Hand Tracking with Microsoft Kinect v2 and Leap Motion

Practical Course of Data Fusion

Hendrik Bitzmann
Hanna Holderied
Christoph Rauterberg
Maximilian Weß

Jun. Prof. Marcus Baum
Institute for Applied Computer Sciences
University of Göttingen
September 2016

Contents

CHAPTER 1

Introduction

The aim of this project of the Practical Course of Data Fusion was to control a 3D model of a coffee machine via tracking the hands of the user with different technologies.

The first one was the Microsoft Kinect v2 which was released in 2014 and comes with several features: an infrared depth sensor, a RGB sensor and 3D motion tracking. The last is especially important for this project. In general, the procedure is very easy: You place yourself in front of a monitor where you can e.g. see the coffee machine and a 3D model of two hands. By moving your hands you can interact with the devices displayed on the screen. Therefore the motions are tracked by the Kinect sensors.

The second device we tried to use is a Leap Motion, also with a build-in infrared camera. This is a small object you place in front of your keyboard and via tracking your hand gestures it allows you to control your PC, or like in our case a coffee machine. Unfortunately we did not have enough time to implement the usage of the Leap Motion completely, but we will share our few experiences anyway. We tried to use the Leap to increase the accuracy of the tracked motions.

In the end we want to find out whether it is possible to use and fuse techniques like this to create a similar user experience with virtual reality compared to real world interactions. This could e.g. be used to evaluate the usability of several products.

CHAPTER 2

Methods

2.1 Setup

Configuration of the Kinect v2

In the beginning we tried to run the Kinect sensors (Kinect v1 and Kinect v2) with Linux laptops (Arch Linux), but did not manage to get everything running and switched to a Windows 10 desktop pc. It is important to mention that you need to have an USB 3.0 port if you want to use the Kinect v2, otherwise it will not work. In addition you need moderate computation power. It turned out that a notebook with just HD4000 graphics is not suitable.

The first step for using the Kinect v2 is downloading the official Kinect 2.0 Software Development Kit from Microsoft for Windows. The SDK includes many useful tools and examples. Next step should be starting the Kinect Configuration Verifier from the SDK browser, which will check the system requirements. Every check besides for the USB controller should be successful. For USB controller a warning is probably not a problem.

Set up of the development environment

We choose a running start by following the tutorial from [???] . This gave us a working development environment consisting of Unity3D, Visual Studio 2015 and the Kinect SDK, already with an "Hello World" example ???. Unity3d is a game development engine. It is a good choice for our application, because of ease which simple tasks can be executed. The import of assets into the project can be achieved via drag and drop and also linking scripts for event handling and update mechanisms to this assets is handled via drag and drop.

(We imported the source code into Unity3D and Visual Studio 2015, as a programming language we have chosen C#. Unity3D is a software to ... and Visual Studio 2015 is an Windows IDE ... - more infos about the kinect v2

First of all, we needed models. Also got a model of a coffee machine from one of patrick harms students
- Started with cubes, found hands, used blender to change hands according to our needs
- added trigger events -> when mesh of a hand touched a button -> triggered the colouring of the button
- how to best imitate/substitute haptical feedback? (colors of the buttons when near/pushed)

- Windows PC with USB 3.0 for Unity and Visual Studio 2015
- Unity, Visual Studio, V2 desktop (Leap Motion SDK), Blender for Modeling hands

- diagram/picture of the construction

- What we stole from other people.
- Lightbuzz/Kinect-Finger-Tracking (https://github.com/LightBuzz/Kinect-Finger-Tracking)
- Kinectplugin for Unity: https://developer.microsoft.com/en-us/windows/kinect/tools
- Links zu Handmodellen
)

2.2 General Procedure

To control a coffee machine with our hands we first needed a model for a coffee machine and for our hands. We got the asset for the coffee machine from Patrick Harms. We downloaded the 3d hand model from [https://clara.io/view/3e6923db-407c-4e0c-a253-2f564dfcd152 ???]. The hand model has full stretched fingers, which isn't perfect for operating a coffee machine. Therefore we transformed the model to a hand with a pointing finger. We used basic transformation operations in Blender (Blender is the free and open source 3D creation suite.) for this. The export can be a bit tricky, because the exported model is not equal to the one you see in blender. For our case this meant, that it wasn't enough to mirror the hand in Blender and export it to get a right hand from a left hand. We also had to invert the normals.
For the right visual feedback we hat to track the hand movements. We used the skeleton information from the Kinect. These contains 3d coordinates from the palm, thumb, wrist and the tip of the hand, which should be the longest outstretched finger. To form a local coordination system for the hand. We used the palm and tip coordinates to form the forward vector. We used the palm and thumb coordinates for form a temporary side vector. The cross product of these two gave us the up vector. To obtain a orthogonal system we recalculated the side vector from the cross product of up and forward vector. Depending on the hand model it is also necessary to adjust for the left respectively the right hand by reversing some vectors. For the positional tracking we just used the positional information of the palm.
To operate the coffee machine we had to track, when the hand is touching the coffee machine. We pursued 2 approaches. The first is to track the distances from then hand to the buttons manually. Therefore we pull a the list of buttons and calculate the euclidean distance of the palm,thumb,wrist and tip to these buttons. We used this to give some indication how close the hand is to the buttons. For the actual button push we used the build in collision detection in unity. To make this work we had to add a Box Collider to the Buttons and enable the isTrigger property. We also added a Mesh Collider to the hands, as well as a rigid body. If the rigid body interacts with the Box Collider, then it will trigger certain events: onTriggerEnter, onTriggerStay and onTriggerExit. We wrote a script for the buttons which will colorize the button green onTriggerEnter and return to the original color onTriggerExit.

2.3 Problems

Frame Selection

The Kinect v2 switches between 15 to 30 frames per second, depending on the lighting conditions in the room where it is used. Sometimes there are single frames in which the Kinect detects the hands in wrong positions or alignments. That made our hand models jump or shake from time to time and it became very difficult to move the hand models close to the relatively small buttons like that.

Due to that we implemented a frame selection: We saved the hand positions of seven following frames and calculated the median of those. This made sure that single wrong detected hand positions did not make the hand model jump uncontrolled, but of course also made the reactions slower and more sluggish. We tried the selection with less than seven frames as well, but seven seemed to be a necessary amount to smooth out irregularities.

Another possibility we did not implement is to have a window of seven frames and build the median of it. With each new incoming frame this window could be pushed forward one more frame, so the overall displayed frame rate will remain the same.

Pushing the buttons

The function to colorize the buttons when a hand is coming close is quite buggy at the moment. For some reason the calculation which button is closer changes frequently. This could be due to difficulties to identify the hand to which the distance should be measured in the moment.
To make a more stable but equally simple solution we could add bigger Box Colliders around the buttons and colorize the buttons on the onTriggerEnter event.This wouldn't mess with the other function, because the onTriggerEnter event of the inner Box Collider would remain intact.

Unstable hand orientation

The hand orientation is unstable in certain situations. Firstly we had a problem when we got to close to the kinect sensor. In this case the orientation gets more and more unstable, which is kind of expected due to the working distance of the kinect. We also had the problem that sometimes the up vector of the hand changes sign and therefore the hand will do a 180° rotation around the wrist.

2.4 Experiences with the Leap Motion

As already mentioned above we just had little time to test the Leap Motion, but still we made some experiences with it. We installed V2 Desktop, a Leap Motion SDK with several predefined scenarios to test the leap and its features. This one is plug-and-play and easy to use: We put the leap in front of the keyboard and opened the software. After choosing a scenario, 3D models of your hands are displayed on the screen and the movement of your hands and fingers are tracked a lot more accurately than with the Kinect v2.

The only problem we spotted is when you turn your hands with the palm facing each other and the thumbs pointing up - so that the fingers closer to the leap hide the other ones. Some motions are not

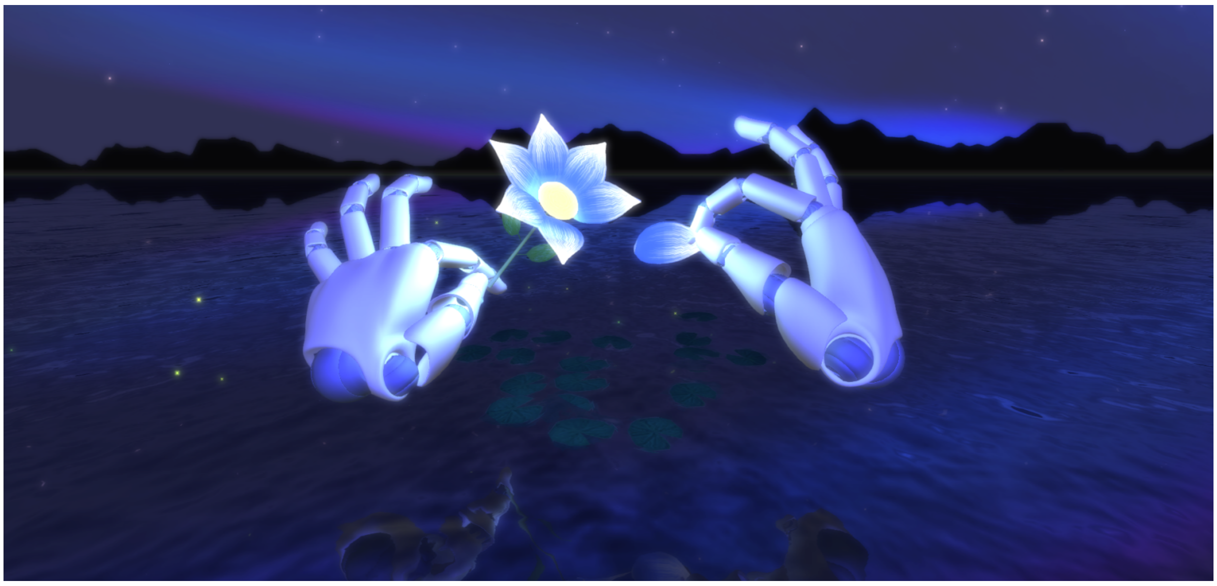

Figure 2.1: Look of the 3D hand models used in the Leap Motion software [1]

tracked correctly than. This problem could probably be solved by fusing tha data of the Kinect with the leaps data and both sensors filming from different angles.

2.5 Fusion/Switching between Kinect and Leap

Our first attempt was to switch between the both sensors so we can use the more accurate Leap when a persons hands are close enough to be detected by the Leap sensors and otherwise use the Kinect data.

This is the easiest way to have the advantages of both technologies fused in one project, but of course it probably is not the best one.

A problem with this approach is to fit both coordinate systems together, so the recorded data can be adjusted properly in the right positions relative to the coffee machine. Therefore you would also need a static setup which we didn’t have due to limited space.

Another possibility would be to really fuse both sensor data to increase the accuracy in all three dimensions, especially to prevent the hand models from being unstable while the hands are rotated. Ideally it would also be possible to import the already implemented, completely movable 3D hand model coming with the Leap Motion software, so you can really track and see the movements of the single finger joints.

CHAPTER 3

Results

In the Practical Course Data Fusion we successfully established a development environment for the use of a kinect v2 sensor and programmed a demo application, where we are able to control a coffee machine with our tracked hands in a virtual environment. For the setup of the environment we had to install the kinect v2 SDK, Visual Studio 2015 Express and Unity3d, which is decribed in 2.1. The general procedure to make the coffee machine operable is described in 2.2. We were able to show that it is possible to create a small virtual reality application with the kinect v2 in a short amount of time using freely available libraries and tools.

- screenshot kinect usage of CM
- screenshot leap usage of CM
- precision of the used technologies

CHAPTER 4

Discussion

Apart from the problems mentioned above we were astonished how quickly we were able to use our setup, move hand models via the Kinect in Unity3D and implement collision detection. Even though some things were a bit tricky, the software works quite well and can - for easy purposes - be almost used as plug-and-play. Unfortunately this can't be said for the software available for Linux PCs, Windows is in the lead here.

Of course there are several possibilities to further improve our project, some of them already mentioned above as well, e.g. the different frame selection or the fused usage of the Leap and Kinect. Our setting also might have been not ideal, e.g. the placement of the Kinect next to and not on top of the screen and the relatively small distances between the Kinect sensors and the hands.

It would also be a nice thing to stream the image of the screen to a smartphone and use a Google Cardboard to create the full virtual reality effect. A Google Cardboard is a cheap device where you can put your smartphone in and - by holding it in front of your eyes and with the help of the build-in sensors of the phone - experience virtual reality. This could improve the usage of objects like our coffee machine even more and make it feel and look a bit more real.

The whole virtual reality business is becoming increasingly important and is used more and more in our everyday life, so there are lots of projects and researches going on concerning this topic. Just to mention a few: there is the so called social virtual reality with its virtual reality chatrooms, of course there are computer games using VR, but it could also be used for medical treatments, e.g. exposure therapies [2].

CHAPTER 5

Conclusion

- possible to fuse data usefully
- possible to create a usable interaction this an object

Bibliography

[1] Hand Models of the Leap Motion - Playground Flower Scene. Website. `http://blog.leapmotion.com/wp-content/uploads/2014/11/playground-flower-scene.png`; september 28th 2016.

[2] Hand Models of the Leap Motion - Playground Flower Scene. Website. `http://www.techrepublic.com/article/10-ways-virtual-reality-is-revolutionizing-medicine-and-healthcare/`; september 28th 2016.