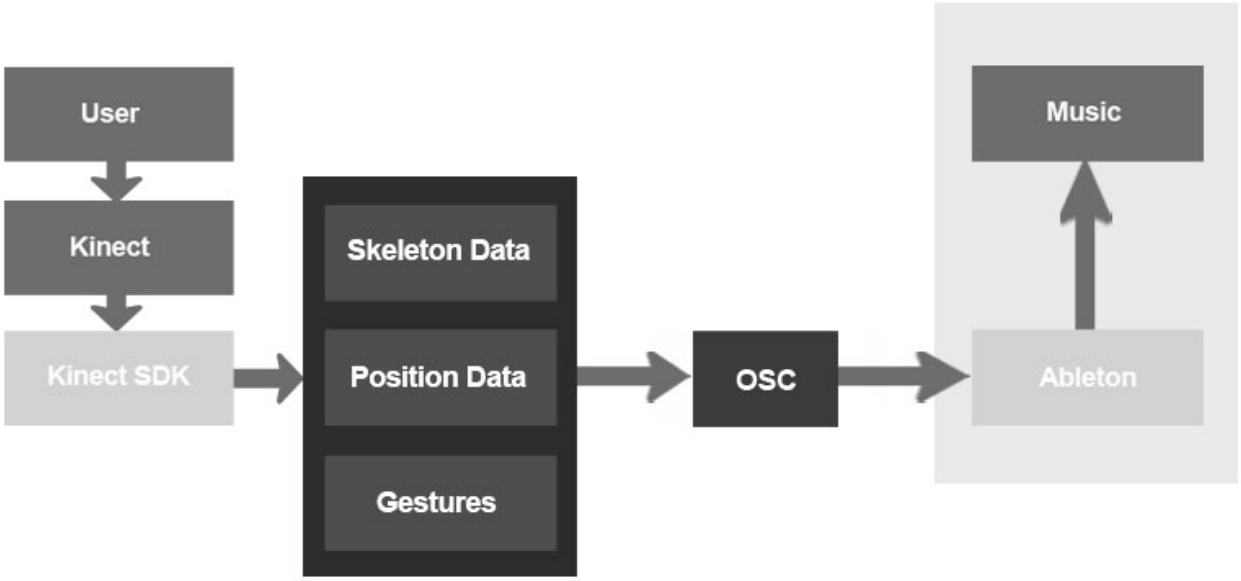


Figure 1. Component diagram of the system

The first component of the system is the user performing the gestures. These gestures performed, are then detected and processed by the Microsoft Kinect, which is turn sends it to the Kinect SDK (Modified Body Basics implementation). At this stage we have access to the complete skeleton and position data, using which, the conditions for various gestures are checked. Depending on the gesture(s) activated, a corresponding signal is sent to the OSC Instrument in Ableton. The OSC signal received, triggers a corresponding action or event in Ableton, for e.g., play a note, start a loop, set an instrument to current partition, overdub the current loop, etc. This action that is trigger controls the sound associated with a particular

specified instrument, set for a particular track in Ableton.

3.2 Inputs and Outputs

I. Partition Modes for the Staging Area.

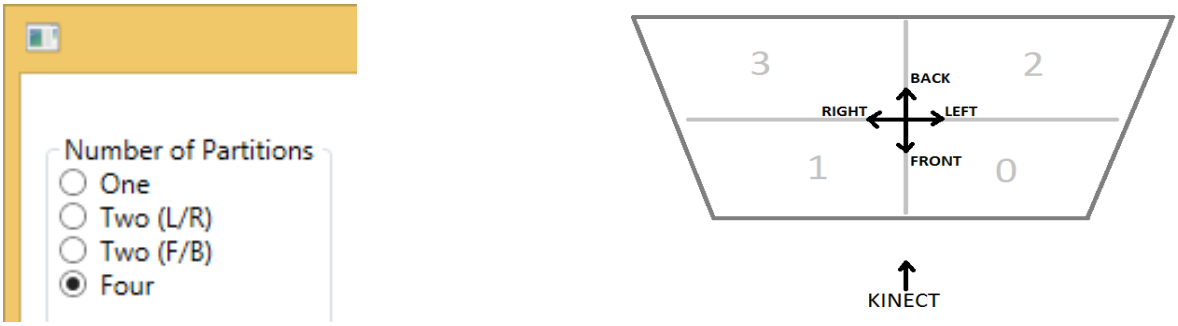


Figure 2. The user is assumed to be standing at the intersection of the four black/gray lines with their back facing BACK, and their front facing FRONT (as well as the Kinect)

L-R 2 (a) Radiobuttons to choose partition type.

2 (b) Top view of the staging area with depiction of partition types.

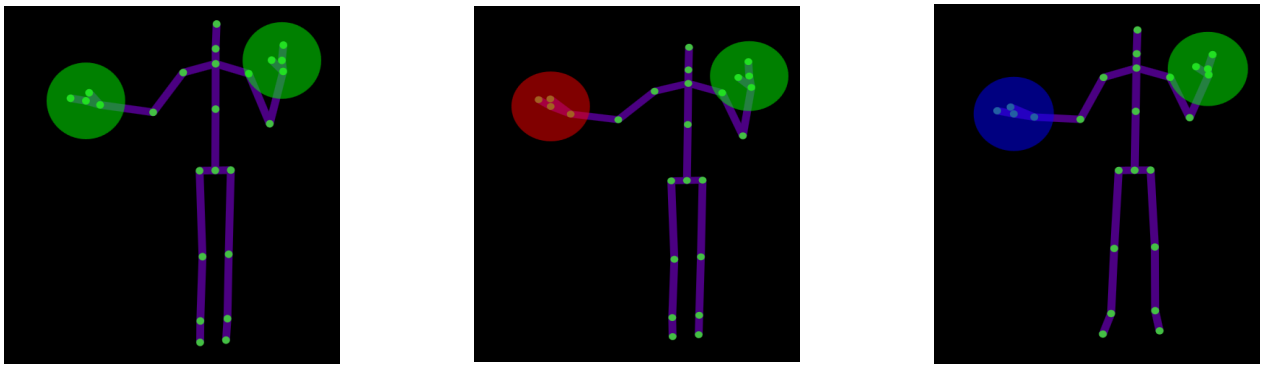
The mode for setting the type of partitions of the staging area is decided by choosing the required radiobutton in the top left of the window displayed. During initial startup, if no radiobutton is selected the default is “One” partition, i.e. the entire staging area is treated as a single partition. Other options are Two partitions (Left and Right), Two Partitions (Front and Back, and Quad partitions. For Quad Partitions, the layout is denoted by the numbers in 1.b., viz., 0, 1, 2 and 3.

II. Basic Gestures

The first and major input of this process is the gesture generated by the user. Specific gestures are used to control basic actions, which were determined by the musician, to emulate certain aspects while playing an instrument. The various inputs and their role is mentioned as follows;

The Open Hand gesture is just a fully opened hand with fingers straight and the palm ideally facing the Kinect. The Closed Hand gesture is to make a fist and for best results, the closed fingers and thumb, face the Kinect. The Lasso is a “Thumbs Up” gesture or one where a single finger is straight and extended and all other fingers are making a fist. These three gestures are

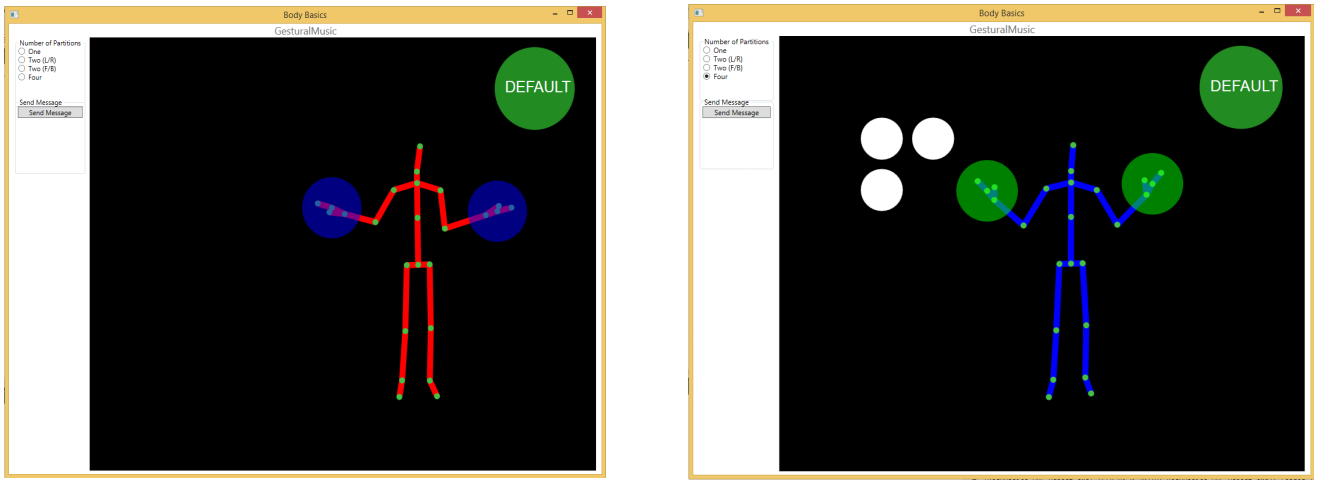
coupled with other body-relative positions to trigger various gestures for playing music through Ableton 9 Live.



(L - R) Figure 3. (a) Open Hand (b) Closed Hand (c) Lasso (Thumbs up)

Images depicting the built-in hand gestures recognized by the Kinect where the user's left hand is always open and the right hand demonstrates. One important thing to note is that the system first needs to establish a baseline for arm length, Although this could be done on the fly, it produces less consistent results. To do so, the user reaches straight out with hi or her arms and make a lasso on either hand.

III. Gestures for Instrument Selection

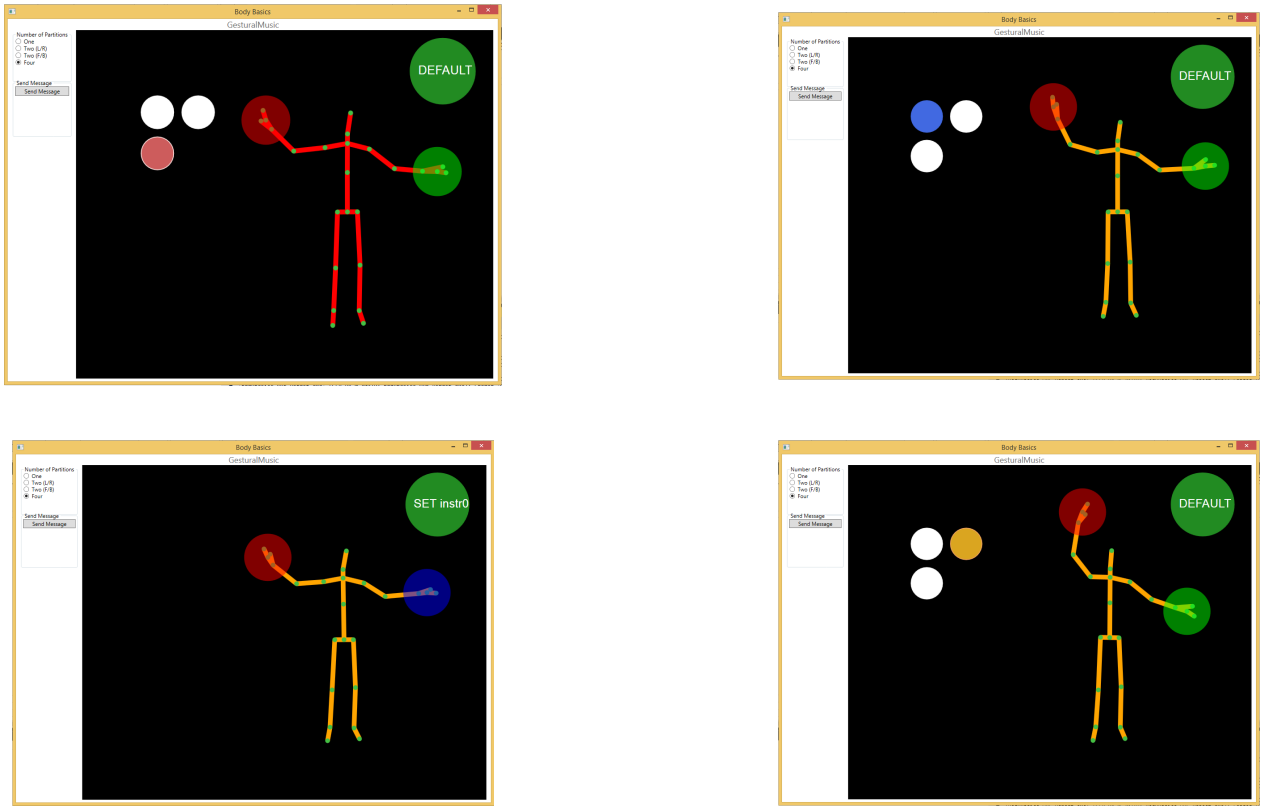


(L - R) Figure 4. (a) Set default instruments (b) Default case for instrument

There are two parts to setting instruments to a particular partition. 4.a. will enable the default

values to be set for the partitions, i.e., instr0 (instrument no. 0 for partition 0), instrument no. 1 for partition 1), etc

The other case is when a specific instrument is set to a partition using instrument selection. In an unset partition, having the right hand over the shoulder displays the three instruments that can be chosen. The type of instrument is chosen by the angle made by the right hand wrist w.r.t. the shoulder_left, where 25, 45 or 65 corresponds to instrument 0, 1 or 2 while the left hand is in a lasso position. To reset partitions, close the Visual Studio application window and run the program again (Close the Body Basics Window. NO need to close Visual Studio entirely)

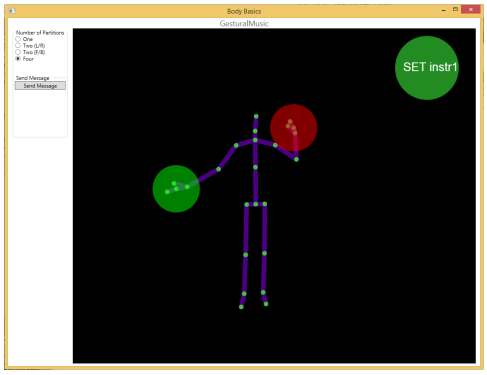


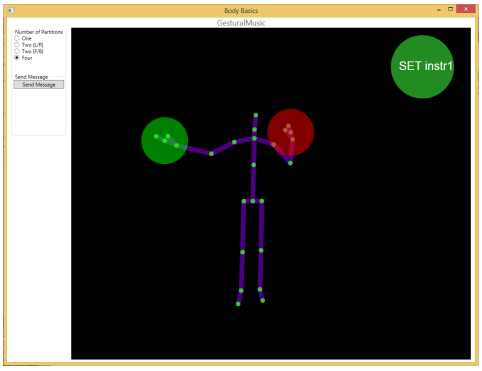
(L-R, Top to Bottom) Figure 5. (a) Selecting instrument 0 (b) Selecting instrument 1 (c) Selecting instrument 2 (d) Current partition set to instr0

Images depicting selection of a particular instrument for a partition. 5.d. shows current partition being set to instrument 0.

IV. Gestures for Playing Notes

These gestures are for playing specific semitones for an instrument track selected in Ableton 9 Live. An open or closed right hand signals not playing and playing of a note, respectively. The horizontal position of the left hand (i.e. along the x-axis) decides which note is being played, and the vertical position of the left hand (i.e. along the y-axis) decides the octave it is being played in. A closed left hand signals playing a black key note from the piano, and an open left hand signals playing of a white key note from the piano.





(L - R) Figure 6. (a) Playing note C in first octave (b) Playing note C in second octave

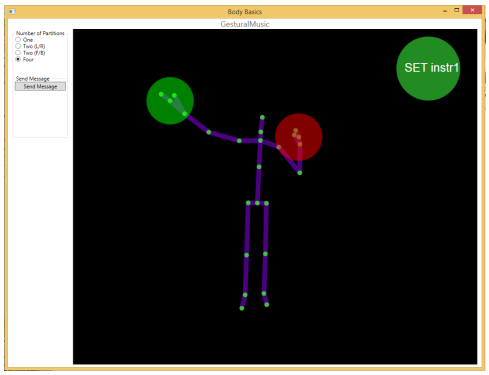


Figure 6. (c) Playing note C in the third octave.

V. Gestures for Loop Controls.

There are four actions possible with respect to looping a sample of music generated. We can signal; the start of recording a loop, play a loop (which will automatically stop the recording), stop playing of a loop, and start overdub on the current loop. The gestures for these controls are activated by the feet, which are as follows;

Right leg behind - Start recording a loop

Right leg forward - Start overdubbing the current loop

Left leg behind - Stop playing the current loop

Left leg forward - Start playing the current loop

To be able to register these gestures the right hand needs to be in a lasso position and the left hand must be an open hand. Both hands must be below shoulder level. It is also not necessary to place the leg on the ground, a mere kick behind or forward timed with the hand activation gesture will trigger the respective gesture easily.

VI. Ableton 9 Live - OSC Instrument

Figure 7. An OSC Instrument in Ableton 9 Live.

For Ableton inputs mentioned ahead, the configurations for a Ableton Live Set have been saved in the project file along with the code. These settings are auto-loaded when the project file is opened. It is IMPORTANT to press the global “play” button in Ableton; failing to do so will result in the looper not working.

Figure 7. shows an OSC Instrument. The “OSC Name” is the string value of the instrument track this OSC packet is connected to (and controls). The pitch, velocity, duration of a particular note is calculated and sent accordingly to Ableton via the Visual Studio code. Figure 8. shows the internal working of the OSCInstrument using Max 6.0.

VII. Ableton 9 Live - Looper OSC

The Looper OSC is the device responsible for relaying the loop control messages to an Ableton

looper. Each track has its a looper associated with itself, which performs the four basic loop control tasks. The “Track” field specifies which track the associated looper is connected to.

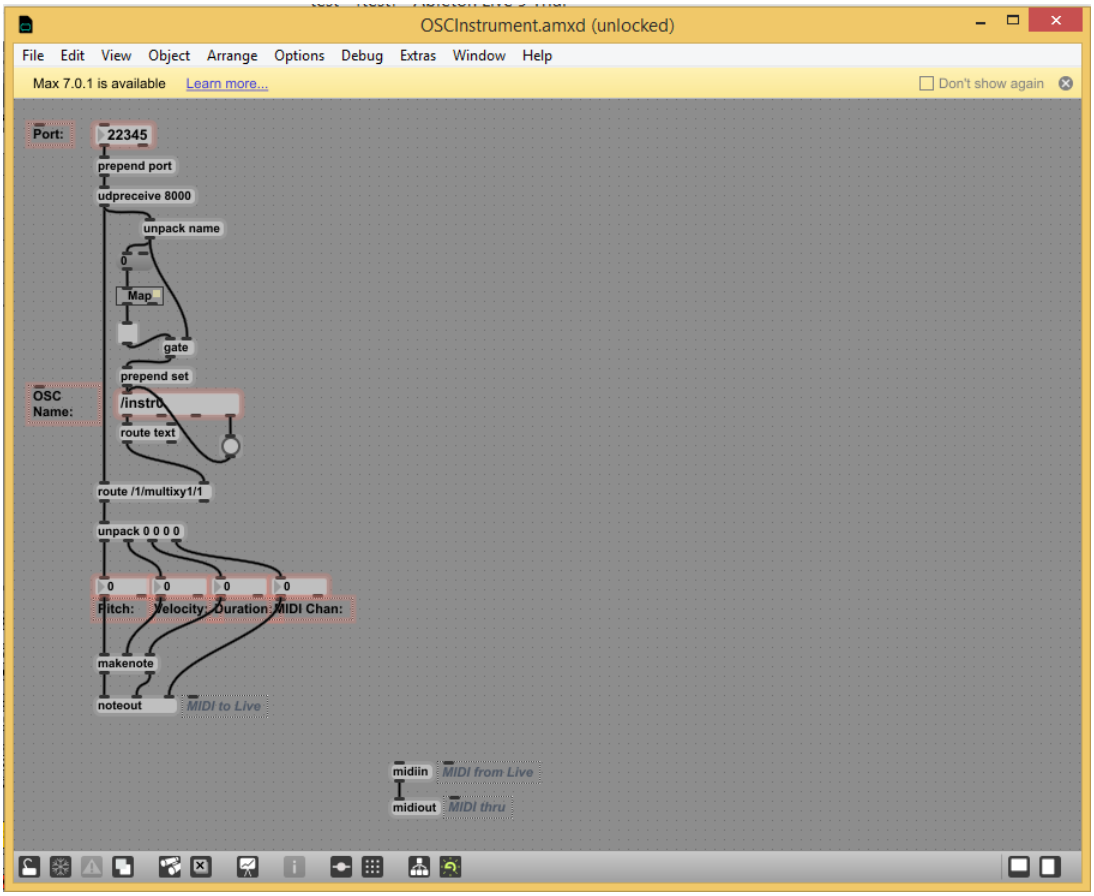


Figure 8. The working of OSC Instrument in Max 6.0

The string below “OSC In: Port”, in the form of “/Looper/looper_number/State action_name” denotes the loop control notifications of the user via the Kinect and the Kinect SDK being received. The string below “OSC Out:Host Port” in a similar form as mentioned earlier denotes the message outgoing to the connected Ableton Looper, which actually performs the required loop control tasks on the corresponding instrument.

Figure 9. LooperOSC to relay messages from the SDK to the Looper in Ableton 9 Live.

VIII. Ableton 9 Live - Looper

Figure 10. The Ableton 9 Live Looper

This device is responsible for performing the actual task of recording, playing, overdubbing and stopping a loop on a track. The outgoing message from the Looper OSC relays commands to the looper to execute these 4 tasks according to user gestures. Quantization should be set to "None" to avoid undesirable effects during play.

4. Implementation

4.1 Dictionaries and Libraries

Dictionaries of Ableton slider controllers and switch controllers were necessary. The slider controllers contained elements such as pitch and volume and can be fetched by their names such as “instrument/pitch”. This is similar to the switch controllers that contain elements that are turned on and off rather than controlled on a sliding scale, like play, which can be fetched in by its name “instrument/play”. We also used the Microsoft Kinect SDK dictionaries to read joint types.

4.2 Hardware and Software

All the skeletal tracking was taken care of using the PrimeSense hardware that comes installed in the Kinect itself. The Kinect SDK enables the creation of applications that support gesture and voice recognition using C++, C#, Visual Basic, or any other .NET language or Windows Store projection. There is access to full source code, Kinect Studio, and resources to simplify and speed up development [5]. We used Microsoft Visual Studio 2013 to run the Microsoft Kinect Body Basics with Kinect SDK 2.0.

4.3 Language

C# was the language used for implementation, since the code was built on the existing Microsoft Kinect Body Basics with Kinect SDK 2.0, which is implemented in C#, as it is already supported by the official Kinect SDK and Microsoft when it comes to application development for the Kinect when using Microsoft's Visual Studio 2013.

4.4 Environment and Miscellaneous.

One important aspect to be noticed is the performance can get severely affected by noise in the area. For best results, it is ideal to have just one body tracked, and as low noise as possible. Noise present in the surroundings hampers the confidence values of joints at times, leading to flickering, which can cause unwanted increases in frequencies of notes being sent.

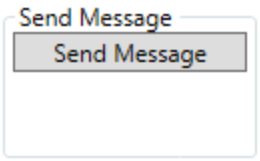


Figure 11. The Send Message button

The Send Message button was added to aid in debugging and checking if messages were being sent to the mentioned IP address, without having to use a Kinect.

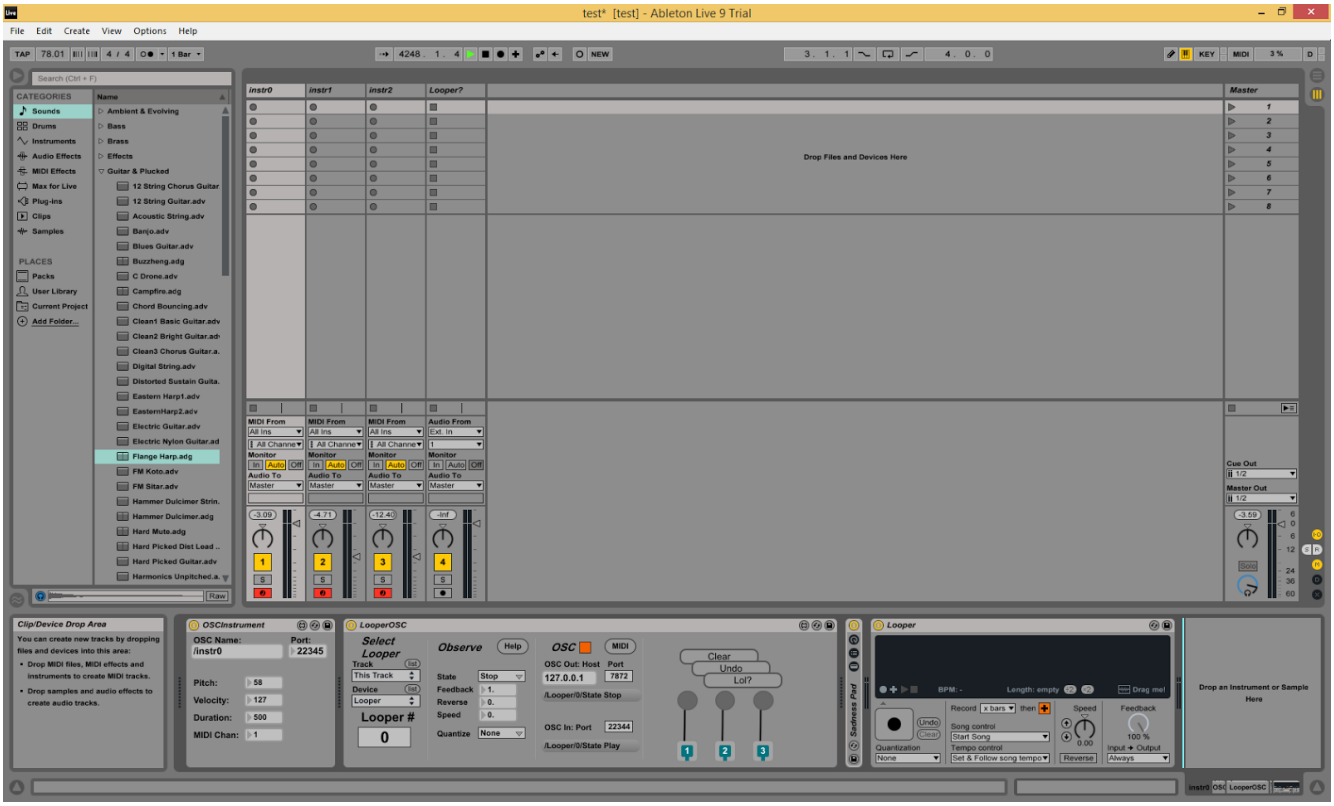


Figure 12. Ableton 9 Live set up for the project

Figure 12. shows how the UI on the Ableton side looks like. The loop control effects will be seen in the "Looper" device in the bottom right.

Figure 13. The global play button in Ableton 9 Live.

NOTE: It is important to press the global play button in the top of the screen to make sure the Looper works. (The Play button can be seen in Figure 13. in green)

4.5 Implementation of Drums

An implementation is also available to play drums by using punches to signify playing a beat. The gestures were as follows;
Punches to the side - Snare
Punches upwards at around a 45^{o} angle - High Hat
Punches downwards at around 45^{o} angle - Bass
However due to technical difficulties, the implementation was not feasible for performance (Explained in Section 6)

5. Results

Working with our musician, Matthew, we were able to create a basic set up and implementation of gesture based music generation using a single connect that communicated with Ableton 9 Live. Demonstration of this application is doable with a Microsoft Kinect for Windows v2 Sensor and Ableton 9 Live. Using input from the Kinect regarding the position of the user's projected skeleton and the gestures the user is making, we are able to output an actual, live, gesture-based musical performance.

Currently multiple recordings of simple melodies and rhythms were achieved using the system. Songs like Frere Jacques, Happy Birthday, Twinkle Twinkle Little Star were successfully performed, with the performances getting more complex as the musician gets better used to this style of play.

6. Issues

There are quite a few issues with the current initial release of the system. Some of them are discussed as follows;

One of the major issues was with the implementation of drums. The Kinect has a latency of 60 ms by default. Since drums decide and severely affect the tempo of a song, a consistent latency adds up to a significant time delay in the performing of the gesture and hence registering of the action, and eventually in the musical output. Over time it became very difficult to synchronize beats with the rest

of the instruments due to the delay, due to which drums are not included in the current released version of this system.

The gestures for the loop control are a bit strict and require precise timing to avoid unwanted outputs. Refinement in the checking of the gestures can help improve this issue significantly.

Currently once an instrument is set using the Instrument Selection gesture (Not the Default gesture), changing the partitions should result in the partitions getting reset, which however, is not the case. It is required to close the Body Basics Window and run the program from Visual Studio again

Once all loopers have been used and have some data in them, to start over again, it is necessary at times to manually stop and clear the loopers. This can be fixed by a global clear/reset signal and/or function.

7. Updates and Future Works

Functionalities that might be added on in future releases are as follows;

- Adding a tutorial mode to help a user to get accommodated with the system.
- Allow option to enable/disable visual cues. For example, a visual depiction of the piano on screen to help the user know which note is being played, instead of having to guess.
- Addition of a beginner mode and a advanced mode.
- Refined instrument selection to allow more instruments and accommodate the same in the UI.
- Resetting an instrument for a particular partition that has already been set.
- Check possible alternatives for implementation of playing drums.
- Adding a global Looper clear and reset which works on all existing Loopers together.
- Refining the gestures for the loop controls to allow more freedom.
- Addition of *clear* and *undo* commands for the loop controls.
- Adding at least 2 to 3 filters (effects) for an instrument being played.
- Increase size of staging area by using multiple Kinects, if feasible.
- Look into adding filtering techniques to increase joint confidence values, allowing for more complex gestures which might result in a better visual performance. For e.g. a pirouette.

8. Conclusion

While the subject of gesture based music using the Kinect's method of markerless motion capture had been broached before, we brought a different approach to it by using a divided stage to create more options for the user. We also implemented a system for looping: the user would initialize a loop, create the loop, and then close it. This loop would then be carried through as the user moved into a different section of the partitioned stage to take up a different instrument or beat. This frees the user

and allows for more variation in a piece of gesture based music.

Another thing to look into is the feasibility of drums: during this project, it was something we sought to work on, but the delay between the user's motion and the respective beat did not allow for a usable method of drumming and creating a beat. This would be a dynamic addition to the project, and something that could be looked into in the future.

Another thing to look at could also be the addition of another musician, or possibly and ensemble. We have partitions created for the user to construct their own ensemble, but the capability for multiple users is already there, so it could be used. A fully gesture based ensemble could be an interesting thing to attempt, as the multiple users would pose a challenge. This would also warrant the use of multiple Kinects to increase the capture space, something that we did not do here because it was not necessary and the occlusion problems that would have been more of a hindrance than anything; one subject can easily be handled by a single Kinect and the capture capability of the Kinect v2 was already sufficient and robust enough for our purposes.

In short, there were things we attempted in this project that were not fully possible within the limits of the technology and things we did not attempt because of the single nature of our musician. With the Kinect's markerless motion capture and its ability to accurately capture multiple subject, the possibilities for expansion and improvement on this project are many and varied.

References

[1] Jinxiang Chai. A Brief History of Motion Capture. Texas A&M University Computer Science. 2012.

[2] John MacCormick. How Does the Kinect Work?. Dickinson College Math/CS. September 2011.

[3] Tamara Berg, Debaleena Chattopadhyay, Margaret Schedel, Timothy Vallier. Interactive Music: Human Motion Initiated Music Generation Using Skeletal Tracking by Kinect. November 2011.

[4] Michael Sherman. Body Pose Tracking and Instrumented Personal Training using Kinect. Capstone Project Report for BSEE. Rutgers University. 2012.

[5] Kinect for Windows. Technical Documentation and Tools. Microsoft. 22 October 2014.