Augmented/Virtual Reality Laboratory

Student: Benjamin Tully-Hanson
Computer Science/Physics, Junior
University of California, Irvine

Mentor: William Little
Aerospace Technologist, Software Systems
IT-C1
Information Technology and Communication Services Directorate

1 Introduction

Real time motion tracking hardware has for the most part been cost prohibitive for research to regularly take place until recently. With the release of the Microsoft Kinect in November 2010, researchers now have access to a device that for a few hundred dollars is capable of providing red/green/blue (RGB), depth, and skeleton data. It is also capable of tracking multiple people in real time. For its original intended purposes, i.e. gaming, being used with the Xbox 360 and eventually Xbox One, it performs quite well. However, researchers soon found that although the sensor is versatile, it has limitations in real world applications. I was brought aboard this summer by William Little in the Augmented/Virtual Reality (AVR) Lab at Kennedy Space Center to find solutions to these limitations.

2 Setup

When motion tracking research began in the AVR Lab, it became apparent that using multiple Kinects at a time would be beneficial. Attempts were made to attach multiple Kinects to a single computer, but due to oversaturation of USB bandwidth this approach simply did not work. Instead, a client-server architecture was implemented. Four Kinects are positioned in a square facing inward, each connected to its own computer designated as a client machine. The positioning can be seen in Figure 1. These four machines then transmit their separate Kinect data over the network to a server, which then processes the collection of streams and produces a single "resolved skeleton" that tracks the user.

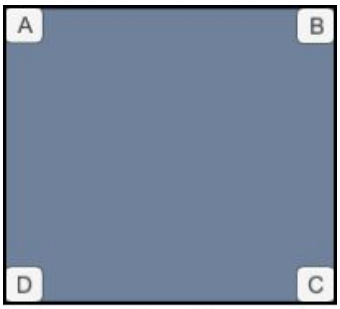

Figure 1

3 Limitations

There are three main problems that arise when researching the Microsoft Kinect's shortcomings: self-occlusion, bone-length variation, and artificial joint vibration. As a general rule, self-occlusion occurs when a joint on the human body is hidden from the Kinect's view by a different part of the body. Common examples include a user facing the Kinect and putting a hand behind his or her back, or turning 90° from the Kinect and therefore hiding either the left or right side of his or her body. In these scenarios, the joints that are hidden from the sensor change from

having a tracked state to having an inferred state. What this means is that although the sensor doesn't know the position of the joint, it will make an educated guess based on the anatomy of the human body. Unfortunately, this guess can be extremely inaccurate in certain positions.

Bone-length variation occurs when the distance between two connected joints on the human body changes. In the real world, the distance between any number of connected joints should remain constant. This problem is especially apparent when the user makes a swift motion, such as kicking out a leg or swinging an arm as if to serve a tennis ball. Various tests have shown bone-length to almost double in certain scenarios, certainly something that needs to be fixed.

Artificial joint vibration is slightly harder to pinpoint, and can be seen even when the user is standing still. When the Kinect slightly mistracks a joint repeatedly, the position can oscillate wildly between two or more positions. This oscillation is especially apparent in the joints of the user's hands. Even when standing still with the hands away from the body in an attempt to avoid mistracking, the hand points jump around and frequently bend in ways that are impossible for the human body to bend.

It is the AVR Lab's goal to be able to use a system of Kinects as a robust and affordable method of motion tracking. A number of applications are possible, including realistic training of astronauts, remote control of hazardous ground operations, and more efficient relay of information in video conference calls. However, the above listed limitations must be overcome for the Kinect to become a useful tool in real world use.

4 Joint Occlusion

When I started working with the Kinects this summer, the most glaring issue to me initially was the joint occlusion. If the user came close to a 90° angle to the sensor, the displayed skeleton barely looked human. I was given papers to read concerning this issue, the most notable being *Improved Skeleton Tracking by Duplex Kinects* [1]. This particular paper implemented a duplex-Kinect system, with two Kinect sensors at orthogonal angles to each other, in an attempt to resolve the joint occlusion issue. By combining the data from Kinects with different views of the person, the hope is to have a "resolved" skeleton that is more accurately tracked. I read through this paper over the course of a few weeks, implementing bits and pieces as I went, and eventually came to have an intuitive understanding of the algorithms outlined. Figure 2 shows

an example of the raw data given by the Kinect sensor, along with an example with my implemented algorithm.

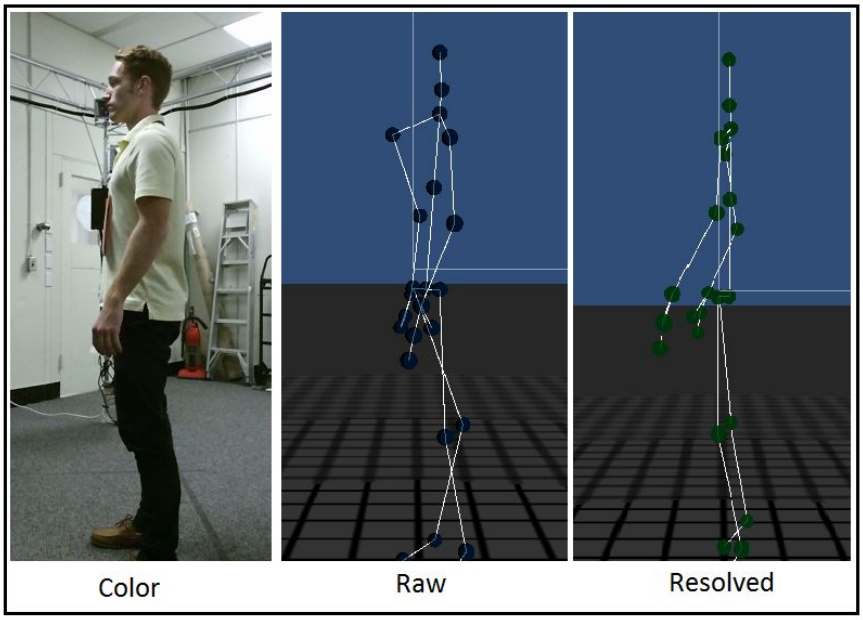

Figure 2

The general idea behind this algorithm works off of the concept of perspective distance difference between two orthogonal views. Say for example we have a person positioned facing Kinect A, as shown in Figure 3. Kinect A will have a view of the scene that looks similar to the image in Figure 4. While Kinect B, which is at a 90° angle from Kinect A, will have a view of the scene that looks very similar to the Raw image in Figure 2.

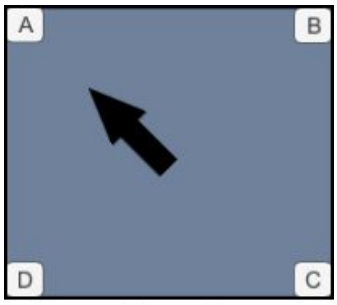

Figure 3

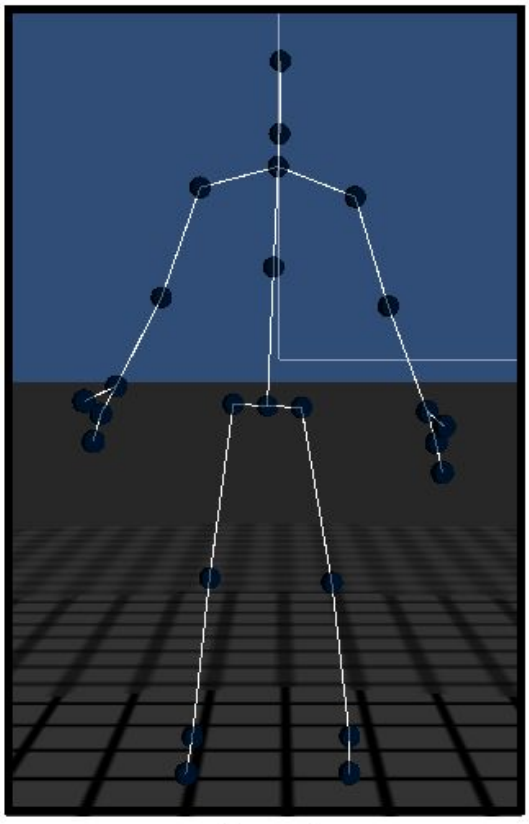
Figure 4

We first cycle through the joints for each respective Kinect, and find the joints with the closest distances that are not connected to each other by a bone. Whichever Kinect has the *smallest* distance, we then assume that those joints could have been incorrectly tracked by the Kinect sensor. Now that we have the potentially incorrect joints, we compare the joint distances in both of the views of A and B. Whichever Kinect has the *greater* distance, we assume to be more accurate as it is less likely that the joints were

tracked incorrectly. Following the above process using both the left and right hand joints as examples, one can see that the algorithm would then choose the hand positions of Kinect A as being more accurate. This same process carries throughout the rest of the skeleton, and the result is a resolved skeleton.

5 Bone-length Variation

After resolving most issues with the joint occlusion, I began to notice more clearly the changes in bone length. The hands and feet would repeatedly stretch in their length, and could even be seen doubling at some points. The general approach I took was to implement a "calibration stage" at the beginning of the motion tracking. In this stage the user would stand still for about two seconds while an algorithm calculated the distance between all the stationary joints. After completing the calculation, these bone lengths were then implemented as hard constants for the joint distances. At each frame, a unit vector is calculated that represents the normalized direction of the bone. From here all we must do is multiply it by its respective bone length, propagating this change down the limbs, generating a constant bone-length skeleton. To monitor the impact of my method, I constructed a table that monitors the value of each bone length. When a length changes more than a designated amount that rectangle flashes red. Figure 5 demonstrates the raw data from the Kinect without my bone-length constraint.

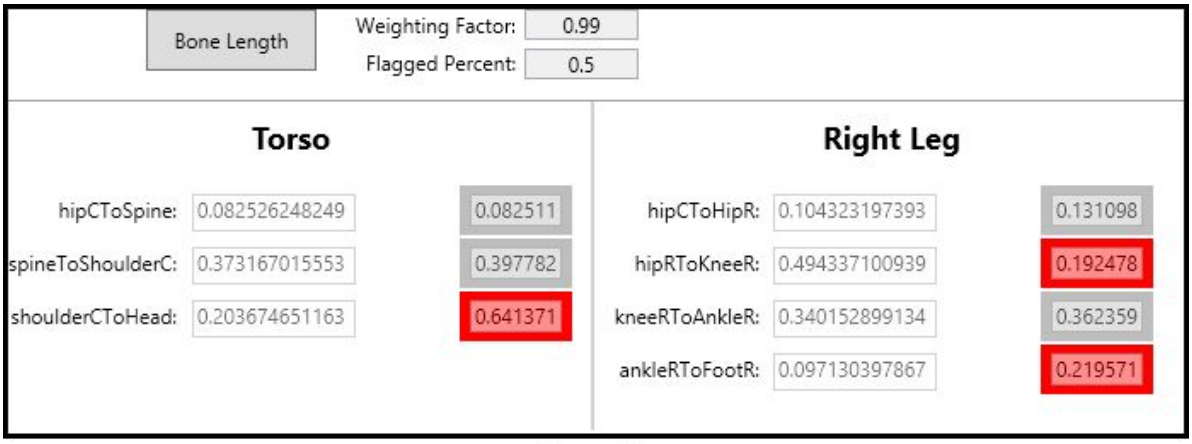

Figure 5

As shown, the bone length in this example changes more than 50% of the actual value. If the bone-length constraint is enabled, the resulting tracking is much more accurate. As Figure 6 shows, the bone length is constant with less than 0.00001% error.

Figure 6

6 Artificial Joint Vibration

Joint Vibration, or what we call jitter, is one of the more difficult challenges to overcome with raw Kinect data. There are a number of dampening algorithms that can be implemented but we walk a fine line of over dampening the data and introducing latency into the system, or under dampening and still having significant jitter. Although this is still an area of active development, my hope is to eventually use the depth data provided by the Kinect to reduce jitter. Depth data is inherently more stable than skeleton data, as instead of just having a single data point for a joint we have a large surface area of pixels upon which we can make calculations. By monitoring the number of depth pixels within a certain area per frame, we can use this as a second check to determine if a joint has truly moved or not.

Depth data is initially reported by the Kinect as a single value-per-pixel that signifies the distance in millimeters from the sensor. By filtering this data, I remove the background and leave only information about the user, achieving a green screen like effect. I then assign an RGB value to each pixel depending on the reported distance; the closer to the camera, the darker the color. Figure 7 shows an example of the resulting depth image.

7 Directional Tracking

Although implementing a multi-Kinect system is beneficial in providing more data points for accurate tracking, it also introduces a limitation of its own. Kinect sensors were built with the idea that the user would be facing them directly. As such they cannot distinguish if they are

looking at the user's front or back, and will always assume the former. Various research papers have made mention of being able to use a marker on the front of the user to distinguish it to the Kinects as the front. I had yet to see one that tested and implemented this method effectively, so I decided to do it myself.

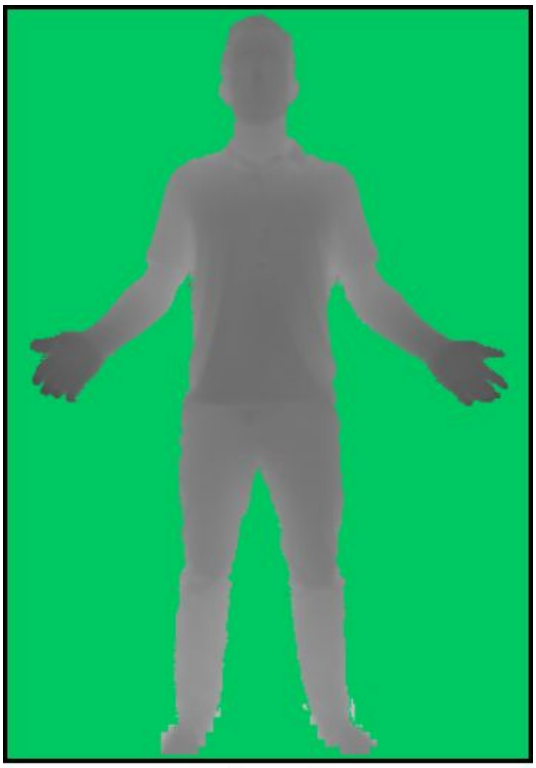

Figure 7

Each Kinect client is set up to pay special attention to the middle spine position on the user, while still transmitting the full body/camera/depth data to the server. It then pulls a square sample surrounding this position from the color stream, and begins to check the color of each pixel while also displaying it on the screen. If the user is facing the Kinect, the sampled area should be the color of the marker (in our case I used a red marker), and a notification rectangle will turn green as shown in Figure 8. If instead the user is not facing the Kinect, the sampled area will generally be a color other than the marker, and the notification rectangle will turn red as shown in Figure 9.

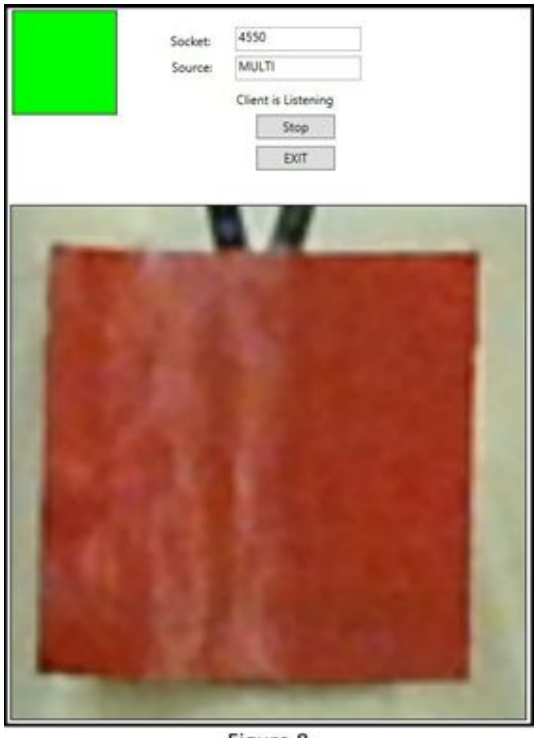

Figure 8

The specific values of the algorithm are modifiable by the user, but in general it calculates the percentage of the region that is within a range of red shades. This percentage is then reported back to the server, where it can distinguish based on the percentages exactly which direction the user is facing. This method is not only useful on its own, but can also be used to aid in more accurate

resolution of the joint occlusion limitation. A simple compass-like display was also created to show the direction of the user in real time, as shown in Figure 3.

Figure 9

8 Conclusion

This paper demonstrates methods to resolve common issues with the Microsoft Kinect motion sensor. By implementing the steps shown here, a multi-Kinect system can be used as a robust and cost-effective way to track human motion. Common issues that are prevalent in single Kinect tracking, such as joint occlusion and bone-length variation, have been resolved in the AVR Lab's setup. Future research will focus on ways to resolve the artificial joint vibration seen in real time motion tracking, as well as increase the fidelity of current motion capture techniques.

References

[1] *Improved Skeleton Tracking by Duplex Kinects: A Practical Approach for Real-Time Applications,* J. Yeung, T. Kwok, and C. Wang, The Chinese University of Hong Kong, Journal Of Computers and Information Division In Engineering, October 16, 2013.