VIRTuoso

Final Report

Evelyn Fifi Yeung & Lucas Tai-Macarthur

Introduction

Motivation

Becoming a classically trained conductor is a long and difficult process involving many years of academic study and practical instruction. It relies both on a wealth of historic knowledge about music as well as the ability to translate that knowledge into voiceless direction for many dozens of musicians. Without an orchestra to practice with, it is difficult, if not impossible to practically apply the skills learned. It takes a long time and a lot of human effort to convene an orchestra for this purpose, therefore, it would be advantageous if there were a system that would allow a conductor to practice or sharpen their skills without requiring other people. Our project fills this gap by creating a method by which a conductor may both practice, as well as receive auditory feedback completely autonomously.

Objective

In this project, we have developed a virtual reality application that allows users to interact with a piece of music in a novel way. We use the virtual wall setup created by XLab in order to both collect user input and output the music and a user interface. Users are able to control a baton in the style of a conductor in order to change the tempo of the music and the volume of specific orchestral sections. Beyond the application, there are several components that could be extended through the gesture recognition system and the embed system to create custom gestures or haptic responses.

Timeline

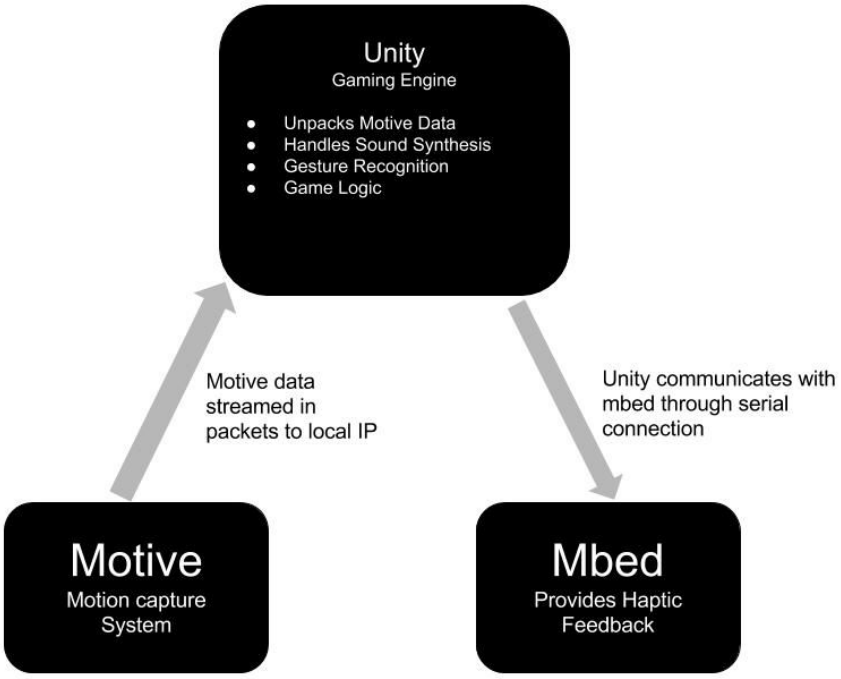
- Week 1: Fixing up the Virtual Wall and exploring Unity
- Week 2: Working through sound synthesis to achieve smooth transitions between tempos
- Week 3: Learning Unity and creating a baby demo of the sound features. Set up streaming from Motive to Unity
- Week 4: Wrote gesture recognition system and integrated with the sound features

Software Components

Components

1. Optitrack Motive system
    a. Handles tracking of rigid bodies through camera system
    b. Gathers position and orientation data by tracking centroid of a “rigid body”, which is 3 IR reflectors in a particular shape
    c. Can export data to other sources by streaming either to local IP or other IPs on network
2. Unity
    a. Unpacks Motive data into coordinates and orientation using quaternions
    b. Collects user input and processes the various outputs
        i. Feeds position data into a gesture recognition system
        ii. Sound features: volume changes and tempo changes
        iii. GUI to better interact with all the orchestral sections and receive visual cues
        iv. Communicate with MBed over serial
3. MBed
    a. Master mbed receives signals from Unity and sends commands to slave mbed on the haptic glove

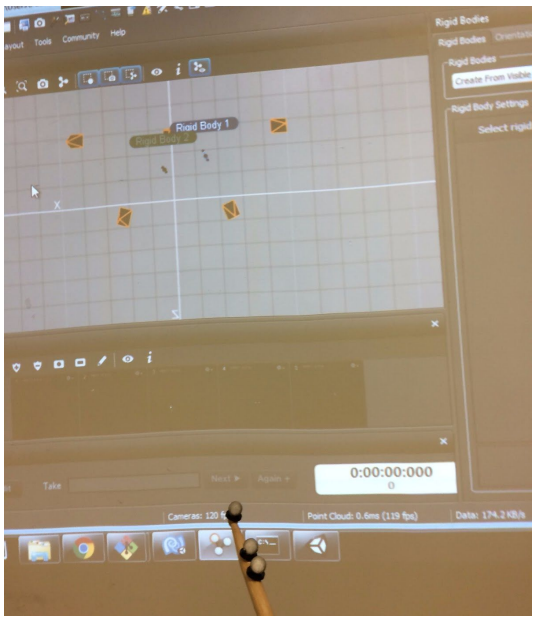
System Diagram



OptiTrack

In Optitrack's Motive software, we set up to track the conductor's baton. First, a user must calibrate the system using a specific calibration wand, which is a wand with 3 IR reflectors at a specific spacing. Optitrack provides a very easy interface to help with calibration and ensure quality results. Calibration also allows you to set a ground plane, which is important to make sure the rigid body moves in a cartesian space consistent with the user's position.

Once we have calibrated the system, we can set it up to track rigid bodies. A rigid body is created by spacing 3 IR reflector balls in a particular shape. Optitrack will track the centroid, and you will be able to see your body in the software. In this figure you can see the user's baton being tracked on screen.



After the rigid body is tracked, you can set up data streaming. We followed the InVIRT tutorial very closely to get this set up. Their project can be seen here: http://jiteshgupta.github.io/InVIRT/

We set up to stream to the local IP, then ran an additional script that unpacks the Motive IP packets so Unity can use it.

Matlab

Matlab is used to preprocess the audio files so we can achieve smooth tempo transitions. It is difficult to achieve tempo changes without also changing the pitch. This is a feature that is not native to Unity, so we do it in Matlab. We use a script that takes in sound samples (one for each channel representing an instrument in the orchestra), and produces however many different tempo tracks that you can specify.

This is done using a Phase Vocoder. A phase vocoder is a routine that modifies a signal in the frequency domain using fourier analysis so the time can be stretched, and the pitch is rescaled to match any time stretching.

http://labrosa.ee.columbia.edu/matlab/pvoc/

Ideally this would live inside Unity or we could call this resampling dynamically to achieve any speed. However, we could not find a way to interface Matlab and Unity to perform this function due to the way Unity requires assets to be set up.

Unity

Unity is a powerful game engine that allows users to create complex applications.

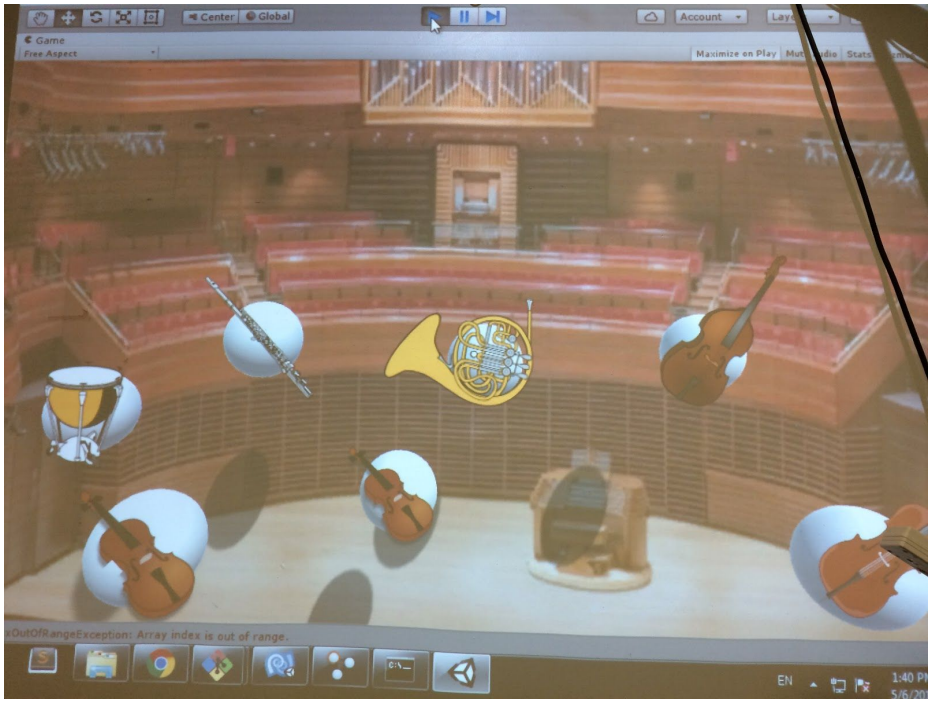
1. **RigidBody.cs -** We use provided files from the InVIRT project to attach the Motive data to a specific body in the project, in this case the baton. This specific file only moves the baton's position.
2. **Instrument.cs** - Each "instrument" has it's own behavior. It can be selected, which causes it to turn a different color - an important user feedback feature. Once selected, the volume can be changed. This file also handles switching tempos. In order to switch tempos, we simultaneously play all the tempo tracks at the same time, but mute all but the active one. We time scale all the tempo tracks so that they actually line up in time by using the **pitch** parameter. Therefore we can increase the pitch of slower tracks in order to make it play at the same speed as the normal track. For example, a track that plays at .8 the normal would have a pitch of 1.25 so it plays at the same speed as the 1.0 speed.

Therefore, we can achieve smooth transitions between speeds by muting and unmuting active channel, and scaling all the other tracks accordingly.

3. **Wand.cs** - This handles the gesture recognition and sound synthesis. Processing position data is inherently noisy, so we do a fair amount of smoothing to get the most accurate data possible.
    a. First, we get the direction the wand is moving by checking changes along certain coordinates. For example, if the wand is moving right, the x value would be greater in the next timestep, and y value should not be significantly different. Also, we require a certain speed threshold for it to register as a step in that direction, which decreases the amount of noise.
    b. The directional data is smoothed over multiple samples, like a window. The user can adjust the amount of smoothing, but in our demo, we take the majority direction of the last 4 samples and output that as the current gesture.
    c. In order to detect the tempo, we seek to find direction changes in the baton. Similar to the last, we do a lot of smoothing here. We take another "window" of customizable length (here 40) of past gestures and split it into two halves. We take the majority gesture of the first half and compare to the majority gesture of the second half. If they are different, we have a transition. A length of 40 was determined to work very well because a direction change tends to slow down a lot at the edges and cause a lot of "NONE" gesture types since it appears the baton isn't moving. A large window is required to accurately get real data from the transition.
    d. The time of each direction change is recorded and used to determine current BPM. Every measure (every 3 or 4 gesture changes depending on the defined time signature of your piece), the last few gesture change times are averaged to find the current BPM. The speed of the song is also defined, so you can determine the ratio of your BPM to the actual, and scale the song accordingly.
4. **SerialComms.cs** - This is the interface to the embed. You can customize this depending on how you program the embed. Here, we can write bytes to tell it to vibrate, or light up LED's on your hand for which instrument should be played
5. **GameLogic.cs -** For the last portion, you can do "Guitar Hero" style playing. This file handles that logic by allowing you to specify specific actions that should happen on particular instruments. You can specify a crescendo, decrescendo on an instrument, or a

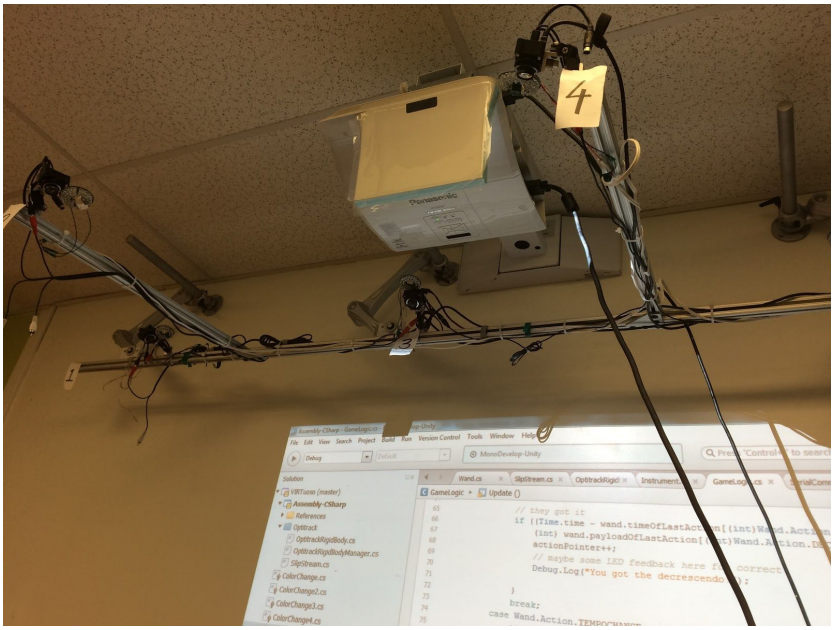
tempo change. You can also do this from the main Unity screen by modifying the array directly. This file handles the logic for when the actions succeed or when they fail.

In Unity, we also tried to make the user interface pleasing.



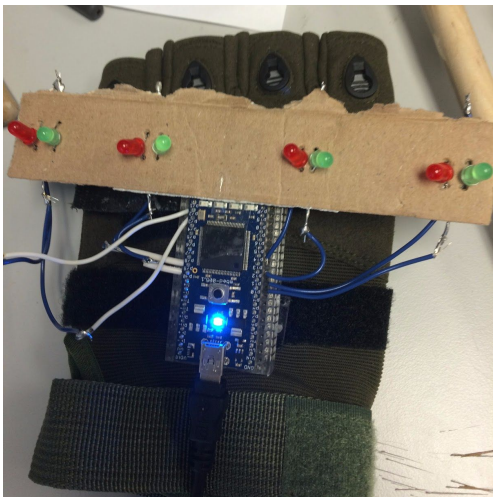
This way, users can point at specific instruments and interact with them. The spheres behind the instruments change color on select, and the instruments grow and shrink proportional to their current volume.

Hardware Components



The main hardware component is the VR Wall. This consists of 5 greyscale cameras, set up to see infrared light. Attached to these cameras is an array of IR LEDs. This way the cameras can easily see the IR reflectors attached to the rigid body.

There is also a projector so the user feels more immersed in the virtual experience.



We also created a haptic feedback glove. This glove includes a buzzer motor that we can trigger from Unity, and this LED display for the “conductor hero” play mode.

Results and Future Work

The project was successfully able to create an immersive conductive experience for a user. We were able to make it intuitive and interactive, with our users reporting high levels of fun attained from playing the conductor game.

The haptic feedback was not at the levels we had hoped for when we first started the project. That portion could be expanded to include additional cues beyond just the glove. We were unable to brainstorm haptic features that would actually be interesting or make sense for the application. One extension we conceived of was also being able to tap your foot to set tempo.

Also, it is a big disadvantage to have to process the audio files in Matlab. Given more time, I would have liked to try doing the tempo changes dynamically in Unity. It would be significantly more difficult given that MATLAB has vast numbers of built in functions that help with digital signal processing. The import process to change songs is rather tedious and cumbersome, so that should definitely be streamlined.