VRTyper: A Universal Keyboard for Virtual Reality Applications

Sam Xu
Stanford EE267
samx@stanford.edu

Partner: Kylee Krzanich
Stanford
krzanich@stanford.edu

Abstract

This project presents a 3D virtual reality (VR) keyboard using only a single LeapMotion camera to track the position and pose of the fingers and hand. The absence of tactile feedback, the inability to accurately distinguish the depth of the tapping finger, and inconsistent sensor data are all major challenges for a fully VR keyboard. To overcome these challenges, we follow an iterative process in our project. We first explore and evaluate the performance of three different sensors, the Intel Realsense, the Microsoft Kinect, and the LeapMotion, and decide which one was the most appropriate sensor. After we decided on the LeapMotion, we began development on the demo environment using the Unity game engine where we slowly tweak and optimize our virtual keyboard.

1. Introduction

Virtual reality (VR) has the potential to become a powerful collaboative platform used for schools, businesses, and even social gatherings. But regardless of the use case, any virtual office needs a reliable and effective alphanumerical text entry interface. Although other symbolic inputs, such as speech, has been proposed for VR, voice control is still limited by ambient noise sensitivity, privacy concerns, and instrusiveness in a shared or collaborative environment. With this in mind, a keyboard is the ideal and necessary solution. However, with most non see-through head-mounted-displays (HMD), it is diffcult to perceive objects from the real world, such as trying to locate and use a keyboard without visual feedback. To address this, we need a virtual environment in which a keyboard could be readily and accurately interfaced.

Current consumer and enterprise VR systems, such as HTC Vive, Oculus Rift, or Microsoft's Hololen's, often utilize a virtual pointer using hand-held controllers or head or gaze direction. However, these methods are limited in performance and consequently mostly used to enter short texts, such as passwords. Due to the full utilization of fingers, typing on a physical keyboard is perhaps the fastest text entry method, with average input speed between $60 - 80$ wpm. Consequently, many consider ten-finger typing as the most effective means that human enters text in daily lives.

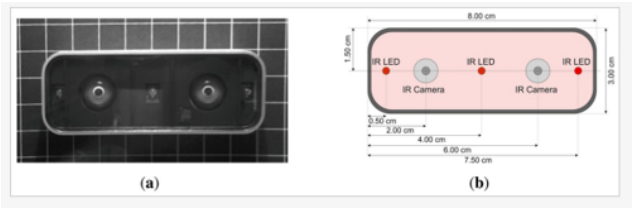

Figure 1. The physical structure of a LeapMotion Camera. Due to its small dimensions, we are able to easily mount it onto a HMD.

The task of creating a keyboad interface in a virtual environment is one of many design choices. Will our system simply render a physical keyboard that is in front of the user? or will the keyboard be entirely virtual? How will we track the fingers for our input? Will we use additional hardware like a glove-based tracker? Or will we simply use cameras for optical tracking? We seek to address these engineering questions and create a virtual keyboard that solely relies on optical tracking using a single infrared depth measuring camera

We explored three different brands of depth measuring cameras, the Intel RealSense, the Microsoft Kinect, and the LeapMotion. All three of which are built using the same core technology: stereo infrared depth sensors and infrared LEDs. However, each of these sensors are subjected to certain limitations. Ultimately, we chose the LeapMotion controller since it provides the most accurate finger tracking data, offers the lowest-cost, and is physically the least bulky.

We develope a VR typing demo in Unity using the Leap Motion camera. We use this demo to iteratively improve the user experience of our keyboard, improve our tracking strategy, and identify potential shortcomings.

2. Related Works

In the past, a variety of works have investigated the implementation of VR keyboards using wearable gloves. The

glove would be designed to tracks finger and hand mounted data that represent different keypress events. In this case, a wearable glove subsitutes the physical keyboard. [7], [9], [1], [4], [11]. A glove-based keyboard offers high data sampling rates with minimal background noise and a variety of different input techniques. Some gloves produce input when the hand touches a specific part of the hand, like the pinch Glove [1] and KITTY [7]. Both of these implementatoins require the user to memorize specific motions, like pinching the lefty pinky and thumb represents a 'a'. While this technique is very accurate and relatively fast, it is also the most inconvenient and the hardest to learn.

Others have tried to use neural networks to map physical motion into key presses [8]. A glove based tracker also enables features such as heptatic feedback from the glove when a key pressed is registered[11]. However, most of these gloves require a considerable amount of learning to use, and may also limit the user's ability to use other inputs in VR, such as controllers for games.

Other works have tried to map and render a physical keyboard into the VR environment [5], [6], [3]. These methods also require a optic motion camera setup in order to track the position of the fingers and the keyboard. However, such setup requires the user to have the physical keyboard in front of them, and the system is only tailored to that specific keyboard since the rendered keyboard must have the same dimension and layout.

Relatively few work have explored fully virtual airtyping keyboards based only on 3D handtracking data. One work was able to achieve great accuracy rate .3% [13]; However, they had many restrictions imposed on their setup. Several of which includes: the hand was cannot be occluded by itself or anything else, and the sensor was placed directly beneath the hands, the words were entered based on a predetermined vocabulary, and the key pressed was estimated using a bayesian distribution. Similary but using a completly different technology, the Canesta Keyboard [10] allows users to type on any surface by projecting the image of a keyboard on the surface. To do this, the device emits a plane of infrared light slightly above the typing surface to detect finger taps. They reported a input speed of 37.7 WPM.

Other studies have compared the effectiveness of the methods above through multiple user studies [2], [6], and found that even without haptic feedback, users prefered using purely virtual keyboards over the case when virtual keyboards are overlaid on top of physical keyboards, which adds unnessary equipment and bulk to the scenario. Similarly, for these same reason as well as the limitations we previously discussed, we chose to not go with a glove-based method to provide the most immersive experience. Additionally, Grubert et al. found that users need some form finger-tip indicators to use the keyboard correctly [2].

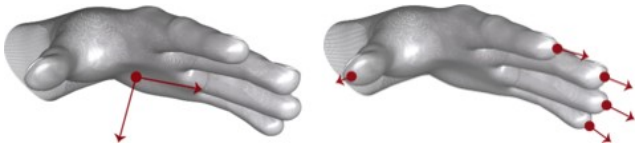

Figure 2. The left hand depicts tracking data for the normal and position of the palm. The right image depicts tracking data for the normal and position of the fingers

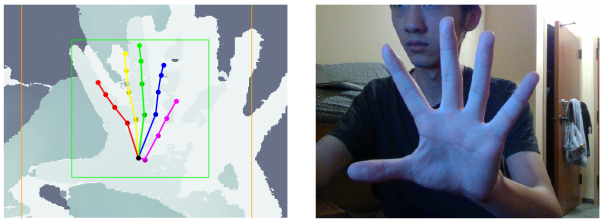

Figure 3. The right picture depicts the RGB image captured by the Intel Realsense camera. The left picture depicts the depth map and the estimated hand pose. This was implemented using the pretrained convolutional neural network provided by Wu et al. [12]

3. Method/Approach

The first sensor we tested for this project is the Intel RealSense, as shown in Figure 3. The Intel Realsense uses stereo infrared vision and a RGB camera to calculate a depth map. It offers a high resolution of 1280×720 with a field of view of $91.2° \times 65.5°$. The decent field of view and high resolution makes the Intel RealSense the most pixel dense depth camera out of the three tested. The Intel RealSense doesn't come with any finger tracking software, and its development has primiarly focused on full body skeletal tracking.

To extract the 3D handpose from the depth map, we used an pretrained convolutional neural network called HandMap[12]. The algorithm, implemented through Python, incorporates both convolutional layers and dense guidance maps into its framework. Despite this, we had limited success getting accurate finger tracking data with this technique, as shown in figure 3. Additionally, Realsense has no official support for the Unity engine, making it diffcult to utilize the tracking data in an actual demo.

We also tried the Kinect v2 for Windows. The Kinect utilizes stereo depth camera and a RGB color camera. The kinect has a smaller resolution of 512×424 with a field of view of $70.6° \times 60$. While the Kinect has a small resolution and field of view, it has a very long range, being able to spot a person from a long distance.

Unlike the Intel RealSense, the Kinect does have native support for tracking the hand, the thumb, as well as some

hand gestures. To implement full finger tracking in Kinect, the algorithm approximately searches the $10 - 15$cm radius around the position of the hand joint to estimate the contour of the hand. Next, to estimate the position of the fingers, we form a convex hull from the hand contour, where each vertex of the convex hull is a finger position. The process is shown in Figure 4 We show the results of our Kinect experimentation on Figure 5. However, similar to the Intel Realsense, the performance of this algorithm was fairly poor.

Finally, we tried the LeapMotion, which again utilizes stereo infrared camera and three infrared LEDs. The device utilizes wide angle lens that give it a wide field of view of $150^\circ \times 120^\circ$ and a short range of two feet. However, in actual use, we found that the tracking accuracy near the border of the LeapMotion to be very poor. The Leap Motion API stores all of its tracking data internally as frames at a sampling rate of 30 frames per second.

Finally, we chose to use the LeapMotion for our final demo for the following reasons: its widen lens design is designed solely for the tracking of the hand and fingers, whereas the RealSense and Kinect is more tailored for fully body skeletal trackng. The LeapMotion also provides the most accurate finger tracking data. It converts raw data from its sensors into model space coordinates using its propietary algorithm, whereas the other sensors had performance issues when even using advanced techniques [12]. Addtionally, LeapMotion is the only sensor that directly supports Unity with its SDK, allow us to easily port the sensor into a usable demo.

3.1. Unity Game Engine

For our demo, we created a virtual environment in which a keyboard is statically placed in from of the camera where the virtual hand is placed. The Leap Motion interfaces with Unity via several Unity CoreAssets library provided by Leap Motion developers, such as their Interaction Engine. This allows us to easily track both the right and left hand through simple function calls. The Unity engine tracks the sensor data per frame, using which, the rigidbody and collider physics class determines the outcome of the scene.

3.2. Implementation

To design the demo for VR, we oriented camera and the sensors to be calibrated with the head mounted position, where the LeapMotion camera is placed near the forehead. We create a Leap Rig with a hand controller, where the hand controller maintains the palm's position, the palm's normal vector, the positon for each finger, and the normal for each finger, as shown in Figure 2. The hand model provides information about the identity, position, and other characteristics of a detected hand, the arm to which the hand is attached, and lists of the fingers associated with the hand. The hand model uses a low poly mesh prefab to depict the model of the hand when in view of the camera. If a part of a finger is not visible due to occulusion, the finger characteristics are estimated based on recent observations and the anatomical model of the hand. A Finger object provides a Bone object describing the position and orientation of each anatomical finger bone.

We define our keyboard as a collection of Interaction Buttons, which registers as being pressed when it is pushed from its normal surface. Each key is combination of text mesh, texture mesh, rigged box, and collider. The key is given a script so that it glows when a finger is above its normal, and upon being pushed, the key will modify the text corresponding to an event script we wrote, and notify the user that the key press is registered with a sound.

Person	% of correct key strokes	% of false positives	Time to complete
A	.70	0.2	3:36
B	.80	0.15	3:05
C	.70	0.35	4:01
D	.65	0.35	3:59

Table 1. Our recorded results from a four person user study.

4. Results

We implemented our fully virtual keyboard in a environment as shown in Figure 7 . By correctly manuevering the hand into a button press, we were able to create a typing platform using nothing but a LeapMotion camera.

We noticed a few things. First, the LeapMotion camera starts to break down when too many fingers are present, which means that the setup doesn't currently work with multiple fingers typing at the same time. We found that the sensors are the most accurate when we only have a single finger pointing on the screen at a time. Secondly, we noticed that users have diffculty realizing when their hands were behind the keyboard, and unknowingly they weren't able to make collision with any of the keys.

4.1. User Study

For the user study, we conducted a experiment with four people and recorded their approximate correct percentage of keystrokes, percentage of false positives, and length of time. We asked them to type 20 characters that were randomly generated beforehand, and recorded the time it took to complete the task. Note that none of the users were given training beforehand the experiment. The results are shown in Table 1.

4.2. Discussion

In our experiments, the biggest cause of mistypes or other errors is sporadic sensor failure from poor occulusion perdiction, the hand accidentaly moved behind the key-

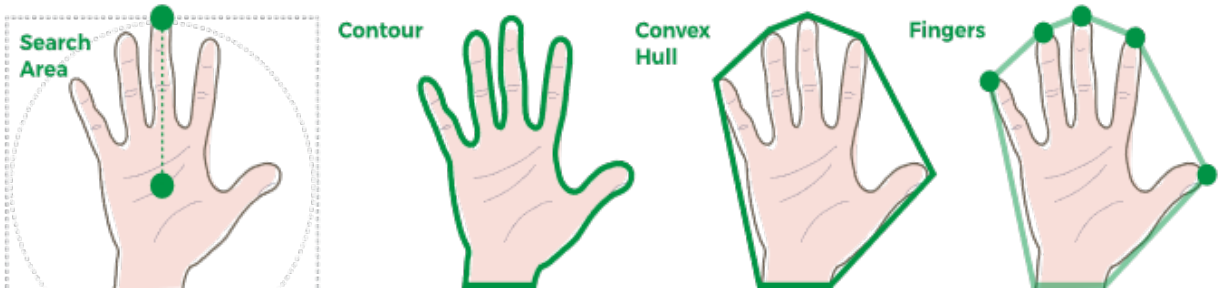

Figure 4. The figure depicts how contour of the hand is extracted, the convex hull of the hand, and why we label the vertices of the convex hull as finger positions.

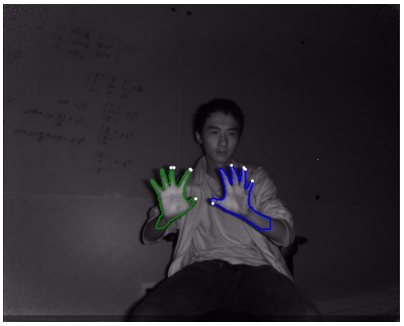

Figure 5. The figure depicts an real life example where the contour of the hand extracted using the Kinect, and the estimated finger position using its convex hull.

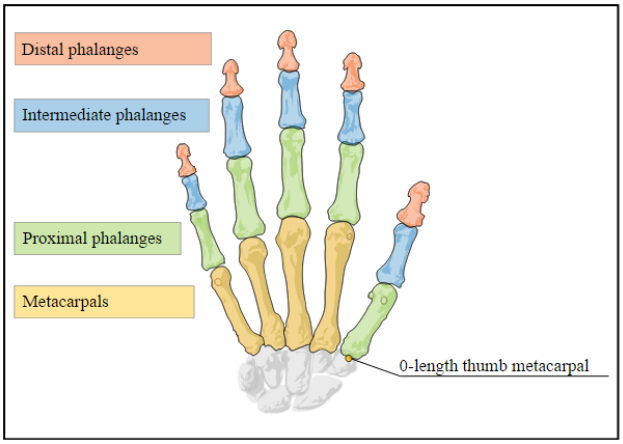

Figure 6. A finger object from the hand controller provides a bone object that describes the position and orientation of each anatomical finger bone. All fingers contain four bones and are ordered from base to tip

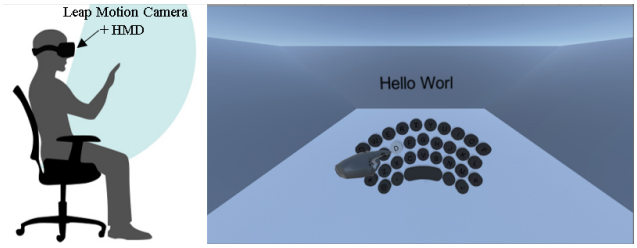

Figure 7. The left side depicts the Unity scene that contains the VR keyboard. The right side depicts the intended setup where the LeapMotion is attached to the HMD and the field of view/interaction.

board, and the button not registering a push from the side of a key. To alleviate this, we calibrated the position and orientation of the camera and keyboard such that its less likely to accidentaly press the keys on the side from a 3D perspective.

Additionally, several users reported that the keys near the border were much more diffcult to press. This is caused by the fact that the camera's tracking ability deteroiates as one's hand move to the border of the LeapMotion's field of view. Additionally, pushing the keys against its sides was more common because the keys aren't angled towards the camera and the hands; consequently, it was visually and

physically more diffcult to push against the normal surface of the keys.

Another issue we faced was incorrect sensor prediction due to occlusion. Normally, the sensor is very accurate if the palm is facing the LeapMotion camera. When orientied onto a HMD, the palm is now facing against the camera. Consequently, the fingers would sometimes be occluded from the camera by the back of the hand, after which the camera will start giving sporadic data about the hand position. However, due to the limitations of the infrared sensors, whose performance is highly dependent on the environment, it is impossible to perfectly track the finger through a software solution alone. Without a better sensor, it is likely that we will have to consider sensor-fusion where we use multiple sensors in addition to the LeapMotion. This will give us more data points where we can more accurately approximate the correct position and orientation.

However, by adding more sensors into the setup, we will also increase the complexity and bulk of the keyboard, reducing its convinence. For future work, it is likely that we will have to consider a trade-off between how much hardware we are willing to add and how much accuracy we are willing to sacrifice for less bulk.

5. Conclusion

In the end, we were able to create a virtual environment with a functional air-typing keyboard using a single LeapMotion camera. After exploring several sensors, developing an demo using the Unity game engine, and iterateively imporving our keyboard through user studies, we were able to conclude achieve a usable keyboard, albeit somewhat inefficient. The LeapMotion sensor uses nothing but a stereo infrared depth sensor, and several infrared leds. The LeapMotion is small and portable, sizing at only $3 \times 1.2 \times 0.5$ inches, making it easily attachable to any existing head mounted dispaly.

Our virtual environment contained several environmental cues that tries to alert the user to any updates. We highlight the key that the finger is currently above, and provide sound to indicate when a key is pressed. Despite this, the users from our studies still reveled many shortcomings that still needs to be addressed.

Finally, the post-experiment interviews of our user studies revealed some important information about usability and user preference. Many of them expressed the importance of additional environmental cues to subsitute the loss of haptic feedback. Additionally, some though that rendering the whole hand was unnucessary and only cluttered the view. Overall, every single participant expressed enthusiasm for the keyboard and the future of VR typing regardless of their performance in the user study.

6. Future Work

Through the user studies, we found several limitations to our virtual keyboard that could be improved. One of which is the collision engine. Because we setup our keys as interactive buttons, it doesn't register side-way collisions as key presses. This made clicking very diffcult, especially when trying to pressthe keys on the side of the keyboard or when the sensor misbehaves. To alleviate this, we would like to remake our own collision script and replace the default behavior with a more natural one.

In addition to the environmental cues we currently have, we would like to make it more visually clear to the user when their hands are clipping against the virtual keyboard. Some sort of visual highlight would alert the users when their hands are underneath the keyboard.

Finally, to solve inaccurate readings caused by the occulusion of the hand, we wondered if we can utilize a dual LeapMotion setup, where at least one camera have a unoccluded view of the entire hand (ie one placed on the HMD, and one placed on the desk, underneath the hand). Doing so could greatly improve the keyboard's reliability. it would simply become a algorithm problem sensor fusion.

References

[1] D. A. Bowman, C. J. Rhoton, and M. S. Pinho. Text input techniques for immersive virtual environments: An empirical comparison. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 46(26):2154–2158, 2002.

[2] J. Grubert, L. Witzani, E. Ofek, M. Pahud, M. Kranz, and P. O. Kristensson. Effects of hand representations for typing in virtual reality. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 151–158, March 2018.

[3] J. Grubert, L. Witzani, E. Ofek, M. Pahud, M. Kranz, and P. O. Kristensson. Text entry in immersive head-mounted display-based virtual reality using standard keyboards. In *2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pages 159–166, March 2018.

[4] Y.-T. Hsieh, A. Jylhä, V. Orso, L. Gamberini, and G. Jacucci. Designing a willing-to-use-in-public hand gestural interaction technique for smart glasses. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 4203–4215, New York, NY, USA, 2016. ACM.

[5] P. Knierim, V. Schwind, A. M. Feit, F. Nieuwenhuizen, and N. Henze. Physical keyboards in virtual reality: Analysis of typing performance and effects of avatar hands. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 345:1–345:9, New York, NY, USA, 2018. ACM.

[6] J.-W. Lin, P.-H. Han, J.-Y. Lee, Y.-S. Chen, T.-W. Chang, K.-W. Chen, and Y.-P. Hung. Visualizing the keyboard in virtual reality for enhancing immersive experience. In *ACM SIGGRAPH 2017 Posters*, SIGGRAPH '17, pages 35:1–35:2, New York, NY, USA, 2017. ACM.

[7] C. Mehring, F. Kuester, K. D. Singh, and M. Chen. Kitty: keyboard independent touch typing in vr. In *IEEE Virtual Reality 2004*, pages 243–244, March 2004.

[8] K. Min. Text input tool for immersive vr based on 3 x 3 screen cells. In G. Lee, D. Howard, and D. lzak, editors, *Convergence and Hybrid Information Technology*, pages 778–786, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[9] A. J. F. Natoli. Virtual reality keyboard system and method.

[10] H. Roeber, J. Bacus, and C. Tomasi. Typing in thin air: The canesta projection keyboard - a new method of interaction with electronic devices. In *CHI '03 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '03, pages 712–713, New York, NY, USA, 2003. ACM.

[11] C.-M. Wu, C.-W. Hsu, T.-K. Lee, and S. Smith. A virtual reality keyboard with realistic haptic feedback in a fully immersive virtual environment. *Virtual Reality*, 21:19–29, 2016.

[12] X. Wu, D. Finnegan, E. O'Neill, and Y.-L. Yang. Handmap: Robust hand pose estimation via intermediate dense guidance map supervision. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[13] X. Yi, C. Yu, M. Zhang, S. Gao, K. Sun, and Y. Shi. Atk: Enabling ten-finger freehand typing in air based on 3d hand tracking data. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 539–548, New York, NY, USA, 2015. ACM.