Tutting Dance with Flex Sensors and IMU

Amr Mohamed*, Lucas Lin†
EE267 Final Report
Instructors: Gordon Wetzstein and Robert Konrad

Figure 1: *Tutting Dance*

Abstract

In this report, we explore an exciting application of virtual reality technologies. We create a virtual environment where agents can dance through avatars. Agents can learn new dance routines by mimicking another avatar and getting feedback. Due to time constraints, we decided to focus on tutting dance, a form of dance that involves the use of the body to make precise geometric shapes. Our enterprise is more modest yet for we focus on tutting dance routines using only the forearm, wrist, and hand.

We use IMUs and flex sensors to track the wrist and forearm movements. In addition, we record the movements and angles, compare them to the correct movements, and provide feedback to the dancer. The result is a virtual environment where a dancer can learn new dance routines as accurately as possible and receive feedback, or follow accessible dance routines just for fun.

Keywords: Virtual Reality, Tracking, Dance, Tutting, Hip Hop

1 Introduction

1.1 Dancing Avatars

In 2003, Second Life was brought into the world. Second Life is an online virtual world with approximately 1 million regular users and a GDP of $64 Million. A large percentage of the time spent in Second Life is spent dancing. In addition, a sizable portion of the Second Life economy depends on avatars dancing. The popularity of virtual dancing in Second Life as well as the commercial success of dancing games such as Dance Dance Revolution and Ubisoft's Just Dance hints at an undeniable fact: millions of people enjoy dancing through virtual avatars.

For many years, people have been learning to dance by watching other dancers and mimicking them. However, it is really hard to get the dance routines correctly and accurately without feedback from an instructor or taking expensive lessons. In this project, we explore an alternative solution. We exploit virtual reality technologies to allow people to enjoy dancing through virtual avatars and receive personalized feedback on their dancing style.

1.2 Tutting Dance

In this project, we started by focusing on tutting dance. Tutting is a form of dance that involves the use of the body–usually the arms and hands–to create geometric shapes and usually with right angles. This style of dance is usually focused on making sure the shapes created by the limbs are as precise as possible. Additionally, these motions generally are done to the beat and each position quickly flows from one position to another. Tutting has its roots in hip hop, street dance, and funk styles from California during the late 1960s-1970s.

1.3 Goals

The goal of our VR project is to develop a hand/arm tracking program to teach the dance style of tutting by mimicry. This includes creating a virtual environment where people can improve their dancing or dance just for fun.

2 Hand Angle Tracking

2.1 Related Work

Previous work has seen the use of inertial (and magnetic) measurement units for hand tracking and rendering. One example of this work has used a glove outfitted with six IMUs to track finger and hand positions.1 It has been noted, however, that use of IMUs to track position leads to drift, so we decided to design our approach on hand and arm angles rather than absolute position. Additionally, we use flex sensors for further disambiguation of wrist angle vs. hand angles. These flex sensors vary their resistance based on their bend and have also been used in a glove construct to track fingers.

2.2 IMUs and Flex Sensors

We use the Spectra Symbol flex sensors to keep track of the wrist angle and the IMU on the back of the forearm to calculate the angle

*Department of Mathematics, Stanford University
†Department of Computer Science, Stanford University

of the elbow length-wise rotation of the forearm. Since flex sensors only vary resistance reliably in one bend direction, we use two flex sensors in conjunction on the back of the wrist to obtain both the forward bend as well as backwards bend. The circuit used for the flex sensors is a simple voltage divider and we use pre-recorded calibrating values to estimate the degree of bend from the input voltage from the divider. As for the elbow angle and forearm rotation, we transformed the quaternion from the IMU into Euler angles and then calibrated the values based on the maximum and minimum rotation of the arm/wrist.

2.3 Flex Sensor Application

In our device, we used the flex sensors to try and capture the angle the wrist is being bent at any time. We could have used an IMU but that would require a separate IMU from the one on the forearm and further disambiguation of angles as we would have to take into account the current whole arm rotation in order to translate the IMU data into correct wrist motions.

We originally planned on using a single flex sensor per wrist, but physically the sensors we are working with only give meaningful information when bent in a single direction. Looking at the figures below, we see that flex sensors work by having the conductive layer that is coated on one side be stretched or not depending on the bend, and that is what causes the change in resistance (resistances indicated on the figures are not equal to our flex sensors).

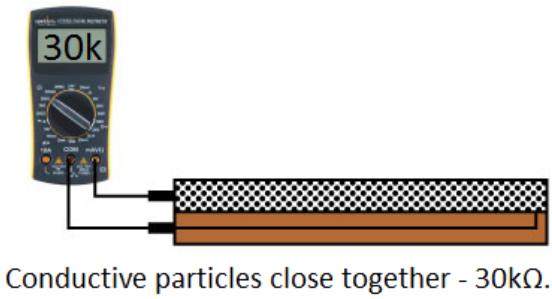

Figure 2: *Illustration of a straightened flex sensor*

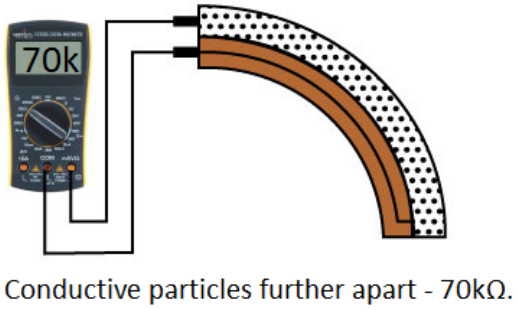

Figure 3: *Illustration of a bent flex sensor*

Since the flex sensors vary their resistance based on their degree of being bent, we hook the sensors up in a voltage divider seen in the next figure. Voltage dividers are a passive linear circuit that produces an output voltage as a fraction of the input voltage based on the ratio of resistances between the first and second resistor. Since the flex sensor is essentially a variable resistor, we can use the voltage divider circuit and have the Arduino take in the voltage of the voltage divider output. This voltage will be varying based off of the current resistance of the flex sensor, and thus, we can estimate the angle of the flex sensors using this voltage measurement using Ohm's Law.

As you can see from the figure, we create two separate voltage dividers–one for each flex sensor–and direct each of the output voltages into one of the GPIO pins. We read the ADC value from the respective pins, convert these values to voltages, and then apply Ohm's law to obtain the resistance of the flex sensor at that point in time. Then, using resistance measurements of the straight state and ninety degree bent state, we use the Arduino map function to estimate the angle that the current resistance corresponds to.

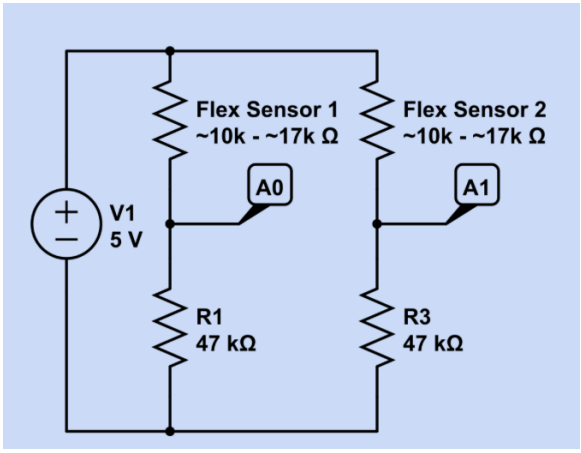

Figure 4: *Voltage Divider Used*

2.4 IMU Application

While we used the flex sensors to track the angle of the wrist, we still needed to be able to obtain the elbow angle and the whole arm rotation to cover a natural range of motion of the forearm. Naturally, rotation about the axis of the arm itself is impossible for the flex sensor to detect, and there were not enough flex sensors to also cover the elbow region (not to mention the fact that this would then have to be a separate section of the device as well). So, we used the IMU to estimate these metrics.

By placing the IMU on the forearm behind the wrist as seen in Figure 5, the IMU is in a good position to track the motion of the forearm without being confounded by the additional movement of the hand/wrist as well.

We translated the data obtained by the IMU to usable angles that can be manipulated in Unity in the following manner. First, we obtained the rotation quaternion with the complementary filter using the same method used for the last two labs. After sending this data to Unity via serial port, we transformed the quaternion into its Euler angle representation with the native Unity method (Quaternion.eulerAngles). We then used simple linear transformation functions to map these angles that we got from the quaternion to the correct rotation of the virtual reality first person arm model.

3 Translation of Reality to VR

3.1 18 Dance Positions

To easily distinguish the hand angles and hand positions, we decided to limit the dance movements to 18 different positions. These 18 positions are the result of combining the following:

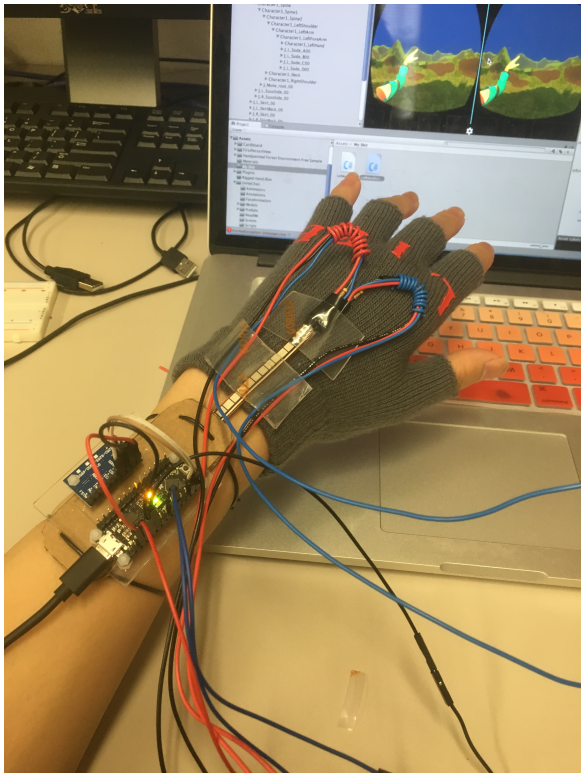

Figure 5: *Progress: Sensors and IMU on the device*

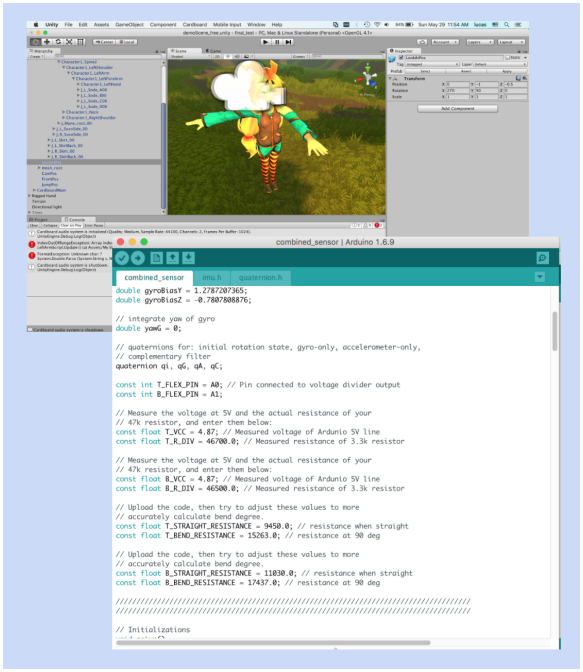

Figure 6: *Progress: Integration with Unity*

2 elbow positions: arm extended so that elbow is at 180 degrees, and arm bent so that elbow is at 90 degrees.

3 wrist positions: hand facing the avatar, wrist straight, and hand facing away from the avatar.

3 elbow rotations: elbows rotated so that palms are facing the avatar, elbows rotated so that palms are facing each other, and elbows rotated so that palms are facing away from the avatar.

Combining these positions together gives us $2 \times 3 \times 3 = 18$ positions that are different enough to be easily distinguished.

3.2 Difficulties

Working with both the flex sensors and the IMU in conjunction with the Unity-based virtual environment, there was a lot of communication between hardware and software that needed to be correct for the entire system to work. From serial port connection issues to problems using the received data in reference to the virtual reality model, there were more than a couple difficulties that we encountered when trying to assemble the entire system.

3.2.1 Delay and Multiple Ports

After capturing hand motion and integrating with Unity, we experienced extreme delays between the hand motion and the response from the avatar. At this point in time, we were using one Arduino solely to collect information from the IMU and stream it and one Arduino solely to collect information from the flex sensors.

After investigating the speed issues we were having, the bottleneck turned out to be the switching between ports to get different readings from flex sensors and IMUs, which were on the different Arduinos. We decided to use the same port and same Arduino for flex sensors and IMU. While this makes the processing onboard the Arduino a little more slow, this lead to great speedups and almost real-time response on the Unity-side, which was the main problem.

3.2.2 Physically Possible Motion and Snapping

During the initial stages of testing the translation of measured angles to model joint rotation, some snapping of the model arm was observed. Looking into this, we found that this was due to the fluctuations of the incoming data as well as the fact that the returned angles often went below zero–becoming negative–or above 360–wrapping around back to 0–which was messing with some of the calculations we were doing with our linear mapping functions.

To solve this issue, we decided to limit the range of motion on the arm to what is normally physically possible. This was done by manually calibrating the range of possible angles to establish hard boundaries that the avatar can not exceed because they are physically impossible in reality. In other words, fluctuations in measurements that lead to arm/wrist angles that are not physically feasible are ignored. For example, if a value less than zero or greater than 200 (looking for wrap around of values going below zero) is detected, we assume that the incoming data is meant to be at a zero degree angle but is just a fluctuation, so we set these values back to 0.

4 The Virtual Environment

4.1 The Avatar: Unity-chan

Due to a lack of free but still properly rigged models of the arm and hand, we used the free Unity asset ”Unity-chan” as the user avatar. ”Unity-chan” is a cute 3D model girl from Japanese game scenes. By locking the skeleton of this avatar with the first person controller and warping the arm in front of the camera, we were able to create a first person perspective onto this model.

Figure 7: *Unity-chan*

4.2 Recorded Routine

In addition to having ”Unity-chan” as the user avatar, the virtual environment we created for the demo also includes a recording of a couple tutting routines that are designed for the user to follow along in virtual reality. The user is to follow the routine as best as possible and receive a metric on how accurate they were at the end.

4.2.1 Recording Process

To record a routine to be played later, we decided to store exactly the values that come into Unity before any transformations of the values occur of both the IMU and the flex sensors. For each line that successfully is read from the serial port, we append it to a text file that is created in the same directory as the Assets folder. This whole process is can be turned on or off by changing a variable–bool recording–that exists in the script file for the forearm.

4.2.2 Replay Routine Process

In order to play back the routine that has been recorded already, we set up an identical Unity-chan model in front of the user and create analogous scripts for the mirror’s forearm and hand that match the scripts for the user’s forearm and hand. Since we cannot simply use each line from the recorded values file on each update cycle on these elements (this was tried initially and we realized that it caused a sped up version of the routine), we decided to link the mirror to the user. So, whenever the script on the user successfully reads a line from the serial port and is going to alter the transformation on the user’s arm, it also calls a manual update on the mirror, causing the mirror to alter its state to the next line in the recorded routine.

4.2.3 Sound

Since music is a huge part of dance and also acts as an essential cue for beginning and keeping the user on the same beat as the recorded routine, we decided to record all the tutting routines in conjunction with the music and have the music start playing as soon as the user presses the return key, which also signals the tutting routine of the mirror to start as well.

Interestingly, because the recording of the routine begins immediately when the we enter the virtual environment after it has been loaded and that the serial port from the Arduino actually gives some garbage initial few lines due to abrupt starts or unflushed registers, the recordings are also initially not calibrated to start with the music. We ended up having to manually adding silent frames to the recording in order to get it to begin on the beat we wanted it to begin on.

4.3 Scoring

Given that the user’s goal is to learn a dance or in general practice some tutting, we decided to gamify the demo in order to produce a goal for the user and provide feedback metrics. To do this we developed a somewhat forgiving algorithm–as there is sometimes a lot of fluctuation for incoming data–to represent a user’s accuracy in following the recorded algorithm.

First, we use two thresholds on each of the degrees of freedom–elbow angle, arm rotation, and wrist rotation–to determine how different the value for the user and the value for the recording is at each update in the user’s motion. We separate between the ”perfect”–passing the smallest threshold, the ”small error”–passing only the larger threshold, and the ”large error”–passing no threshold. So, on each tick, a user gets three potential errors they can make. We then average these errors and scale them down using a value of 0 for no error, 1 for a small error, and 2 for a large error. If the scaled-down average is small enough to no error (a value of 0), we assume that the next four ticks are also going to be correct. So, the cumulative percentage is just 100 percent minus the 0.1 times the percentage of small errors and 0.5 times the large errors. This also means that the worst score possible is 50 percent.

4.3.1 Scoring Visual Representation

We wanted to give users real-time feedback to they would be able to track their progress throughout the routine and perhaps understand which section the user messes up on. So, we generated a 3D block in the virtual environment and then attached a TextMesh that is updated every time the score changes.

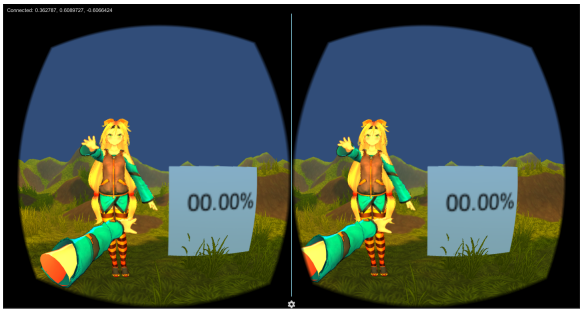

Figure 8: *Virtual environment rendered with all components*

5 Results

We were able to successfully use the data from the flex sensors and the IMU to control a rigged arm model in Unity. In order to prevent some snapping of the model arm due to potential fluctuations of incoming data–like a value of 360 degrees wrapping around to 0 or 1 degree–we limit the range of motion on the arm to what is normally physically possible and typically used in a tutting routine. One major concern about the current algorithm is that it is fairly dependent on the manual calibration being correct as the degree bends of the various angles are estimated using base values that we measured, so a future improvement could be the automation of calibration at the beginning of each demo round.

6 Discussion

6.1 Use of Flex Sensors

Initially, we were planning on using IMUs for all the angle measurements. After this prototype, though, we feel that the combination of the flex sensors with the IMU is probably a more robust design than the pure IMU design. Since there was little interference between the flex sensor side of the measurements and the IMU side of the measurements, we were able to focus on optimizing each specific angle measurement, most likely making the overall device better to use.

One of the limitations of the flex sensors was the fact that it could only reliably obtain bend information between zero and ninety degrees for one side. However, this is remedied by using two flex sensors back to back. Unfortunately, we did not have enough time to stress test the flex sensors as it had been mentioned before that continuous use–especially in the direction opposite of which it was measuring–could potentially alter the properties of the flex sensors. We did not explicitly see any changes, but the tests we put it though were not particularly rigorous and the device was not used that many times in total.

6.2 Use of IMU

Between the flex sensors and the IMU, we felt that the IMU was slightly more unwieldy and difficult to work with. As it was stated before, using only the IMU to do positional tracking is fairly difficult and drift sometimes occurs as well. While we did not need exact positional data, the angular data we needed was similar in terms of requiring to maintain a certain angle as a certain measurement.

After several tries with the IMU when we learned what made the measurement device more finicky than other times, we were able to have the data translated to arm angles relatively accurately. What we had to do in order to make sure that the starting position was calibrated alright is to reset the Arduino once the program was running and the user had their arm in a stable starting position.

6.3 Future

There are a couple of features that we would have wanted to include in this project if we had the additional time and/or resources. First, since tutting is usually done with both hands at the same time, it would be interesting to see if the device/system built could be replicated for the other arm and then used in conjuction–some serial port communication issues would have to be resolved due to overloading probably. Additionally, the use of the flex sensors as calculating small joint angles could be extended even to the fingers with a full glove and a flex sensor per digit.

On the software side of things, a useful continuation on this project would be a calibration system so that the Arduino would not have to be set each time the program is run and that if the flex sensors start changing properties, it would not matter.

7 Conclusion

All in all, the device that we constructed using and Arduino, flex sensors, and and IMU was able to successfully track 3 degrees of freedom of the arm/hand area. Combining this with a movement playback function allowed for the creation of a tutting mimicry game.

Acknowledgements

Thanks to Professor Wetzstein for providing all the hardware. Thanks to our EE 267 peers for sometimes giving advice when we were stuck with a debugging issue.

References

BALDI, L., MOHAMMADI, M., SCHEGGI, S., AND PRATTICHIZZO, D. 2015. Using inertial and magnetic sensors for hand tracking and rendering in wearable haptics. *World Haptics Conference (WHC)* (June), 381–387.

JIMB0. Flex sensor hookup guide. https://learn.sparkfun.com/tutorials/flex-sensor-hookup-guide. Accessed: 2016-05-20.

SORAL, L., AND AMBIKAPATHY, M. 2013. Hand-talk gloves with flex sensor: A review. *International Journal of Engineering Science Invention 2*, 4 (Apr.), 43–46.

ZIMMERMAN, T., LANIER, J., BLANCHARD, C., BRYSON, S., AND HARVILL, Y. 1987. A hand gesture interface device. *ACM SIGCHI Bulletin 18*, 4.