Camera-Based Selection with Low-Cost Mobile VR Head-Mounted Displays

by

Siqi Luo

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Computer Science

in

Human-Computer Interaction

Carleton University
Ottawa, Ontario

Abstract

I present a study comparing selection techniques for low-cost mobile VR devices, such as Google Cardboard. My objective was to assess if alternatives to common head-ray selection methods were feasible with current computer vision tracking approaches on the mobile. In the first experiment, I compared three selection techniques, air touch, head ray, and finger ray. Overall, hand-based selection technique (air touch) performed much worse than ray-based selection techniques. In the second experiment, I compared different combinations of selection techniques and selection activation methods. Results indicated that the built-in Cardboard button worked well with head ray and hand gesture with ray-based techniques can be an interaction potential on mobile VR. I concluded that camera-based ray selection techniques and hand-based activation mechanism are promising on Mobile VR in the future.

Acknowledgements

First of all, my deepest appreciation goes to my supervisor, Dr. Robert Teather. This research would not possible be accomplished without his guidance. His patience and wisdom made me keep the right way along my whole research period. Since two years ago, my first time came to Carleton University as an international student, Dr. Robert always give me full understanding and tolerance and keep me improving constantly.

I would like to thank my friends, Yasin and Heather, who always mentally encourage me. Especially during the period, my cousin past away, they helped me get out of depression and overcome homesickness.

I would like to thank my boyfriend, Zeqi. He is always understanding and unconditionally support me.

Finally, I would like to thank my parents, without their understanding and efforts, I would never have chance to study here and meet such great people.

Thanks to all these people, I could make my thesis done. Thank you!

Table of Contents

List of Illustrations

List of Appendices

1 Chapter: Introduction

1.1 Technology of Virtual Reality

Immersive virtual reality, more commonly referred to simply as Virtual Reality (VR), offers a way for people to interact with simulated 3D environments as naturally as the real world [32,3]. The objective is to take advantage of the user's experience with the real world and transferring understanding from reality to the simulated environment [2], or vice versa in training scenarios such as through military training and phobia therapy. In a good VR system, users report having strong feelings of really "being there", a phenomenon known as presence [20]. Presence is thought to be caused, at least in part, by systems offering a high level of immersion [7].

According to Slater [20], the definition of immersion is "the objective level of sensory fidelity a VR system provides". Since ideal VR systems include all sensory modalities, better visual display technology (e.g., using stereo rendering, or head-tracked viewpoint) enhances immersion. However, it also relates to the quality of sensors used both for interaction, and to support head-tracking. If the sensor data is synchronized well (i.e., little latency and noise due to technical problems) then the immersion will be greater and thus so too will be presence.

Various tracking systems have been developed to work with different types of VR devices. The most common examples include full-body tracking and head tracking. Full-body tracking requires a user to wear markers spread across the body, which are typically tracked by cameras for real-time computation [23]. This type of tracking necessitates complex data processing to determine the position of each marker worn by the user. Full-body tracking can offer absolute 6 Degree of Freedom (DOF) tracking. Head tracking,

which often uses 6DOF trackers, is a key element of immersion. It is achieved by tracking the user's head orientation and position to match the view presented in a head-mounted display (HMD). Head pose information can be acquired using various high-end sensors (e.g., optical, magnetic), but today, inertial sensors such as accelerometers and gyroscopes built into HMDs can be sufficient.

Although several VR displays have been previously developed, the most common commercial solution today is the head-mounted display (HMD). Commercial HMDs often include tracked hand-hold controllers as input devices. The two most popular devices are the HTC Vive[1] and the Oculus Rift[2]. Both include an HMD, two controllers that allow the user to interact with the virtual environment and extra sensors to track the HMD and controllers. They are priced at $699 and $529 respectively. Due to factors such as high price and physical space requirements, these devices have reached only a limited market [27]. In addition, these devices require a high-end PC to drive them. To overcome this shortcoming, future self-contained versions of the Oculus Rift[3] will no longer require a PC anymore. This suggests that VR is trending towards portability.

A popular alternative to dedicated VR devices is mobile VR devices. Mobile VR devices do not need an external computing device. Instead, a mobile phone (which is assumed to already be in the user's possession) acts as both the computing device and display. As a result, mobile VR devices are priced much lower than dedicated devices, and

[1] https://www.vive.com/ca/product/

[2] https://www.oculus.com/

[3] https://www.oculus.com/?id=1227450960663477

also boast lower hardware requirements. Devices such as Google Daydream[4] (which includes a plastic HMD shell for a mobile phone as well as a touchpad controller) are priced at around $140 and makes VR experiences possible with ordinary mobile phones. Google Cardboard[5] (Figure 1) and similar devices which use a cardboard HMD (but no input device) are even cheaper and more portable.

Figure 1 Google Cardboard provides a cardboard box to hold a mobile device, and optics to help the user view the screen. Typically such devices do not include an external controller, but offer a button to support interaction (seen at the top right side of the device).

Google Cardboard is, simply put, made of pieces of cardboard and two focal length lenses. When using it, the user must put a smartphone into the cardboard box. The smartphone's built-in accelerometer and gyroscope are used to track head rotation. The button located on the side of Google Cardboard can be pressed down and can provide input (e.g., activating a selection). These are among the most accessible VR devices today. Lower costs and usage simplicity attract a larger population of customers. According to

[4] https://vr.google.com/daydream/

[5] https://vr.google.com/cardboard/

Statista[6], since 2014, Google has shipped 10 million Google Cardboard headsets and it is estimated to have sold between 2 million and 3.5 million Daydream devices in 2017 alone[7]. Samsung sold around 6.7 million Gear VR headsets in 2017[8]. Besides, the fact that big companies (e.g., Facebook) have begun to invest in mobile VR devices suggests that mobile VR has a promising future.

1.2 Motivation

Combining cheap and lightweight cardboard-style HMDs with mobile devices makes VR more accessible to people than ever before. Such devices have the potential to reach a wider user base than high-end products like the Oculus Rift [27]. However, there is a major limitation with these devices. Due to the low fidelity of the built-in mobile sensors as well as the inability to do absolute 6 DOF position and orientation tracking, interaction with virtual environments presented on mobile VR is much more limited than with trackers offered by high-end HMDs. The motivation of this thesis research is to thus explore the interaction potential of using low-cost mobile head-mounted displays such as Google Cardboard.

Numerous interaction techniques have been proposed for interaction (selection and manipulation of objects, navigation, etc.) in VR. However, without absolute position tracking, only a few of these are compatible with mobile VR [27]. Browsing the Google Play store, one can see how the variety of applications is considerably lower than high-end

[6] https://www.statista.com/

[7] https://www.engadget.com/2017/12/20/vr-and-ar-in-2018/

[8] https://www.gamesindustry.biz/articles/2017-05-09-gear-vr-far-and-away-the-leader-with-almost-7m-expected-to-be-sold-in-2017

HMDs due largely to this interaction limitation. The majority of applications are "look-and-see" type applications, which involve a fairly passive user experience of watching videos in stereo 3D, sometimes using head tracking (with head orientation provided by the mobile device's inertial measurement unit) to allow the user to look around the scene [38]. Some applications support rudimentary interaction by way of head tracking. For example, head-ray based interaction can be used to activate selection by looking at the target object. The pointing ray originates from the head and follows the tracked head post by head tracking system. Selection is generally activated by pressing the capacitive button on the Cardboard, or through a timeout. Such interaction styles are much less expressive than the dual-tracked wand input devices provided with high-end HMDs.

There is also little research on whether using the type of button provided on cardboard devices is the best design alternative for selecting targets in mobile VR applications. Previous research by Yoo et al. [35] looked at mobile VR applications on the Google Play store. The researchers tested 32 applications designed for Google Cardboard, chosen based on their popularity and feedback within the store. They categorized the applications into five types: 1) those using the head ray to find the target object and pressing the side button to confirm a selection, 2) those using the head ray to find the target object and automatically confirm selection instantly, 3) those using the head ray to find the target object and confirming the selection by keeping the ray over the same object for a specific timeout delay, 4) using tilt supported by built-in gyroscope to control object's orientation and 5) those requiring the user to connect an extra controller to the phone. Most applications fell under category 1 (using the Google Cardboard button) despite results showing it offered worse than average user feedback among the five categories. Surprisingly, applications requiring extra controllers got worse feedback, perhaps due to the necessity of using a

separate external device (limiting portability of the VR experience). This result indicates that using external controllers is less attractive than controller-less interaction for users.

Above all, my motivation was to find a better interaction method for "Cardboard-like" HMDs. Inspired by modern computer vision, I considered the built-in cameras on cell phones might be another usable sensor. By tracking the users' hands, gesture-based interaction could be used as an alternative to head-based selection using a button. Several popular products (e.g., Leap motion[9]) have achieved gesture-based input utilizing cameras and other sensors. However, these types of products either are PC-based or require high-resolution and/or depth cameras, which are typically not available on mobile.

In my case, I instead propose to use the standard RGB cameras in cellphones for tracking the hand. Although it is anticipated that tracking accuracy is likely lower, I believe it is still worth investigating the performance potential of the single rear-camera on tracking hand gestures. After all, such cameras are already on virtually all mobile devices, conveniently face outward from the Cardboard when mounted in the HMD and necessitate no additional hardware. Specifically, my present work is designed to answer the following research questions:

1. How effective is hand-based technique tracked by a smartphone camera, compared to head-based input provided by the internal IMU?

2. How effective is gesture-based selection activation, tracked by the smartphone camera, compared with using a button?

My work thus compared different interaction methods for target selection, specifically

[9] https://www.leapmotion.com/

focused on those used with mobile VR presented on a Google Cardboard.

1.1 Outline

This thesis is organized in 5 chapters. Chapter 1 is the brief introduction of VR technology and the motivation of my work. Chapter 2 reviews relevant literature related to my research problems. Chapter 3 details of my first experiment, which compared selection using head-based input with two hand-based methods using the camera for tracking. Chapter 4 details the second experiment, which compared different methods of activating selection (the Cardboard button and hand gestures). Chapter 5 presents a final summary of results, limitations and future work.

2 Chapter: Related work

To determine potential interaction methods, I could use with mobile camera-based tracking, I reviewed 1) Fundamentals and theory about existing interaction techniques 2) existing techniques and their limitations.

2.1 Interaction in 2D vs. 3D

Interaction in 3D scenarios is more complex than 2D scenarios [14] depending on the task presented in the virtual environment (VE). Interaction in 2D scenarios only requires up to 3 degrees of freedom (DOF) including translation in the *x* and *y*-axes and sometimes rotation around the *z*-axis. Full 3D interaction requires more DOFs: translation on the *z*-axis, and two more rotational DOF about the *y*- and *x*-axes. Because of this difference, 2D interaction styles are normally not well suited for 3D scenarios [1]. Additional DOFs can give users more freedom to operate in VEs but can also be a source of frustration [22]. Previous work suggests minimizing the required DOFs for manipulating virtual objects in the VE. The more DOFs simultaneously supported, the greater the difficulty to control the interaction technique [6]. Besides, lack of tactile feedback and higher latency and noise from motion tracking system can lower user performance as well [11,23].

In addition, interaction in 3D scenarios is also more physically demanding. Consider, for example, grasping an object in the air in a 3D environment, as compared to a conceptually equivalent task (dragging an icon) using a mouse on a desktop. Unlike the 2D scenario, which requires small and quick muscle movements, grasping in 3D requires compound muscle movements that tend to be larger and slower, and can yield substantial arm fatigue [38]. However, the familiar 2D "desktop metaphor" is unsuitable in VR since the physical mouse and keyboard cannot be seen while wearing head-mounted display devices [11]. Current solutions are 2D-inspired interaction styles, often modelled after

mouse control while using 3D input devices (e.g., using a wand to point a ray at an essentially 2D UI).

Above all, interaction in 3D has many difficulties. In my case, however, interaction with low-cost mobile VR HMDs exacerbates these problems with simplistic interaction styles and poor motion tracking hardware (IMUs). This thesis explores potential interaction methods under these constraints.

2.1.1 Selection in 3D

Selection is one of the fundamental interaction tasks across all user interfaces [5] and involves specifying a target using a controller, often for subsequent operations. For example, object manipulation (moving/rotating an object) in VR is usually preceded by selection. Selection is also frequently used as a means to control the system (e.g., selecting options from menus). Although selection tasks generally are short (1-2 seconds), due to their relative frequency, better selection techniques can improve system performance overall [1]. Based on this observation, I focus on comparing different 3D selection techniques in mobile VR scenarios.

Many factors impact selection results, including the target's size and location [2], properties of the input and display devices, target density, etc. For example, the degrees of freedom (DOFs) supported by the input device influence selection, with lower DOFs generally providing better performance [5]. Display size and resolution can also affect on selection performance [23]. Research by Teather et al.[30] shows that even simplistic visual feedback mechanisms, such as target highlighting, can increase selection time while decreasing the selection error rate.

2.2 Interaction techniques

Many previous interaction methods have been proposed for interaction in VEs.

According to Poupyrev et al. [26], these interaction techniques can be classified into two types: exocentric and egocentric techniques. Exocentric techniques or third person techniques, are those where the user works from outside of the environment and sees an avatar in the VE. For example, the Oculus flagship game, Lucky's Tale[10], requires players to control a fox avatar who represents the player. Exocentric interaction is comparatively rarer than egocentric interaction in VR. Egocentric interaction is often referred to as first-person interaction. The user stands inside of the VE. Egocentric metaphors are widely used in immersive VR to give user a stronger sense of presence or, the feeling of "being there". Most egocentric interaction techniques fall roughly into one of two broad categories, virtual pointer metaphors (remote pointing with a ray) and virtual hand metaphors (direct touch for selection). I compare representative selection techniques from each category in this thesis, and so review common examples here.

2.2.1 Virtual Pointer metaphor

The virtual pointer metaphor, or ray-based interaction, entails selecting a target by emitting an infinite ray from a tracked body part, usually the head or hands (or hand-proxy such as a wand). Ray-based interaction is one of the most widely used interaction techniques for selection tasks [6]. Ray pointing generally requires a tracked controller which emits a selection ray. By moving or rotating the controlled object, the ray changes direction to point to target objects. The tracked objects can be instruments such as a pen, laser mouse, or even limbs. Rays can be easily generated, for instance, using the head tracking system; the head ray originates at the head position, and the follows the head's rotation. This style of interaction is widely used with devices such as Microsoft's

[10] https://www.oculus.com/experiences/rift/909129545868758/

HoloLens[11] and Google's Cardboard. High-end VR systems (e.g., HTC Vive and Oculus Rift) use hand-held controllers to emit a ray instead; the ray is controlled by pointing the controller at targets.

It is still questionable that if the head ray is the best pointing technique for Cardboard-like HMDs. Previous research shows that hand-directed pointing has higher performance than eye-directed pointing [18]. Un-instrumented in-air pointing with tracked hands was studied in desktop settings for use in situations where a mouse was impractical, such as or cooking without an extra hand touch the screens [3]. However, it is difficult to define a selection ray by tracking hand motion with a single RGB camera, since such input is notoriously noisy. Inspired by Zeleznik et al.'s research on image-plane interaction [37], I implemented another pointing technique in my work – the finger ray – as a comparison to the head ray.

The advantage of using pointing rays to select targets is evident when the target is located out of arms reach [26]. Also, as mentioned above, rays require comparatively fewer DOFs to control: as few as 2DOF – rotation about the *x*- and *y*-axes to control the ray direction. This can further improve the selection performance since it is easier for users to manipulate [7]. In my case, I use the head ray as a representative interaction technique commonly used with Google Cardboard.

However, ray-based techniques also have some disadvantages. For example, the user must keep the pointing device steady until confirmation [1]. Movement of the ray's origin during the final steps causes the pointing ray to move out of the target, resulting in missing the target. This is compounded by the so-called "Heisenberg" effect [8], where pressing

[11] https://www.microsoft.com/en-us/hololens

the input device button to indicate selection causes the device to move at the moment of selection. Another problem is that standard ray pointing is unsuitable for situations with occluded targets [10]. Rays generally support the selection of only the closest target to the user; another object in the ray's direction makes it hard to distinguish which one should be selected.

2.2.2 Virtual hand metaphor

Using the hand is the most natural interaction style for humans and this makes it one of the ideal interaction styles in VR [17]. The traditional virtual hand metaphor is a one-to-one mapping between the tracked hand, and a hand representation displayed in the VE, and supports full 6DOF interaction. However, direct selection using hands in VR is quite different from the real world [1]. For example, the user might be unable to see their body in the VE but may see only a simple hand avatar [19]. Moreover, the lack of tactile feedback yields lower performance [9]. Compared with ray-based techniques, the main limitation of using virtual hands is the reachable distance. For remote objects, users must first navigate to the target to accomplish selection tasks. Despite this limitation, previous work has shown that virtual hand techniques can offer higher performance than ray-based interaction within arm's reach distance [22]. Since virtual hands require full 6DOF control, users have more freedom, but this can also cause problems. For example, users must move their hands in multiple directions, and the lack of proper depth perception affects performance.

Besides, absolute 6DOF position tracking is not yet fully realized for normal mobile devices. Glove-like input devices can offer full 3D hand tracking (including fingers) using various sensors [6]. These techniques require a higher level of sensor fidelity, which is inappropriate for common mobile devices. Vision-based tracking, in contrast, has become

more popular with the development of technologies like Leap Motion[12]. This process relies on images or videos captured by a camera, then determining the hand pose through processing and analyzing. To date, few mobile devices use the kind of depth sensors provided by devices like the Leap Motion though. Vision-based hand tracking on mobile devices is faced with a different set of challenges, notably the fidelity of the built-in camera, and comparatively limited computing power for hand detection and gesture recognition [12]. Overall, since my motivation was to better understand hand-based input performance in mobile VR, and the fact that virtual hand techniques are popular in modern VR, I intended to determine if the advantages of virtual hands held true when tracking was done with the single built-in RGB smartphone camera in mobile VR scenarios.

It is difficult to determine whether ray-based or virtual hand techniques are the better interaction method for VEs [26]. Past research has shown that ray-casting techniques result in better performance when high accuracy is not required, while virtual hands perform better in higher accuracy tasks [26]. A possible explanation is that the user can clearly see the virtual hand interacts with virtual objects so that, to some extent, overshooting or undershooting can be avoided [26]. Therefore, choosing a suitable interaction method depends on the task content. In the present study, different interaction methods for mobile VR scenario are compared. Moreover, variables affecting each interaction method's performance are explored.

2.3 Interaction on Mobile VR

Due to the low-cost components and simple configurations, mobile VR is facing several interaction problems. First, a limitation of HMDs, like Google Cardboard, is that

[12] https://www.leapmotion.com/

users cannot touch the mobile device screen or press buttons on smartphones inside the box. To a large extent, this restricts the number of interaction possibilities and has a negative influence on application variety [27]. Noting this problem, the first version of Cardboard[13] was equipped with a magnetic button on the side. Pressing the button triggers the smartphone's magnetometer, which could be used to activate discrete actions such as menu or object selection. The second version of Cardboard[14] replaced the magnetic button with a conductive button which taps the screen of the smartphone inside the box upon being pressed. These support simple interactions such as using a ray emitted from the head position and controlled by the view direction (I refer to this as head ray in this thesis), along with the button to activate the selection. Additionally, higher priced mobile VR HMDs, like Google Daydream and Gear VR, are equipped with an extra touchpad controller as another way of input. Compared to them, Google Cardboard is designed without an extra controller, which makes interaction inconvenient and restricted.

The interaction methods of existing applications for Google Cardboard can be summarized in five types [35]:

1) using the head ray to find the target object and pressing the side button to confirm a selection
2) using the head ray to find the target object and automatically confirm selection
3) using the head ray to find the target object and confirming the selection by keeping

[13] https://vr.google.com/cardboard/get-cardboard/

[14] https://vr.google.com/cardboard/get-cardboard/

the ray over the same object for a specific time

4) using tilt supported by built-in gyroscope to track head's rotation
5) using extra controllers connected to the phone.

However, various types of mobile VR games require more efficient and comprehensive interaction methods. For example, for navigation games, the player needs to observe surroundings and react at the same time; head-based interaction can be unsuitable in such scenarios. There has also been little research looking at the effectiveness of buttons in the style of that used with Google Cardboard. Yoo et al. [35] found that although most applications use Google Cardboard button as the input approach, it offers worse than average feedback among the options listed above. Surprisingly, applications requiring extra controllers received the worst feedback. One reason is that when users are doing tasks with fewer devices, they can focus better on the task rather than get distracted by the devices themselves [1]. Based on these observations, hand-based interaction without extra controllers has potential as a more expressive interaction style for mobile VR [20]. Besides, the hands are universally accessible in mobile VR and thus do not require any additional equipment [32].

2.3.1 Input devices

Input devices for mobile head-mounted displays like Google Cardboard can mainly be summarized as the following types:

***Controller-less input*:** Controller-less input does not rely on extra equipment except HMD devices and cell phones. This type of interaction usually makes use of the built-in sensor such as an accelerometer to track gazing direction. Or, using the built-in cellphone camera captures the image then provides pose information after processing and analyzing. Fistpointer is a new controller-less technology presented by Ishii et al. that offers a bare-

hand method for mobile virtual reality [16]. They published a thumbs-up hand gesture for target selection game. Using only the rear camera, a players' hand could be tracked. Their software can also detect a clicking gesture to trigger target selection. However, their study focused on hand gestures recognition algorithms and did not compare performances between different interaction methods. Baldauf et al. proposed a method to detect a users' hand by tracking the points of fingertips [2], which could be applied in both VR and augmented reality (AR). However, they did not evaluate their interaction method.

External Controller Input: In contrast, this type of input needs extra controllers or sensors to send information of tracked objects. PAWdio [26] is a technique that uses common earphones as input devices to track the distance between user hand and cellphone. Through acoustic sensing, distance data is detected by calculating the time difference between sound chunks played by the mobile speaker. This is sufficient to give a distance estimate, providing 1DOF input. This offers a good solution for interaction along the *z*-axis using only simple extra devices. However, PAWdio only works with built-in head-ray pointer for *x* and *y*-axis control, in order to provide full 3/6 DOF operation.

FaceTouch [13] offered a back-of-device gesture for mobile virtual reality by attaching three extra touchpads in the back, the left and the right side of a head-mount device respectively. Users can interact with a virtual environment directly by using their fingers. Similarly, Wigdor et al. [33] created a back-of-device interaction method called LucidTouch for small screen mobile devices. The main difference is that LucidTouch allows fingers to be visible with a semi-transparent figure so as to avoid colliding with menu or objects on the screen while FaceTouch does not offer this feature.

As mentioned in the previous section, controller-less input is more welcomed in the mobile application market nowadays [9]. There are no researches on controller-less input

for mobile VR HMDs with quantitative evaluation in the past. In my work, since I focused on selection tasks, it is worthwhile to think about compatible selection techniques in such scenarios. Besides, since users cannot directly touch the cellphone's screen in the box, it is also important to explore the suitable selection activation mechanisms coupling with selection techniques.

2.4 Fitts' law

Fitts' law [21] has been used widely in the measurement of pointing performance, and I use it in my experiments. This has been formalized in the ISO 9241-9 standard [15] for pointing device evaluation.

Fitts' law models the relationship between movement time (*MT*) and selection task difficulty as a linear regression model:

$$MT = a + b\ ID \qquad \text{Equation 1.}$$

Where *ID* is index of difficulty, the selection task difficulty based on target size (*W*) and distance to the target (*A*). ID is thus given as:

$$ID = \log_2(A/W + 1) \qquad \text{Equation 2.}$$

Throughput is a metric which combines movement time and accuracy, and is calculated as:

$$TP = \frac{\log_2\left(\frac{Ae}{We}+1\right)}{MT} \qquad \text{Equation 3.}$$

TP represents throughput, and the log term is the *effective* index of difficulty (ID_e). ID_e better captures actual participant performance in an experiment through an accuracy adjustment, treating missing selections near the target as hits on an effectively larger target so as to correct experimental error rate to 4% [15]. *We* is effective width and A_e is the effective amplitude (distance) of movements. A_e is calculated as the average of the actual

movement distances. W_e is calculated as:

$$W_e = SD_X \times 4.133 \qquad \text{Equation 4.}$$

W_e is calculated from the standard deviation (SD_x) of selection coordinate over/undershoot lengths, projected onto the task axis (the line between subsequent targets). This "flattens" the selection task into 1D, since Fitts' law was originally derived for one-dimensional movements. SD_x is then multiplied by 4.133 [8], which corresponds to a z-score of ± 2.066 on the normal distribution, the scores at which 96% of the values fall under the curve. In the case of target selection, this corresponds to 96% of selections hitting the target, or in other words, a 4% error rate. W_e thus corrects the experimental error rate to 4% via this accuracy adjustment, which enables comparison with other studies, and better captures the task performed by users, rather than the task presented.

In my experiments, I adopted a previous methodology [31] for extending a 2D Fitts' law task into 3D scenarios. For ray techniques W_e was calculated by projecting the target point onto the task axis (the vector from the last target to this target) while for air touch, is calculated by the straight-line distance from the target to the selection coordinate instead. For ray techniques, A_e, effective movement distance, is the averaged actual movement distances from projected cursor point on previous trial to projected cursor point on the present trial. For hand-based techniques, A_e is the Euclidean distance between the previous cursor point to present cursor point [31]. These are calculated differently between the two interaction techniques, since this better capture the effect of target depth on pointing performance.

To explore the relationship between accuracy, speed, and throughput, Mackenzie and Isokoski conducted a study in which data were collected separately in three different-oriented tasks, speed-oriented, accuracy-oriented and speed-accuracy balance. The result

shows that throughput is constant regardless of participant tendency towards accuracy or speed [21]. In my studies, participants were told to finish the task as well as possible with both accuracy and consumed time where they can successfully select an object would be considered as performance.

2.5 Summary

In desktop computing and even VR selection tasks, the mouse is generally accepted as the most efficient input device so far [14]. However, I argue that a key requirement of mobile VR interaction is that it does not require a secondary input device, in order to maintain portability. Hence, external control devices such as the mouse are unattractive for use in mobile VR contexts. I instead opt for camera-based approaches.

My observations prior to my experiments are that the tracking reliability of the phone's camera is considerably lower than a mouse. A survey on present vision-based hand tracking technologies also suggests that generally speaking, hand posture estimation relying on a single camera is quite challenging [11]. However, there are no previous research on camera-based interaction in mobile VR, it is still unclear how much better or worse camera-based interaction techniques are than other existing techniques, like head-ray based selection. Also, it is worth exploring what kind of current camera-based interactions offer the best utilization of built-in cellphone cameras. For example, using interaction techniques which rely on depth precision might result in reduced performance, or camera-based input may be best suited to discrete actions, e.g., as alternatives to buttons. Based on these unknowns and potential, the present study is intended to give some guidance on the possibilities of interaction design in mobile VR.

In terms of the experiment settings, since performance is influenced by factors such as background colour and lighting, which affect the contrast between the hand being

tracked and the background, my experiments controlled these factors for optimal conditions; the background colour was black, and the lighting was bright. While simpler and more appropriate than a real-world situation, my camera-based techniques were too unreliable under "general" usage conditions. I note that this is a limitation of my experiment, but that my results should be considered best-case with current technologies.

In summary, and based on the past research reviewed above, I have found that there are no previous Fitts' law evaluations of camera-based selection methods for mobile VR. Such experiments would give a good sense of the "ranking" of camera-based input on mobile; a principle advantage of the ISO 9241-9 methodology is that throughput facilitates cross-experiment comparison of results. In my case, I decided to compare different interaction methods for target selection, specifically focused on those commonly used with mobile VR presented on a Google Cardboard, as described above. Since camera-based tracking can also support gestural selection activation, I also conducted an experiment comparing button-based selection activation to gestures. My present work is thus designed to evaluate:

1. How effective are two alternatives, finger ray and air touch for hand-based interaction tracked by a smartphone RGB camera, compared to head-based input provided by the internal IMU?

2. How effective is gesture-based selection activation, tracked by the smartphone camera, compared with using a button?

3 Chapter: Experiment 1

In this experiment, I compared three different selection techniques using a mobile VR HMD, specifically a Google Cardboard. The selection task was based on that provided by ISO 9241-9 [15].

The three selection techniques investigated included the head ray, finger ray, and air touch. The head ray is a representative interaction technique used in real mobile VR scenarios, as described earlier. Selection is performed using a ray originating from the user's head, with the ray direction controlled by the users' head orientation (via the mobile IMU). See Figure 2.

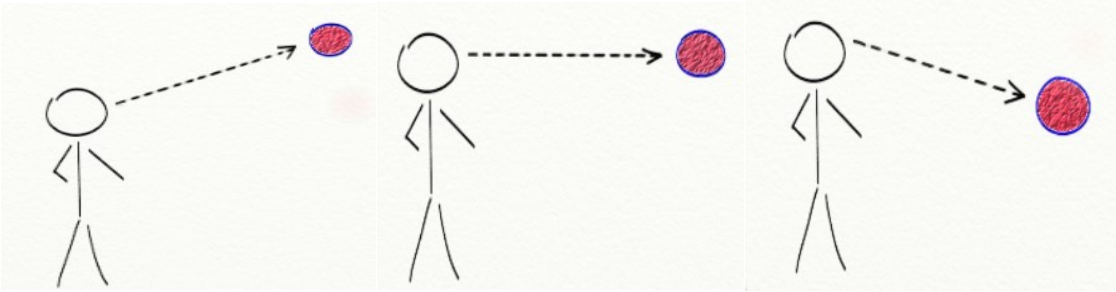


Figure 2 Participants with Head ray pointing to targets in different positions

The finger ray technique is modeled after image-plane interaction [37] and also uses a ray originating at the head. However, the ray direction is instead controlled by tracking the users' index fingertip with the mobile device camera. See Figure 3.

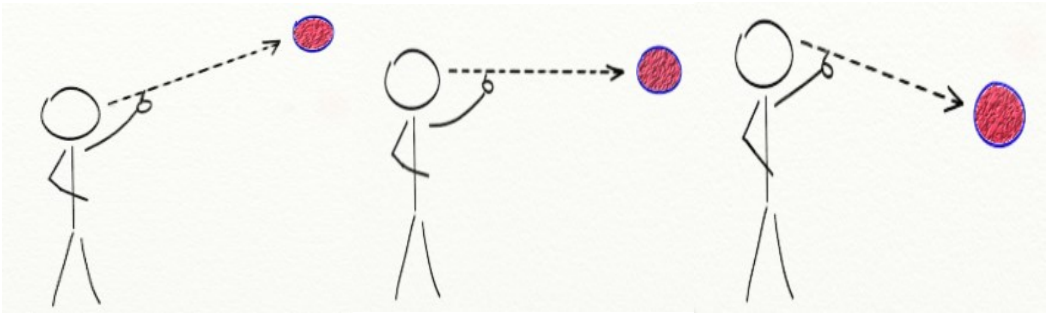


Figure 3. Participants with Finger ray pointing to targets in different positions

Finally, the air touch technique is an example virtual hand and works exactly like touching objexts in the real world; the user must physically tap the targets in space (Figure 4). This is accomplished by tracking the finger position with the mobile phone camera. Hand depth is estimated using the Manomotion SDK[15], normalized between 0 and 1, where 0 is the maximum hand area supported (80% of the camera view) and 1 is the minimum hand area supported (the furthest distance, 2% of the camera view).

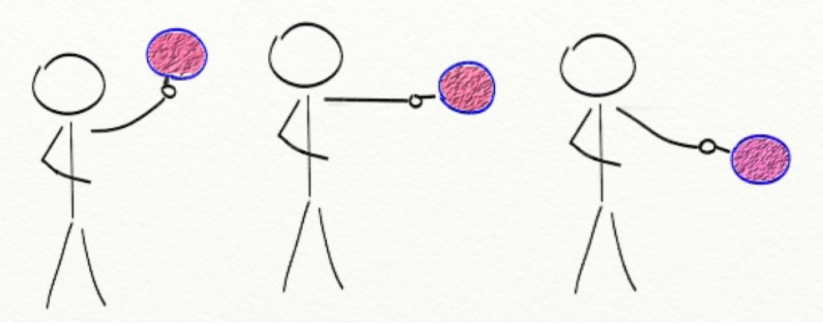


Figure 4. Participants with Air touch pointing targets in different position

3.1 Hypothesis

My hypotheses include:

H1: Finger ray will produce the highest accuracy because, as found in previous research, hand postures can help determine the selection target positions [37]. Also, working with the hand and head together is arguably more natural than aiming with the head only, as with head ray.

H2: Head ray will yield the shortest movement times, since it is only controlled by the head (and thus requires fewer DOFs to control than finger ray) and the IMU hardware provides relatively more stable position/orientation data than the computer vision library

[15] https://www.manomotion.com/tutorials/sdk-general-overview/

used with the finger-based techniques.

H3: Air touch will take longest movement time since it takes time to adjust depths on the z axis.

3.2 Participants

I recruited 12 participants (2 females and 10 males) aged between 18 and 30 years old (mean≈ 22.67 years old). Two were left-handed. All were Carleton University students with two having prior experience with Google Cardboard, and three having previously experienced VR with other hardware. The rest had no prior VR experience. None had vision or motor problems.

3.3 Apparatus

I used a Samsung Galaxy S8[16] smartphone as a display device (Figure 5, left) to display the VE. The device has a 5.8 in. screen at 1440x2960 pixels resolution and 12-million-pixel main camera. It weighs 155g, and its dimensions are 5.86 x 2.68 x 0.31 in. I used a Google Cardboard v2 (Figure 1) as the HMD. The Version 2 Cardboard has a conductive button on the right side; pressing the button taps the mobile touchscreen inside the HMD. I considered that the right-sided button on Google Cardboard would prevent the use of the right hand to perform hand postures, specifically in the finger ray and air touch conditions. However, since most of my participants (and people in general) are right-handed, using the left hand to perform hand postures while using the right hand to press the button would certainly provide unrealistic performance results. Consequently, I used an extra touchpad device, a Xiaomi cellphone[17], as external selection activation mechanism

[16] https://www.samsung.com/

[17] https://www.mi.com/en/minote2/

for all conditions. The Xiaomi touchscreen phone was connected to the Samsung Galaxy 8 through Bluetooth as a touchpad for confirming selections (Figure 5, right). The secondary mobile phone was positioned at the middle of the table in front of the participants, displaying a virtual game controller. In all conditions, selection was confirmed by tapping the "A" button on the touchscreen. All virtual environments were developed on Unity3D 5.5 and C#.

Figure 5. Samsung Galaxy 8 (left). Xiaomi touchpad (right).

I used the Manomotion SDK[18] to acquire the hand position for use with both the finger ray and air touch conditions. Manomotion uses the built-in RGB camera on the back of a smartphone to track the users' hand, providing the coordinates of the fingertips and palm center. More details about software can be find in 5.2Appendix G

3.3.1 Software Design

In my software environment, one target sphere appears in each selection trial at a specified position. The target was initially blue upon appearing and became pink when being hit. The first target appeared at the top of the ring cycle. The overall targets sequence

[18] https://www.manomotion.com/tutorials/sdk-general-overview/

is seen in Figure 6

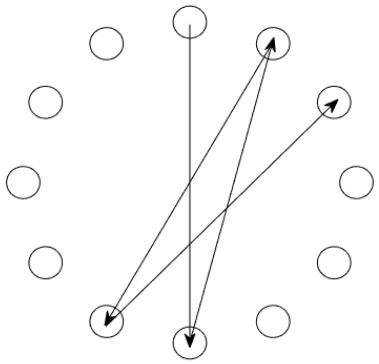


Figure 6 Appearance sequence of targets in a ring

In the head ray condition, the ray is cast from the head and follows the head's orientation tracked by IMU hardware inside the smartphone (Figure 2). A black dot in the center of the viewport provided a cursor to use for selection. When the participant turned their head, the cursor always remained in the center of the viewport. Pressing the "A" button on the secondary touchpad (Xiaomi mobile phone) "clicked" the target. Upon clicking, the target disappeared immediately, and the next target appeared, whether the target was hit or not by the selection ray. The same style of "clicking" targets was used with the other two techniques as well.

In finger ray condition, the ray is cast from the head and the direction follows the participant's the index fingertip (see Figure 3). The position of the index fingertip is tracked by the rear camera at the back of the smartphone. A black dot at a specific distance on the ray provides a cursor for selection.

In the air touch condition, the participant's position of index fingertip was tracked by camera and depths was computed by cellphone's CPU (Figure 4). A black dot followed

tracked index fingertip's position as a cursor for selection.

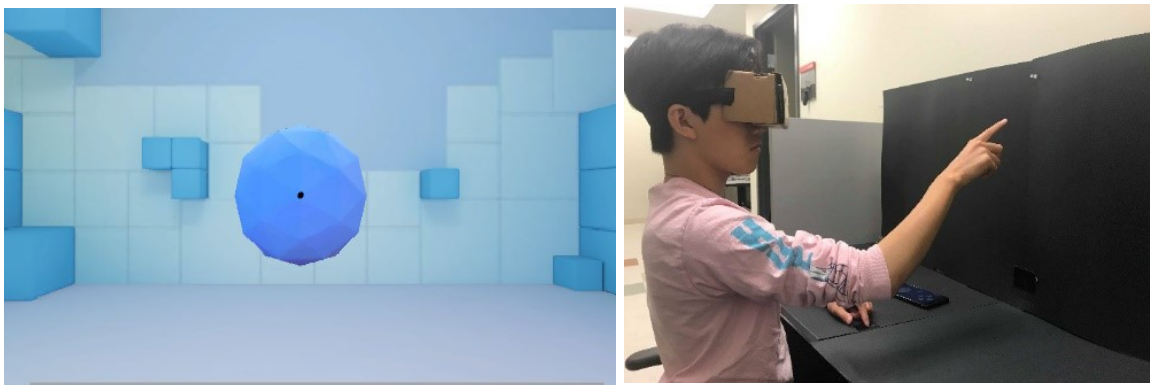


Figure 7. Virtual environment(left) and user study environment setup(right).

3.3.2 Software Development Challenges

Initially, I designed the software using 12 same-sized spheres arranged in a ring centerd on the center of the view. The smallest target size was considerably smaller (radius 0.1 m) than that finally used in the experiment. However, pilot testing revealed that performance was very low with air touch. This is because the hand-tracking reliability of the mobile camera was related to the hand position. For example, when the target was located in the lower part of the target ring (i.e., corresponding to the bottom of the camera viewing area) the tracking accuracy was not good enough to capture the fingertip's position; the input was simply too noisy to select targets this small. Accordingly, I adjusted all target sizes to be larger and only one object appeared on screen at a time. This also avoids the problem of accidental selections of the wrong target due to input noise.

In addition, since the hand gesture condition may potentially result in the failure selection by hardware problems, I set a timeout (15s) to exclude some trails. A failed selection is defined as a target that cannot be selected due to technical issues. Such failed trials were excluded from analysis; I filtered these trials after the experiment during data analysis. Only trials that were deemed "successful" (i.e., ended upon selection of the target,

or upon missing the target in under the timeout) were included in the analysis.

Since the built-in mobile device RGB camera has no depth sensor, the depth coordinates of the tracked hand were determined by the Manomotion SDK proprietary algorithms. The further the hand moves from the camera the larger the reported *z* value. To ensure the farthest targets were still reachable with the air touch condition (which required directly touching targets and hence precision in depth) I iteratively adjusted a scale factor between the VE and the Manomotion provided depth coordinates many times. In the end, a distance of 2 m in the VEs mapped to approximately 70 cm of actual hand motion in reality. This ensured that the farthest targets (2 m into the screen) were still reachable in a seated position with air touch.

3.4 Experiment design

The experiment employed a 3×2×3 within-subjects design. The three independent variables included selection technique, object size, and object depth (the distance between objects and the player's eyes).

Selection technique: Head ray (HR), finger ray (FR), and air touch (AT);

Object depth: close (1.3m), medium (1.7m) and far (2m);

Object size: big (0.7m) and small (0.4m);

For each selection technique, participants completed 6 blocks (3 object depths × 2 object sizes) of 12 selection trials, for a total of 72 selections with each selection technique per participant. Each selection trial required selecting one target sphere. Within a block, both target depth and target size were constant. Target depth increased with experiment block, and target size order was counterbalanced. Each selection technique was crossed with 2 target sizes and 3 target depths. I conducted 12 selection trials for one combination of one target size × one target depth. Selection technique order was counterbalanced

according to a Latin square. Overall, there were 12 participants × 3 selection techniques × 2 target sizes × 3 target depths × 12 selections = 2592 trials in total.

The dependent variables included movement time (s), error rate (%) and throughput (bit/s). Movement time was calculated from the beginning of a selection trial when the target appears, to the time when the participant confirmed the selection by pressing the button on the secondary touchpad. The error rate was calculated as the percentage of trials where the participant missed the target in a given block. Throughput was calculated according to Equation 2 presented in Chapter 2, section 2.6. Finally, I also collected subjective data using questionnaires and interviews after each participant completed the experiment.

3.5 Procedure

First, participants were given an introduction and explanation of the whole experiment. After they signed the consent form, they filled out a questionnaire asking about their experience with VR. Next, they sat down and put on the HMD, and I gave them instructions about how to control each selection technique and gave them about a minute to practice using the system. These practice trials were not recorded.

During testing, the first target sphere would appear at the center of the viewport. After selecting the first target, the formal test began. Participants confirmed each selection by tapping the touchpad button. Upon tapping the touchpad button, the current target disappeared immediately, and the next target appeared, whether the target was hit accurately or not. Targets appeared in the VE following the ring pattern common to ISO 9241-9 evaluations, as described above [15]. See Figure 6. Upon completing one condition, which consisted of 72 trials (12*2*3), participants were given approximately 1-2 minutes to rest and get ready for the next condition. After all conditions were completed, they filled

out the questionnaires and were interviewed for subjective feedback.

3.6 Result and Discussion

Results were analyzed by using repeated-measures ANOVA at 5% significance level. Full ANOVA test result can be found in 5.2Appendix G

3.6.1 Movement time

Movement time (MT) is the average time to select a target. Mean movement time for each interaction method is seen in Figure 8. Movement time was analyzed using repeated-measures ANOVA. The result ($F_{2,18}$=33.98, $p < 0.05$) shows that selection method had a significant main effect on MT. Overall, air touch took much longer to select targets than the other selection techniques at 1.86 s, about 50% slower than head ray and almost twice as long as finger ray. Head ray was, on average, slower than finger ray. Post hoc testing with the Bonferroni test (at the $p < 0.05$ level) revealed that the difference between all of selection techniques was significant. This result indicates that air touch performed much worse in terms of speed and finger ray was much more efficient than air touch and head ray.

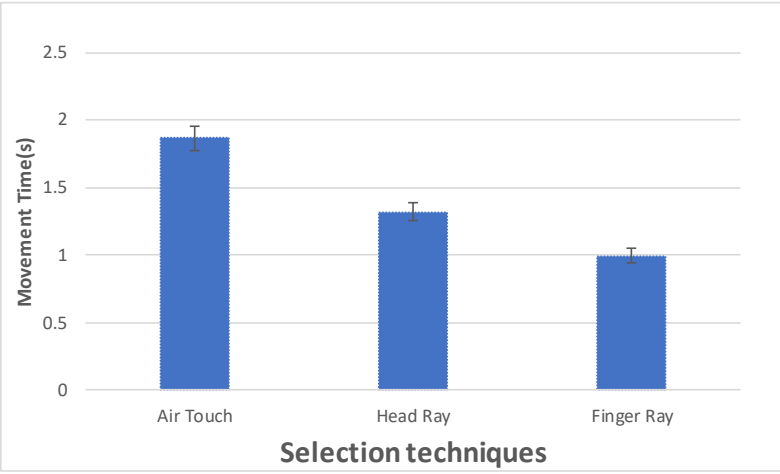


Figure 8 Overall mean movement time by selection techniques. Error bars show ±1 SD.

Figure 9 below illustrates the average movement time for each selection method,

further separated by target size and target depth. There were significant interaction effects between selection technique and target size ($F_{2,18}$=3.906, $p < 0.05$), as well as selection techniques and target depths ($F_{4,36}$=2.98, $p < 0.05$). Predictably, smaller size targets took longer for all selection techniques. The effect ($p < 0.05$) was most pronounced with air touch, where small targets were about 60% worse than large targets. Target size only affected ($p < 0.05$) the ray-based techniques when the distance was close or medium. There was no significant interaction effect between target depth and selection technique.

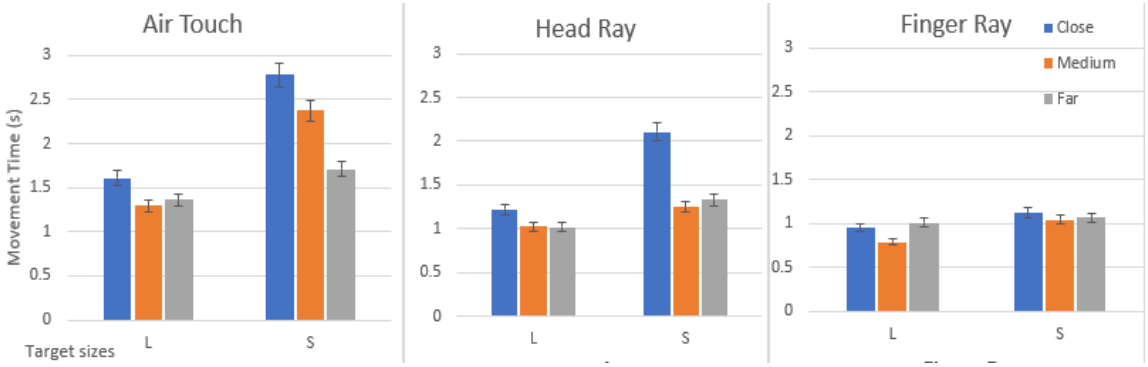


Figure 9. Movement time by different target sizes (Large and Small), depth (Close, Medium and Far) and selection techniques (Air touch, head ray, and finger ray). Error bars show ±1 SD.

I had initially predicted that air touch would take longer to select targets than finger ray and head ray. Air touch necessitated a "two-stage" selection – first, lining up the target in the plane, then adjusting the hand along the depth axis to select the target. In contrast, since finger ray and head ray are both ray pointing techniques, participants only needed to work in a plane; depth was handled automatically.

Camera noise was also expected to affect selection speed as well. It is encouraging that despite this camera noise (which was not a factor with head ray) finger ray still offered faster selection, especially with smaller targets. As noted earlier, the combination of large target size and closer target distances (close and medium) resulted in a significant difference in movement time between head ray and finger ray. This is likely because head

ray required more head motion than finger ray, which supported subtle finger or arm movements. This is consistent with previous research that also found that finger-based selection was faster than the head when reaching a target [28].

3.6.2 Error rate

A selection error is defined as missing the target, i.e., performing the selection while the cursor is outside the target. Error rate is calculated as the percentage of targets missed for for each experiment block (i.e., 12 selections). Average error rates for each selection method are seen in Figure 10. The ANOVA test result ($F_{2,18}$=385.52, $p < .05$) revealed that there was a significant main effect for selection technique. Post-hoc testing with the Bonferroni test (at the $p < 0.05$ level) showed there were significant differences between each pair of them as well ($p < 0.05$). Air touch yielded a significantly higher error rate than other two selection techniques, five times that of head ray, and around double that of finger ray. Moreover , there was significant interaction effects between selection technique and

target size ($p < 0.05$) as well as selection technique and and target depth ($p < 0.05$).

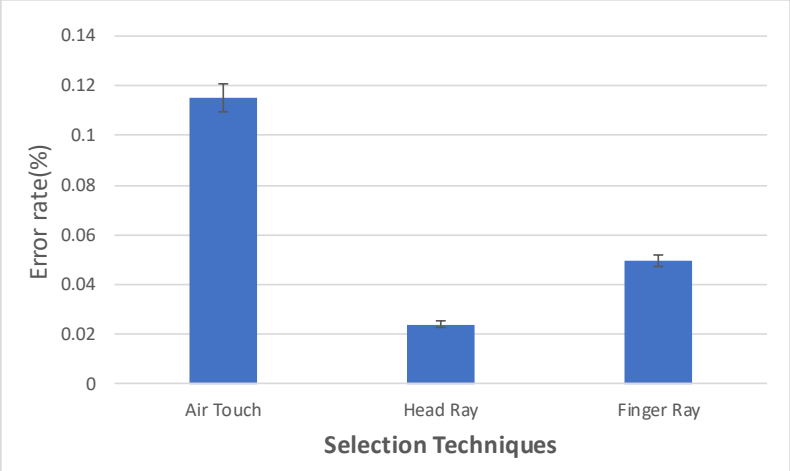
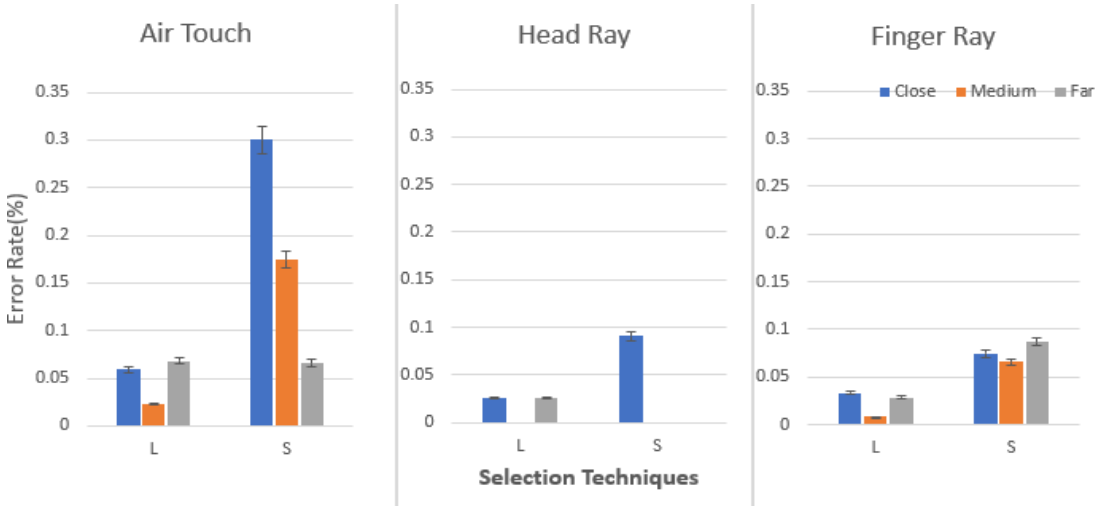


Figure 10. Error rate (%) in different input methods. Error bars show ±1 SD.



shows error rates separated by target depth and target size for each selection technique. Interestingly, post-hoc testing with the Bonferroni test (at the $p < 0.05$ level) showed no significant difference between the three target depths with either finger ray or head. This is likely both of the ray-based techniques did not require accuracy in depth. However, there were significant differences between target depths with air touch. This is unsurprising, given the technique required participants position their finger at the correct depth with air touch. Overall, medium distance (1.7m) had the lowest error rate for all

selection techniques. Air touch and head ray performed significantly worse at close distance than other two depths.

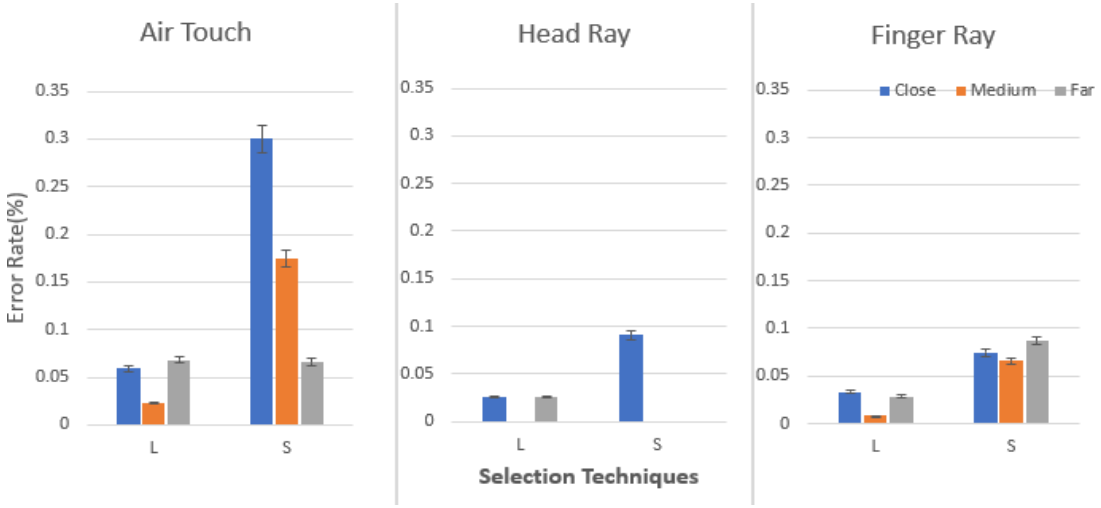


Figure 11. Error Rate for two target sizes, three target depths, and three selection techniques. Error bars show ±1 SD.

Post-hoc testing revealed a significant difference ($p < 0.05$) between target sizes with air touch, but not the other two selection techniques. The error rate increased dramatically with the smaller target size when using the air-touch interaction method. Generally, smaller target sizes had a higher error rate for each selection technique.

As seen in Figure 11, air touch was much less precise and had a higher error rate than other two selection techniques, especially with different target depths. This indicates that participants had difficulty identifying the target depths when using direct touch. Conversely, this was not a problem with the ray techniques. Medium distance produced the lowest error rate for all selection techniques.

3.6.3 Throughput

Throughput (TP) is a metric which combines both speed and accuracy to evaluate the performance of a selection technique [21]. I calculated throughput for the three selection techniques as described in Section 2.6. The average throughput for each selection technique

is seen in Figure 12. ANOVA revealed that selection technique had a significant main effect on throughput ($F_{2,18} = 90.63$, $p < 0.05$). Post-hoc testing with the Bonferroni test (at the $p < 0.05$ level) showed that all three selection techniques were significantly different from one another ($p < 0.05$). Finger ray had the highest throughput at 2.62 bit/s, followed by head ray (1.88 bit/s) and then air touch (1.01 bit/s). Throughput for head ray was in line with previous work [28] which reported about 1.9 bit/s when using a similar head-based selection method. Finger ray also offered higher throughput than using a mouse in a HMD-based VR environment. This suggests there is merit to using camera-based finger tracking as an alternative to common head-based selection in mobile VR.

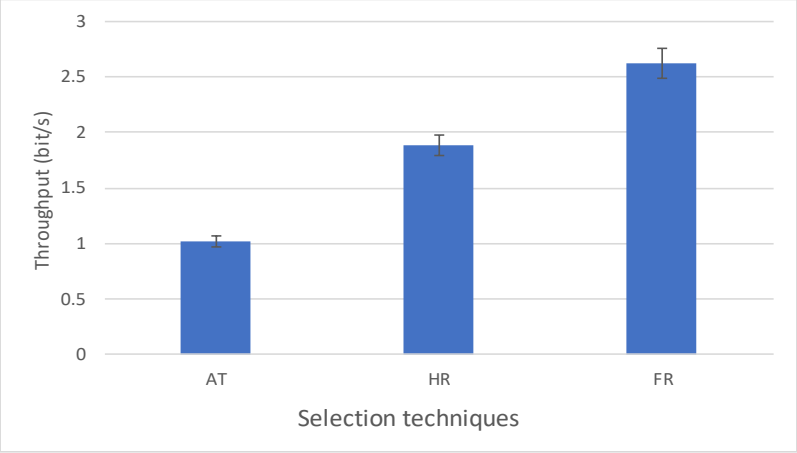


Figure 12. Throughput for three selection techniques. Error bars show ±1 SD.

3.6.4 Subjective data

Participants completed a questionnaire ranking the selection techniques. Overall, air touch scored lower than the other two. Most participants gave air touch 3 out of 5 and the rest of them gave it a lower score. All participants gave finger ray and head ray 3 or higher score out of 5. The average score for finger ray (4.1) is slightly higher than head ray (4.0). See Figure 13.

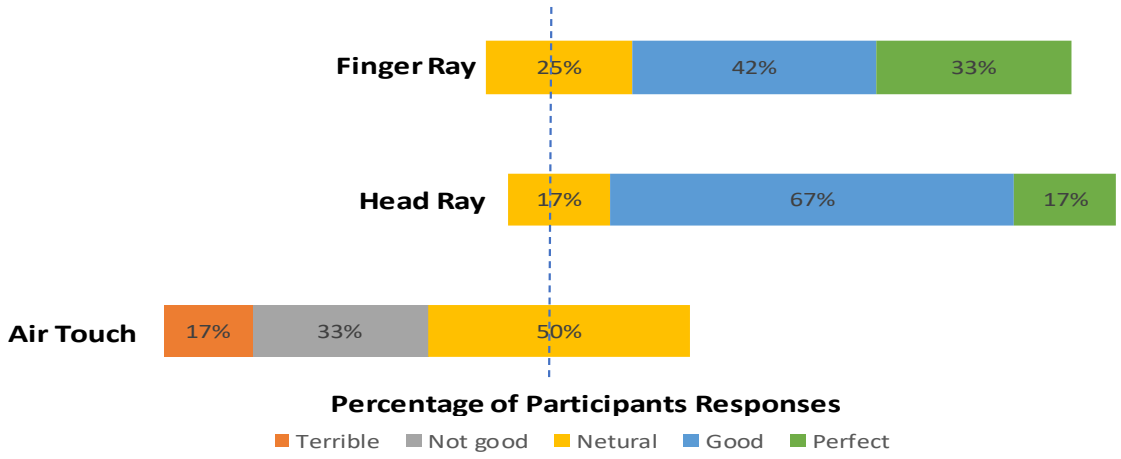


Figure 13 Number of participants giving each score in the subjective questionnaire ranking preference for each interaction method.

3.6.5 Interview

I conducted an interview with each participant after Experiment 1 and asked their feedback about each selection technique. Most participants mentioned physical fatigue with the air touch condition. They felt it difficult to hit the target because they needed to adjust their hand position forward and back constantly to find the target position in depth. This result is similar to previous research on visual feedback in VR [30], which reported the same "homing" behaviour, and found that highlighting targets on touch increased movement time, but decreased the error rate.

Physical fatigue was especially high at the farthest target distance; participants had to stretch their arms further to reach the targets, which made their upper arms and shoulder even more tired. Besides, as reported in previous work [29], stereo viewing appeared to be insufficient for them to reliably detect the depth of targets with air touch, necessitating the use of extra visual feedback. Although I added colour change upon touching a target and used a room environment to help facilitate depth perception, it seemed participants still had

difficulty determining target depth.

Head ray also yielded some neck fatigue, especially when targets were very close. Close targets increased the amount of required head motion as compared to farther targets, as targets could be potentially partially outside the field of view, necessitating greater head movement. Several participants noted that the Google Cardboard HMD felt uncomfortable on their nose as well. Before the experiment, I noticed this myself and put an extra nose pad to compensate, but this appears to have been insufficient as several participants mentioned it.

Despite the poor performance results, two participants reported that they preferred air touch because the found "it is interesting to really use my finger to touch blue balls rather than just turning around my head and touching the button." They found the latter "very boring."

All participants found that head ray was most efficient. "It is very easy to control and fast." However, 10 participants said they would choose finger ray as their favourite because "it is convenient to move my fingers slightly to hit the target. I do not even need to move my head and arms."

3.7 Summary

Overall, contrary to my initial hypotheses, head ray yielded the lowest error rate, while finger ray had the advantage on speed. Medium distance (1.7 m) and large target size (0.7m) offered higher accuracy, consistent with previous research [13].

Air touch yielded higher fatigue in the shoulder and forearm, and subsequently was much less preferred. This is likely largely affected by camera noise; while this was also a factor with finger ray, depth detection with a single RGB camera is much more problematic

than determining the 2D coordinate of the fingertip for finger ray.

Overall, ray-based techniques were superior to the hand-based air touch. I next present an experiment focusing on using different selection activation methods, noting the earlier design decision to use a secondary touchpad to activate the selection.

4 Chapter: Experiment 2

From Experiment 1, I concluded that finger ray and head ray can offer higher selection performance than air touch in mobile VR scenarios. However, an external tool, a touchpad, was used as selection activation mechanism. The main motivation of this thesis was to assess the effectiveness of interaction methods for low-cost mobile VR HMDs. As argued earlier, interaction without any external controllers would offer better portability. Moreover, the action of confirming a selection also affects selection accuracy [1]. Hence, choosing proper selection confirmation methods is also important for overall selection task performance. Consequently, Experiment 2 focuses on evaluating different selection activation mechanisms – in other words, how the user "clicks" a target. The options studied included the built-in Google Cardboard button (CB), a hand gesture tracked with the smartphone camera (HG). I compared these to the corresponding conditions using the touchpad (TP) from Experiment 1. I also reused the same experimental software environment from Experiment 1.

According to previous research, the index finger performs better than other fingers on back of device interaction [33]. Simple gestures also generally yield higher performance than complex ones and are easier to remember. My hand gesture condition thus used the "Tap" gesture. This is the gesture performed when selecting an icon on a touchscreen device (Figure 14) [25]. I originally considered a different gesture, Pinch, which involved bending index finger and thumb like a "C" shape, then closing the fingertips. However, I decided against using this gesture as I found the tracking SDK was unable to reliably detect the Pinch gesture. As a result, the Pinch gesture usually took longer to complete a selection than Tap. Consequently, I chose Tap as an easier and more reliably recognized gesture.

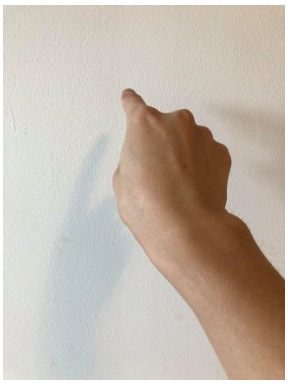


Figure 14. Air Tap hand gesture

4.1 Hypothesis

The touchpad data from Experiment 1 was included as a basic comparison point, since it is similar to clicking a mouse. When using the touchpad, users only need to slightly lift a finger up over touchscreen and press down, yielding a potentially shorter movement time. In contrast, pressing the Cardboard button may take more time since the depth of button is longer. Moreover, the hand gesture is likely affected by several factors, including the individual factors (e.g., finger flexibility/dexterity) but more importantly, tracking quality. Based on these observations, I hypothesized the following:

H1: The combination of finger ray + touchpad condition will retain the fastest-speed spot, since finger ray was faster than head ray in Experiment 1 and the touchpad has higher sensitivity.

H2: The combination of finger ray + hand gesture condition will produce the second highest speed as the gesture can be performed directly with same hand and there is no need to operate two hands.

H3: The combination of head ray + Cardboard button will produce the lowest error rate because of the comparative reliability of the hardware.

4.2 Participants

The same 12 participants (2 females and 10 males) who participated in Experiment 1 also participated in this experiment right after. Their ages ranged from 18 to 30 years old (Mean≈ 22.67 years old). Two were left-handed. None had vision or motor problems.

4.3 Apparatus

I used the same hardware setup as in Experiment 1, including the Samsung Galaxy 8 smartphone, and Google Cardboard HMD.

I also used the same software environment in Experiment 2 as in Experiment 1. See Section 3.4.1 for a full description. This experiment used a subset of the selection techniques from Experiment 1. These included head ray and finger ray. Descriptions of those techniques can also be found in Chapter 3, before Section 3.1.

The two new selection activation methods studied included the Google Cardboard button and the Tap hand gesture described earlier. These, like the touchpad selection activation method all resulted in the present target disappearing immediately, and the next target appearing.

In the Cardboard button condition, participants pressed the capacitive button built into the cardboard frame. I note that since I had expected most participants would be right-handed, I added a second button on the left side of Google Cardboard (see Figure 15). After all, the built-in right-side Cardboard button would prevent participants from using their right hand to perform hand postures (e.g., in the finger ray selection technique) and thus would likely affect performance. Adding the left-side Cardboard button ensured that participants could always select with their dominant hand, and activate selection using their non-dominant hand.

Figure 15. Modified Google Cardboard with both left and right-sided buttons.

In the hand gesture condition, participants performed the "air tap" gesture [25] described above within the camera viewport to accomplish the selection. See Figure 14 This was done using the dominant hand when using the finger ray selection method (i.e., the same hand used to determine the ray direction). Participants were told to use the same hand whatever the selection technique is. Participants had to keep the pointing finger stable until the tap gesture is performed. Finally, the touchpad operated the same as in Experiment 1 and used the same extra hardware.

4.4 Experiment Design

To investigate potential interactions between selection technique and selection activation methods, this experiment included both finger ray and head ray from Experiment 1. The experiment thus employed a 2×2×3×2 within-subject design with the following four independent variables:

Pointing method: Head Ray (HR), Finger Ray (FR);

Selection activation mechanisms: Cardboard button (CB), hand gesture (HG);

Object depth: close (1.3m), medium (1.7m) and far (2m);

Object size: big (0.7m) and small (0.4m);

Participants completed Experiment 2 immediately following Experiment 1. Each block consisted of 12 selection trials for each combination of target size and target depth. Selection techniques and selection activation mechanisms were counterbalanced according to a Latin square. Overall, there were 12 participants × 2 pointing methods × 2 selection activation mechanisms × 2 target sizes × 3 target depths × 12 selections = 3456 trials in total.

I calculated dependent variables including movement time (s), error rate (%) and throughput (bit/s). Movement time was calculated from the beginning of a selection trial, the target appears, to the time when the participant confirms the selection. The error rate was calculated as the percentage of trials missing the target in each block. Throughput was calculated according to Chapter 2, section 2.6. Finally, I also collected subjective data by questionnaires and interviews after each participant accomplished experiments.

4.5 Procedure

Experiment 2 immediately followed Experiment 1. First, participants sat down and put on the HMD, and I gave them instructions about how to control each pointing method and selection activation mechanism. I gave them about a minute to practice using the system. These practice trials were not recorded. The first target sphere would appear at the centre of the viewport. After selecting the first target, the formal test began. Participant confirmed each selection by different mechanisms, no matter the target was hit accurately or not, the current target disappeared immediately, and the next target appeared. Targets appeared in the VE following the ring pattern common to ISO 9241-9 evaluations [15]. Upon completing one condition (1 pointing method × 1 selection activation mechanism), which consisted of 72 trials (12*2*3), participants were given approximately 1-2 minutes to rest and get ready for the next condition. After all conditions were completed, they filled

out the questionnaires and were interviewed for subjective feedback.

4.6 Results

Results were analyzed by using repeated-measures ANOVA at 5% significance level. For all dependent variables, I included data from Experiment 1 for the finger ray and head ray conditions. Since Experiment 1 exclusively used the touchpad, this was added as a basis of comparison with the two new selection activation mechanisms. Full ANOVA test result can be found in 5.2Appendix G

4.6.1 Completion time

Completion time (CT) is the average time to select a target. Mean completion time for each condition combination is seen in Figure 16. Completion time was analyzed using repeated-measures ANOVA. The result ($F_{5,20}$ =22.44, $p < 0.001$) shows that the combinations of selection techniques and selection activation mechanisms have significant effects on completion time. Post hoc testing with the Bonferroni test (at the $p < 0.05$ level) revealed there a significant difference ($p < 0.05$) between head ray and finger ray when using the Cardboard button. Pairwise differences are visualized in Figure 16 as arrows indicating condition pairs that are significantly different. There was no significant difference when using hand gestures between finger ray and head ray, nor when using the touchpad between head ray and finger ray. Also, there were significant differences between touchpad and either Cardboard button or hand gesture when using finger ray as the selection technique.

Finger ray with either Carboard button or hand gesture had much higher completion time compared to the touchpad. Finger ray also took longer with Carboard button than head ray with Carboard button. This is surprising given how fast the condition was when using the touchpad in Experiment 1. This highlights the importance of investigating selection

activation mechanisms in conjunction with pointing techniques.

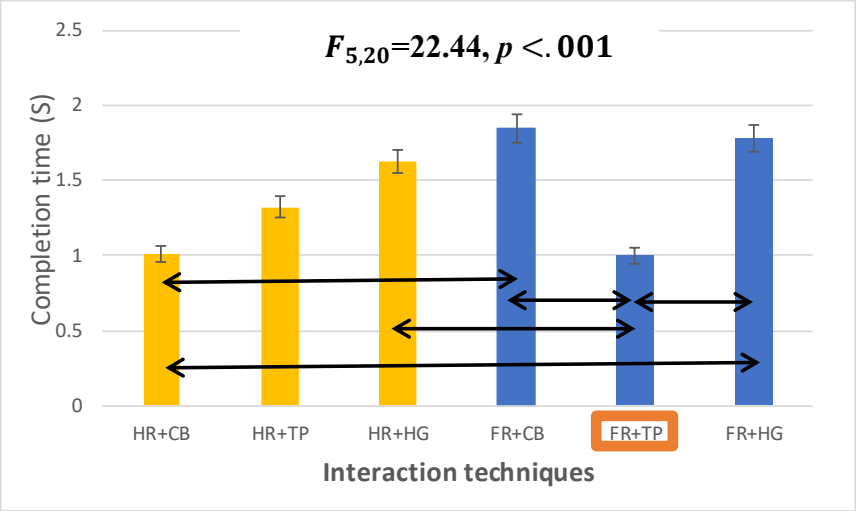


Figure 16. Average completion time for combinations of techniques. Error bars show ±1 SD Two-way arrows (←→) indicate the pairwise significant different between them with post hoc test at 5% significance level, and best performance one was highlighted by red.

Results separated by target depth and size are seen in Figure 17. Generally, smaller target size required a longer time to complete selections. Like Experiment 1, the medium target depth yielded faster movement times compared to the far and close target distances. The finger ray + touchpad still hold the top which takes the shortest time to complete selections for every target size and depth.

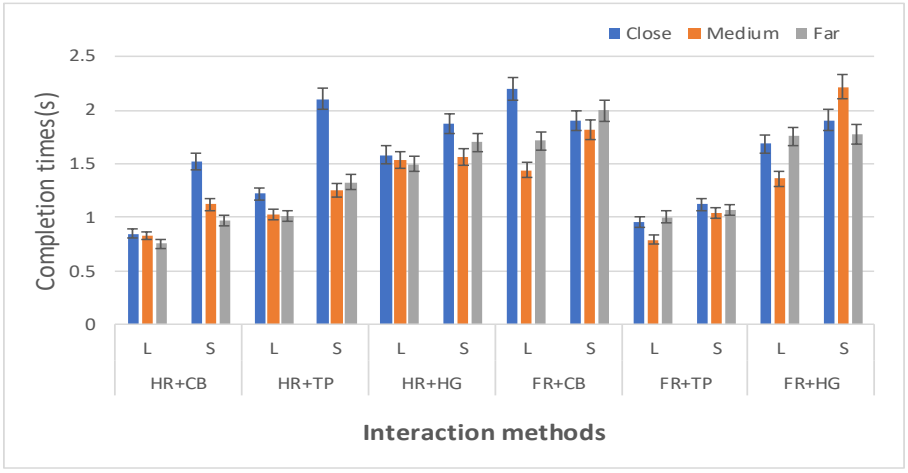


Figure 17. Completion time in depths, sizes, and combination techniques. Error bars show ±1 SD.

4.6.2 Error rate

Average error rates for each technique combination is shown in Figure 18. Repeated-measures ANOVA result ($F_{2,8}$ =16.57, $p < 0.001$) shows that the combinations of techniques have a significant effect on error rate. Post hoc testing with the Bonferroni test (at the $p <. 05$ level) revealed there were significant differences between finger ray + Carboard button and using head ray with all three selection activation mechanisms. Pairwise differences are visualized in Figure 18 as arrows indicating condition pairs that are significantly different. The highest error rate was with the finger ray + Cardboard button condition. Notably, hand gestures worked better with both finger ray and head ray, than either selection method worked with Cardboard button, which had highest error rate for both selection methods.

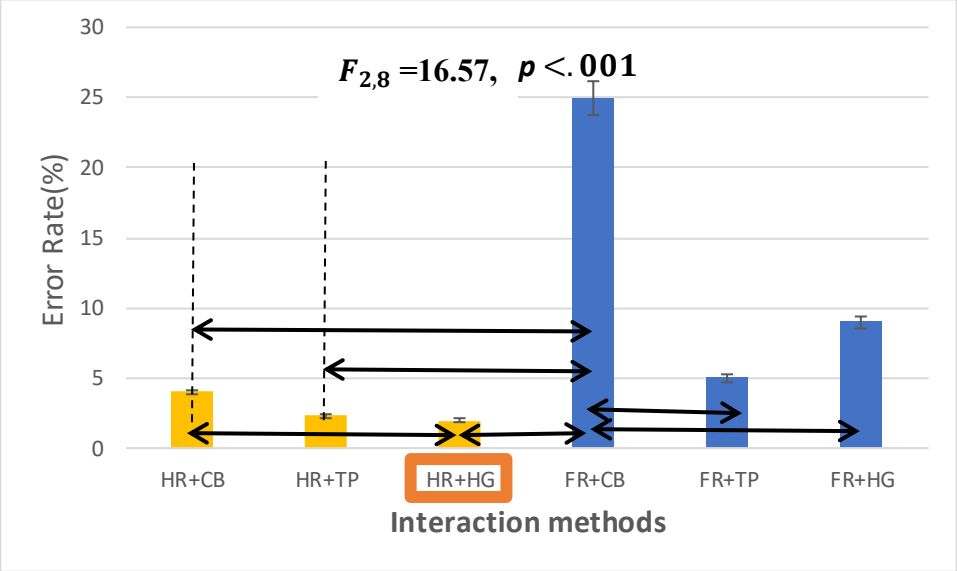


Figure 18 Error rate in combinations of techniques. Error bars show ±1 SD. Two-way arrows (↔) indicate the pairwise significant different between them with post hoc test at 5% significance level, and best performance one was highlighted by red.

According to a Bonferroni post hoc test (at the p = 0.05 level), target depth did not have significant effects on error rate except finger ray + hand gesture combination. Finger

ray + Cardboard combination had higher error rate in each combination of target depths and sizes than other interaction methods (see Figure 19).

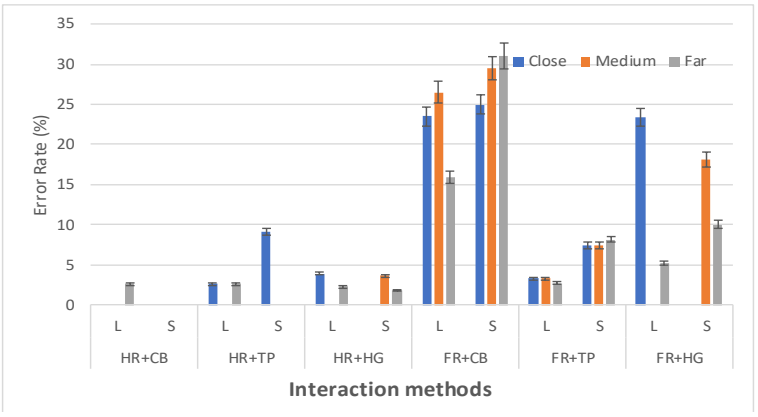


Figure 19. Error rate by techniques combination. Error bars show ±1 SD.

4.6.3 Throughput

Repeated measures ANOVA ($F_{2,8}$=70.08, $p < 0.001$) revealed that the combinations of techniques have a significant effect on throughput. Average throughput for each technique combination is shown in Figure 20. Post hoc testing with the Bonferroni test (at the p <0.05 level) shows when using head ray as selection technique, the throughput was significantly different between each selection activation mechanism ($p < 0.05$), with hand

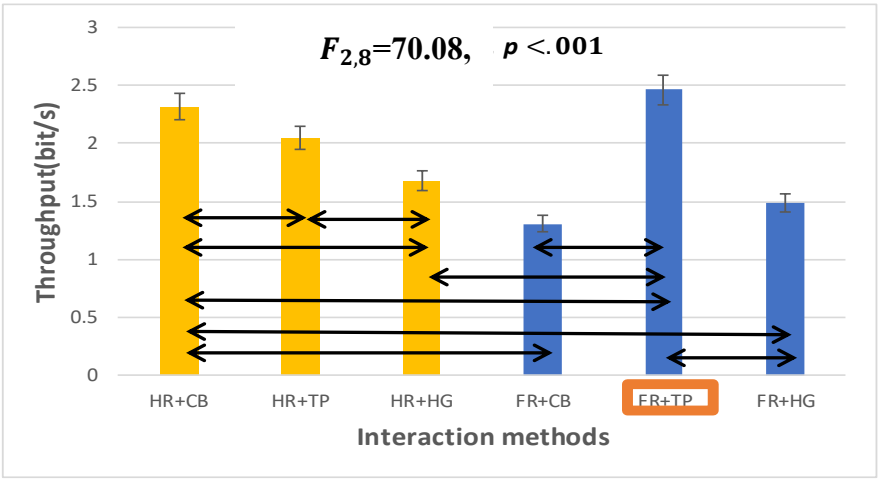


Figure 20 Throughput for each combination technique. Error bars show ±1 SD. Two-way arrows (↔) indicate the pairwise significant different between them with post hoc test at 5% significance level, and best performance one was highlighted by red.

gesture performing worst and Cardboard button performing best. Pairwise differences are visualized in Figure 20 as arrows indicating condition pairs that are significantly different. With finger ray, throughput was significantly lower with hand gestures and the Cardboard button than with the touchpad.

4.7 Subjective results

From the questionnaire results (see Figure 21), participants rated head ray + touchpad as lowest and head ray + Cardboard button as the highest. Finger Ray + touchpad is at the second position, flowed by finger ray + Cardboard button combination. Upon interviewing participants, they mentioned that performing with hand gestures was more convenient than pressing the button or touchpad. Notably, the Cardboard button was sometimes a bit unresponsive, requiring they press it harder at times. Some also mentioned that the HMD was not tight enough when they pressed down the button. They had to hold it with another hand sometimes. No participants mentioned any physical fatigue in Experiment 2, even though they needed to perform hand gesture in the air.

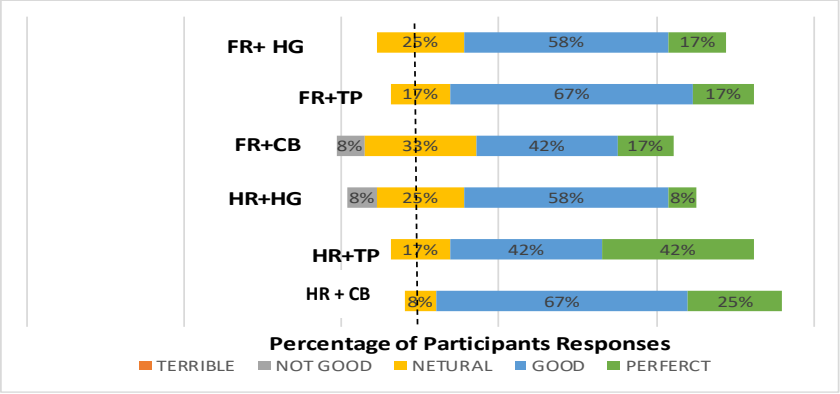


Figure 21 Number of participants giving each score in the subjective questionnaire ranking preference for each combination of techniques.

4.8 Discussion

When using the finger ray with either hand gestures or the Carboard button, selection

performance was notably worse. This was a surprising result, given that in Experiment 1, finger ray was faster than the head ray. This suggests that either hand gesture or Cardboard button might not a suitable selection activation mechanism to work with finger ray.

As I expected from hypothesis *H1*, finger ray with touchpad was fastest. This is likely because the touchpad was more sensitive than either the Cardboard button or using hand gestures recognized by the camera. As mentioned earlier, participants sometimes found the Cardboard button somewhat unresponsive. Similarly, the hand gestures were not always perfectly recognized by the tracking SDK. On the other hand, using head ray with the Cardboard button yielded significantly lower completion times. I suspect this is because head and neck movements resulted in more whole-body movement than simply using the fingers with finger ray. For example, when using the head ray, participants had to turn their bodies slightly to face the target. During such movement, it was faster to press the Cardboard button (since it is positioned on the HMD) rather than tapping the touchpad (which is fixed on the table).

Finger ray + hand gesture took longer than head ray + hand gesture, which was inconsistent with my hypothesis *H2*. I had expected that the hand gesture could be performed with the same hand being used to perform the finger ray selection so that completion time would be less with finger ray + hand gesture combination. I had thought this would be a fast process, since after all, participants could perform the hand gesture as soon as the ray intersected the target, which may thus be faster than pressing the Cardboard button. The comparative lack of camera sensitivity likely explains this result. Usually, the participants' tap gestures were not recognized on the first try; multiple hand gestures thus increased the time to select targets.

I was surprised by the significantly higher error rate for finger ray with the Cardboard

button. This may be because of the so-called "Heisenberg" effect in 3D selection [3], where the selection activation mechanism sometimes moves the pointing device at the instant of selection, resulting in missing the target. In my case, I observed when pressing the Cardboard button, their bodies and head would move slightly, which likely influenced the head ray. Similarly, since the HMD was not as secure as more expensive HMDs, pressing the button also moved the HMD slightly, further affecting the selection ray. However, when using head ray, participants frequently used their other hand to hold the Google Cardboard, so the error rate was fairly stable. However, this factor still influenced the overall error rate, contrary to my expectation (*H3*) that error rate for head ray + Cardboard should be lowest. Head ray + Cardboard button had a higher error rate than head ray + hand gesture combination. In contrast, when using the finger ray, participants used one hand to direct the ray, and the other to press the button. As a result, the error rate increased in that condition. Due to the overall better movement time and accuracy with the Cardboard button, throughput was also higher with this condition.

From the questionnaire data and interviews, the head ray + Cardboard button was rated best, followed by finger ray + Cardboard button. This indicates that smooth operation during pointing is an important factor for users. Further, there was no physical fatigue was reported during the experiment 2, which might because the air touch condition in Experiment 1 caused the physical fatigue reported by participants. This also suggests that using finger ray causes much lower level fatigue than air touch. In general, head ray + Cardboard button still had the advantage on both throughput, error rate and completion time. However, both head ray and finger ray with hand gestures were not far off, and could be a potential alternative in the future, especially with advances in camera-based tracking.

4.9 Summary

Above all, it is difficult to simply conclude which technique combination is the best from my experiment result since they have different advantages and disadvantages. However, the combination of head ray and the Cardboard button achieved relatively stable performance in every metric. As I hypothesized in H3, this combination would further benefit from greater physical stability of the hardware. Head ray + hand gesture offered lower error rates than head ray + cardboard button despite hardware noise (e.g., jitter). This result supports the idea that using the finger offers good utility in mid-air selections [37]. Moreover, although finger ray + hand gesture had higher error rate than head ray + cardboard button, there was no significant difference between them. This suggests the higher average error rate may just be due to only a few participants with poor performance. I suspect that the combination of finger ray or head ray plus hand gestures could produce better performance with more reliable camera detection. Finally, although the touchpad offered good performance overall, I again argue that it is not a practical choice for mobile VR scenarios.

5 Chapter: Conclusion

In my work, I focused on comparing potential selection techniques for low-cost mobile VR. My objective was to assess if alternatives to common head-based selection methods were feasible with current technology, especially employing computer vision tracking approaches on the mobile. To this end, I simplified my hardware condition to only a smartphone and a cardboard-made HMD. In Experiment 1, I compared three selection techniques, air touch, head ray and finger ray in selection tasks, finding that overall, air touch performed worst. In Experiment 2, I compared different combinations of selection techniques and selection activation methods. Results indicated that an external touchpad worked well with finger ray, despite its impracticality. The built-in Cardboard button worked well with head ray.

While previous researches proposed some new interaction techniques for Cardboard type HMD devices, nobody has evaluated the performances of different selection techniques and selection confirmation methods with ISO 9241-9 [15] standard and Fitts' law. My work measured performance by several metrics including error rate, movement time, and throughput as well as participant feedback. From experiments results and analysis, I suggest that finger ray is a promising input technique for mobile VR. Although head ray had the highest accuracy, finger ray was competitive, especially in light of sensor noise in the tracking SDK. Despite this difficulty, finger ray had higher speed and throughput than head ray and air touch. Besides, finger ray had high preference as well from the post-test questionnaire and interviews. Air touch, in contrast, it is not an ideal choice for simple selection tasks; single-camera tracking of the hand seems to be currently incapable of reliably determining hand depth. My user study shows that dealing with higher DOFs might cause lower performance, which is consistent with previous research [9]. Besides,

comparing to two ray techniques, air touch caused much more obvious physical fatigue in both arm and shoulder areas. Head ray caused neck fatigue as well when pointing to close targets. Overall, from Experiment 1, there seems to be some promise for future selection techniques based on camera tracking for mobile VR scenarios.

In Experiment 2, different combinations between the two ray techniques (head ray and finger ray) and selection activation methods (hand gesture, cardboard button, touchpad) yielded surprising performance results. However, it is hard to conclude which combination is best for mobile VR scenarios from my experiment results, as each combination offered different pros and cons. Currently, it appears that the "default choice" (i.e., industry standard) head ray + cardboard button offers the most stable combination. Head ray +cardboard button is competitive on all metrics. However, I suspect that the combination of either finger ray or head ray plus gestures could offer higher performance if the gestures could be more reliably detected. This might be possible in future with more robust vision-based tracking software, which would likely necessitate more powerful smartphone hardware. Likewise, Cardboard button performance could increase with a more stable HMD configuration, reducing hardware noise. In addition, participants with the head ray can be easily controlled and confirm with the hand gesture is more natural.

Additionally, although the touchpad conditions worked well, I note again that these are not realistic for true mobile VR scenarios, as it necessitates additional hardware and setup. Another problem is that it is easy to lose such an external controller when using immersive VR.

Finally, compared to the result from Experiment 1, I can see the importance of choosing the proper selection activation mechanisms in relation with selection techniques. I hope my research results can help design of human-computer interface on mobile VR

devices in the future.

5.1 Limitations

My current work is based on lab settings with particular lighting levels and background colour. As discussed earlier, this was done to optimize conditions for the tracking SDK. Despite these optimizations, during the experiment, I still noticed constant cursor jitter. In real-world usage, interference factors would be much more complicated. Hence, I take my results as a "best-case" with current technologies.

Due to the extreme jitter, the target size was also limited by the camera. It would be impossible to hit very small targets; previous work has shown that once jitter approaches half the target size, selection accuracy falls dramatically [24]. In pilot testing, I modified the experiment interface to account for this problem, but in real-world VR selection scenarios, target size could be clearly be much smaller.

Another limitation is that I only used the tap gesture for the hand gesture condition. In pilot testing, the tap was deemed more reliable than the pinch gesture I also considered. However, other gestures may also be feasible, and could be tested in future work.

Finally, I note that the depth of the virtual hand in the air touch condition was calculated using a scale factor between Manomotion SDK and the VEs coordinate systems. However, the scale factor in the z-axis was fixed in my study, which was not ideal for all participants. For example, one participant with shorter arms had difficulty in reaching the farthest targets. Customizing this ratio for each participant would provide a smoother user experience.

5.2 Future work

The current studies only have used two levels of Fitts' index of difficulty. As a result, I did not get a strong linear regression for my pointing techniques. This limits the generality

of the results and could be improved in a future study using a broader range of target sizes and distances, yielding a more comprehensive range of task difficulties. This would allow for better prediction of indices of difficulty *not* tested, through calculation of linear regression models between *MT* and *ID* (See Equitation 1, Section 2.4).

Due to the aforementioned hardware constraints, I used the built-in RGB camera to detect the hands and gestures. The hand depth relied on calculations described earlier rather than pure detection. As a result, my smallest target size is just 0.4m in VEs due to the unreliability of the tracking. In future work, I propose to use a depth camera to get more accurate tracking. I speculate that in the future, many companies will begin to include depth cameras in their mobile devices, and some devices (e.g., Google Tango) are already available.

Appendices

Appendix A Consent form

CUREB clearance #: 10614216-121

Carleton UNIVERSITY
Canada's Capital University

Consent Form

Title: Camera-Based Selection with Low-Cost Mobile VR HMDs

Funding Source: NSERC

Date of ethics clearance: To be determined by CUREB (as indicated on the clearance form)

Ethics Clearance for the Collection of Data Expires: To be determined by CUREB (as indicated on the clearance form)

Researcher: Siqi Luo, Carleton School of Computer Interaction

---

This study involves one 60 minute experiment. With your consent, the session will be video recorded for analysis by usability software. The experiment involves wearing a head-mounted display (e.g., Oculus Rift) while performing tasks in a virtual environment, such as moving through the environment, and moving objects in the environment using the tracked 3D controller. Software will automatically and anonymously log your task performance data during participation. We may also (with your consent) take audio/video recordings for further analysis, for example, to study facial expressions to assess your user experience with our software.

Like most virtual reality systems, ours uses a stereo 3D display. Hence you must have normal or corrected-to-normal stereo viewing capabilities to participate – if you can see the 3D effect in 3D movies, you should be fine. There is thus also a minor risk of eye fatigue, headache, or nausea, consistent with the use of stereo displays (e.g., 3D movies) for about 10% of the population. Should you experience such symptoms, you are encouraged to take a break to alleviate the discomfort, or alternatively to withdraw from the study.

You have the right to end your participation in the study at any time, for any reason, up until completion of the experiment. If you withdraw from the study, all information you have provided will be immediately destroyed.

As a token of appreciation, you will receive $10 upon completion of the experiment. This is yours to keep, even if you withdraw from the study.
All research data, including audio/video recordings and any notes will be encrypted

and stored on the researcher's password-protected hard drive. Any hard copies of data (including any handwritten notes or USB keys) will be kept in a locked cabinet at Carleton University. Research data will only be accessible by the researcher.

Once the project is completed, all research data will be kept in anonymous form indefinitely, and potentially used for other research projects on this same topic. If you wish to withdraw your data or any audio/video recordings, please contact us and we will arrange to meet with you to do so. If you would like a copy of the finished research project, you are invited to contact the researcher to request an electronic copy which will be provided to you.

The ethics protocol for this project was reviewed by the Carleton University Research Ethics Board, which provided clearance to carry out the research. Should you have questions or concerns related to your involvement in this research, please contact:

CUREB contact information:
Professor Shelley Brown, Chair (CUREB-B)
Professor Andy Adler, Vice-Chair
Carleton University Research Ethics Board
Carleton University
511 Tory
1125 Colonel By Drive
Ottawa, ON K1S 5B6 Tel:
613-520-2517
ethics@carleton.ca

Researcher contact information:
Rob Teather
School of Information Technology
Carleton University
Tel: 613-520-2600 ext. 4176
Email: Rob.Teather@carleton.ca

Do you agree to be video-recorded: ____Yes ____No

I _____, choose to participate in a study on 3D user interfaces.

_____ _____

Signature of participant Date

_____ _____

Signature of researcher Date

This document has been printed on both sides of a single sheet of paper.
Please retain a copy of this document for your records.

Appendix B Pre-test Questionnaire

Result Page

Carleton UNIVERSITY

Selection with Low-cost mobile VR HMDs

Pre-test Questionnaire

Participant Number

1. I identify my sex as:

- 1- Male
- 2- Female
- 3- Other
- 4- Prefer not to answer

2. What is your age range?

- 1) 18 to 20
- 2) 21 - 29
- 3) 30 -39
- 4) 40 - 49
- 5) 50 - 59
- 6) 60 - 69
- 7) 70 or older
- 8) Prefer not to answer

3. What is your dominant hand?

- 1) left
- 2) right
- 3) both

4. Have you ever use any Virtual reality devices before ? How many times a month.

- 1) No
- 2) 1 to 5 times
- 3) 6 to 10 times
- 4) 11 to 15 times
- 5) 16 times or more
- 6) Prefer not to answer

5. Have you ever used Google Cardboard?

- 1) Yes
- 2) No
- 3) Not prefer to answer

6. If this is your first time that you experince VR please tell us your expectation and prediction abour the experiment?

Appendix C Post-test Questionnaire

Result Page

Carleton UNIVERSITY

1.How would you rate for headtracking pointing+touchpad?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

2. How would you rate for fingerRay+touchpad?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

3. How would you rate for air touch+touchpad?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

4.How would you rate for headtracking headtraking+button?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

5.How would you rate for headtracking headtraking+airtouch?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

6.How would you rate for fingerRay+button?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

7.How would you rate for fingerRay+airtouch?

Terrible	Not Good	Neutral	Good	Perfect
1 ☐	2 ☐	3 ☐	4 ☐	5 ☐

Appendix E ANOVA Analysis results for Experiment 1

Effect		Movement time		Error rate		Throughput	
	d.f.	F	p	F	p	F	p
Tech(T)	2,18	33.98	***	385.52	***	90.63	***
Size(S)	1,9	35.46	***	3.99	ns	11.38	**
Depth(D)	2,9	12.20	***	71.62	***	8.04	**
T*S	2,18	3.91	*	6.60	**	7.39	**
T*D	4,36	2.98	*	78.39	***	2.23	ns
S*D	2,18	2.86	ns	14.97	***	11.56	ns
T*S*D	4,36	2.30	ns	15.69	***	4.92	ns

Significance marks: *** represents <0.001, **<0.01)* represents <0.05

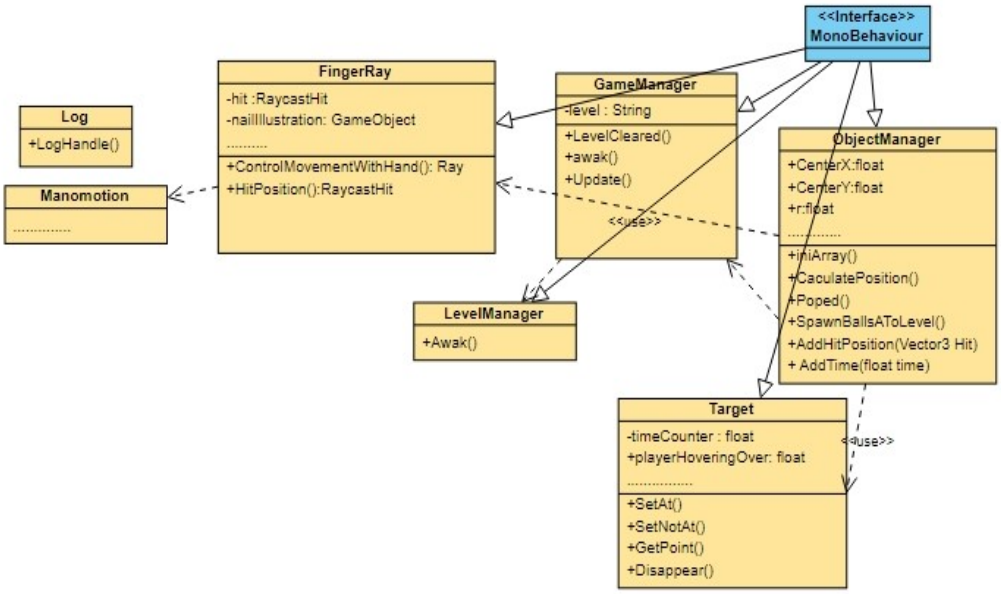
Appendix F ANOVA Analysis results for Experiment 2

Effect		Movement time		Error rate		Throughput	
	d.f.	F	p	F	p	F	p
Tech(T)	5,20	22.44	**	16.57	***	70.08	***
Size(S)	1,4	48.24	**	24.89	**		NS
Depth(D)	2,8		NS		NS		NS
T*S	5,20		NS		NS		NS
T*D	2,8		NS		NS		Ns
T*S*D	10, 40	3.43	*		NS	4.36	**

Significance marks: *** represents <0.001, **<0.01 ,* represents <0.05

Appendix G Software structure

I developed my software on Unity3D in C#. For different techniques, the software framework is slightly different. Main prefabs in Unity editor are Game Manager, Level Manager, Objects Manager, Player, Objects and Techniques. The UML below shows the relationship between these main classes for Finger Ray project.



GameManager class is mainly responsible for updating game level (e.g., depth increase) and switch scenes (target size).

ObejctManager class takes responsivity for set each target in specific locations by sequences and write coordinates of ray intersect with targets and time of each selection takes into console.

LevelManager class manages game's current level and max level.

Target class attached to each target prefab takes responsibility for setting colours based on selecting status. For example, when ray is intersecting with the target, set the target color into pink. Besides, it monitors if the target is selected successfully when the

selection is activated.

FingerRay class is responsible for creating finger ray. ControlMovementWithHand() method controls the movement of black dot following index fingertip position tracked by Manomotion class.

Due to the specific mobile scenario, I used a Log class to writes all logs data onto a txt file in the cellphone. Some important codes for each selection technique is listing below.

FingerRay class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class FingerRay : MonoBehaviour {
   private GameObject hitObject = null;
   private Camera camera;
   private Ray myRay;
   GameObject nailIllustration;
   RaycastHit hit;
   public float maxCursorDistance = 30;
   float detectedDepth, depthAdjustment;
   #region Singleton
   private static FingerRay _instance;
   public static FingerRay Instance
   {
      get
      {
         if (_instance == null)
         {
            GameObject bm = new GameObject("FingerRay");
            bm.AddComponent<FingerRay>();
         }

         return _instance;
      }
   }
   #endregion    void Awake() {
            depthAdjustment = 2f;
      nailIllustration=transform.GetChild(0).gameObject;

   }

   // Update is called once per frame
   void Update () {

      myRay = ControlMovementWithHand ();

      int  layerMask = 1 << 8;
      // Raycast
      if (Physics.Raycast (myRay, out hit, Mathf.Infinity,layerMask)) {

         Debug.Log (hit.collider.name);
      if (hit.collider.tag=="Ball") {
            hitObject = hit.transform.gameObject;

            this.hit = hit;
            hitObject.GetComponent<Ball>().SetGazedAt();
         if (Input.GetButton("Fire1"))
            {
            hitObject.GetComponent<Ball>().GetPoint();

               BalloonManager.Instance.AddHitPosition (hit.point);
         }
         }
      } else {

      if (hitObject != null) {
```

```
            hitObject.GetComponent<Ball> ().SetGazedAtNot ();
            hitObject = null;
        }

    }

}

private Ray ControlMovementWithHand()
{

    if (ManomotionManager.Instance.Hand_infos [0].hand_info.gesture_info.mano_class == ManoClass.POINTER_GESTURE_FAMILY) {

        naillllustration.SetActive (true);
        detectedDepth = Mathf.Clamp (ManomotionManager.Instance.Hand_infos [0].hand_info.tracking_info.relative_depth, 0f, 1f);
        Vector3 pointerPosition = ManomotionManager.Instance.Hand_infos [0].hand_info.tracking_info.finger_tips [3];
        Vector3 camPos = Camera.main.transform.position;
        //Vector3 pointerPosition=ManomotionManager.Instance.Hand_infos[0].hand_info.tracking_info.finger_tips[(int)(FingerTipIndex.POINTER_INDE
X)];
        Vector3 newPos = ManoUtils.Instance.CalculateNewPosition (pointerPosition, detectedDepth * depthAdjustment);
        Vector3 rayDirection = newPos - camPos;
        this.transform.position = Vector3.Lerp (this.transform.position, newPos, Time.deltaTime * 10);
        Ray PointerRay = new Ray (camPos, rayDirection);


        Debug.Log ("detectedDepth :" + detectedDepth + "pointerPosition :" + newPos);
        return PointerRay;


    }

    return myRay;

}

public RaycastHit HitPoint(){

    return this.hit;
}
}
```

GameManager class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class GameManager : MonoBehaviour
{
    private int counter=0;
    float[,] PT =new float[16,10];
    #region Singleton
    int level=0;
    public Scene m_Scene,m_SecScene;

    private static GameManager _instance;
    public static GameManager Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject gm = new GameObject("GameManager");
                gm.AddComponent<GameManager>();
            }
            return _instance;
        }
    }
    #endregion


    // [SerializeField]
    //  AudioSource levelSound;

    private void Awake()
    {
        _instance = this;
        Camera.main.transform.localRotation=Quaternion.Euler(0f,0f,0f);
    }
```

```
    private void Start()
    {
    }

    void Update (){

        m_Scene = SceneManager.GetActiveScene();
    }
    public void LevelCleared()
    {

            PT = BalloonManager.Instance.ptArray ();
        Debug.Log ("Fromx  Fromy  Fromz  Tox  Toy  Toz  x  y  z  MT  \r\n ");
            for (int m = 0; m < 16; m++) {

            Debug.Log (PT [m, 0] + "," + PT [m, 1] + ","+ PT [m, 2] + "," + PT [m, 3] +"," + PT [m, 4] +"," + PT [m, 5] +"," + PT [m, 6] + ","+ PT [m, 7] +","+ PT [m,
8] +","+ PT [m, 9] + "\r");


            }

             BalloonManager.Instance.iniArray ();
            Debug.Log ("you miss" +BalloonManager.Instance.counter+"   ");

        BalloonManager.Instance.j = 0;
            BalloonManager.Instance.counter = 0;
        Debug.Log ("level" + level);

        level++;
        if (level == 4) {
            Application.Quit ();
        } else {

            LevelManager.Instance.Level = LevelManager.Instance.Level + 1;
        }
    }
}
```

ObjectManager class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;

public class BalloonManager : MonoBehaviour
{

    private int currentSphereNum = 0;
    private int randomSphereNum=0;
    private float minDistanceZ = 3.5f;
    private float maxDistanceZ = 5.5f;
    float delta = 24* Mathf.PI / 180;
    float beta=180*Mathf.PI/180;
    float radian = 24* Mathf.PI / 180;
    float centerX;
    float centerY;
    float A =1f;
    Vector2[] positions;
    public float [,] pt=new float[16,10];
    public Material acMaterial;
    public Material inacMaterial;

    public int  j=0;
    public int counter=0;
    private float timeCounter = 0f;


    #region Singleton
    private static BalloonManager _instance;
    public static BalloonManager Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject bm = new GameObject("BalloonManager");
                bm.AddComponent<BalloonManager>();
            }
```

```
        return _instance;
    }
}
#endregion

private static int _balloonsInPlay=13;
public static int BalloonsInPlay
{
    get
    {
        return _balloonsInPlay;
    }
    set
    {
        _balloonsInPlay = value;
    }
}


private void Awake()
{
    _instance = this;
    this.positions=CaculatePosition ();
centerX =Camera.main.transform.position.x;
    centerY = Camera.main.transform.position.y;

    iniArray ();


}

public void iniArray(){

for (int m = 0; m < 16; m++) {

    for (int n = 0; n < 10; n++) {
        pt [m,n] = 0f;
    }

}
}
public void clicked(){


    Debug.Log ("ClickSphere");

    this.counter++;




}
public float[,] ptArray(){

    return pt;
}
public Vector2[] CaculatePosition(){

    Vector2[] positions={new Vector2(centerX + Mathf.Cos (radian) * (A / 2),centerY + Mathf.Sin (radian) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+beta) * (A / 2),centerY + Mathf.Sin (radian+beta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta) * (A / 2),centerY + Mathf.Sin (radian+delta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta+beta) * (A / 2),centerY + Mathf.Sin (radian+delta+beta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*2) * (A / 2),centerY + Mathf.Sin (radian+delta*2) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*2+beta) * (A / 2),centerY + Mathf.Sin (radian+delta*2+beta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*3) * (A / 2),centerY + Mathf.Sin (radian+delta*3) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*3+beta) * (A / 2),centerY + Mathf.Sin (radian+delta*3+beta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*4) * (A / 2),centerY + Mathf.Sin (radian+delta*4) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*4+beta) * (A / 2),centerY + Mathf.Sin (radian+delta*4+beta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*5) * (A / 2),centerY + Mathf.Sin (radian+delta*5) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*5+beta) * (A / 2),centerY + Mathf.Sin (radian+delta*5+beta) * (A / 2)),
        new Vector2(centerX + Mathf.Cos (radian+delta*6) * (A / 2),centerY + Mathf.Sin (radian+delta*6) * (A / 2)),
        new Vector2(centerX + Mathf.Cos( radian+delta*6+beta) * (A / 2),centerY + Mathf.Sin (radian+delta*6+beta) * (A / 2)),

        //    return Ballpositions;
    };
    return positions;
}

//Logic of what happens when a balloon is poped.
public void BalloonPoped()
```

```
    {

        j++;
        BalloonsInPlay--;
        if (BalloonsInPlay == 0)
        {
            BalloonsInPlay = 13;
                    GameManager.Instance.LevelCleared();

        }


    }
    public Vector3 SpawnBalloonsAccordingToLevel(float level)
    {
        Vector3 pseudoRandomPosition =new Vector3((float)(this.positions[13-_balloonsInPlay].x),(float)(this.positions[13-
_balloonsInPlay].y), (float)(level/3));
    //    Debug.Log (_balloonsInPlay+"_balloonsInPlayyyyyyyyyyyyyyyyyyyyyyyyyyyyy");
        if (_balloonsInPlay == 13) {
        } else {
                    pt [j, 0] = (float)(Mathf.Round (this.positions [13 - _balloonsInPlay - 1].x * 1000) / 1000);
                pt [j, 1] = (float)(Mathf.Round (this.positions [13 - _balloonsInPlay - 1].y * 1000) / 1000);
                pt [j, 2] = (float)(level/3);
                pt [j, 3] = (float)(Mathf.Round (this.positions [13 - _balloonsInPlay].x * 1000) / 1000);
                pt [j, 4] = (float)(Mathf.Round (this.positions [13 - _balloonsInPlay].y * 1000) / 1000);
                pt [j, 5] = (float)(level/3);
        }
        return pseudoRandomPosition;

    }

    public void AddHitPosition(Vector3 Hit){
        pt [j,6] = Hit.x;
        pt [j,7] = Hit.y;
        pt [j,8] = Hit.z;


    }

    public void AddTime(float time){
            pt [j,9] = time;

    }
}
```

Object class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Ball : MonoBehaviour
{


    private Vector3 startingPosition;
    private Renderer renderer;
    public Material inactiveMaterial;
    public Material gazedAtMaterial;
    private const float AUTO_DESTROY_TIME =2f;
    public float timeCounter = 0f;
    public bool playerHoveringOver;


    void Start() {


        renderer = GetComponent<Renderer>();
        renderer.material = inactiveMaterial ;

    }
    void Update(){

        timeCounter = timeCounter + Time.deltaTime;
        if (((Input.GetButton("Fire1"))&&playerHoveringOver==false) {

                this.Disappear ();

        }

    }
```

```
public void SetGazedAt() {
    if (inactiveMaterial != null && gazedAtMaterial != null) {
        renderer.material = gazedAtMaterial ;
        playerHoveringOver = true;
        return;
    }
}
public void SetGazedAtNot() {
    if (inactiveMaterial != null && gazedAtMaterial != null) {
        renderer.material = inactiveMaterial;
        playerHoveringOver = false;
        return;
    }
}

public void GetPoint(){
        BalloonManager.Instance.BalloonPoped ();
        this.renderer.material = inactiveMaterial;
        this.gameObject.SetActive (false);
        int sibIdx = transform.GetSiblingIndex ();
        int numSibs = transform.parent.childCount;
        sibIdx = (sibIdx + Random.Range (1, numSibs)) % numSibs;
        GameObject randomSib = transform.parent.GetChild (sibIdx).gameObject;
        randomSib.transform.position = BalloonManager.Instance.SpawnBalloonsAccordingToLevel (LevelManager.Instance.Level);
        randomSib.SetActive (true);
       BalloonManager.Instance.AddTime (timeCounter);


        timeCounter = 0;

    }

public void Disappear(){

      BalloonManager.Instance.BalloonPoped ();
      BalloonManager.Instance.clicked ();
      int sibIdx = transform.GetSiblingIndex ();
      int numSibs = transform.parent.childCount;
      sibIdx = (sibIdx + Random.Range (1, numSibs)) % numSibs;
      GameObject randomSib = transform.parent.GetChild (sibIdx).gameObject;

      randomSib.transform.position = BalloonManager.Instance.SpawnBalloonsAccordingToLevel (LevelManager.Instance.Level);
      randomSib.SetActive (true);
      gameObject.SetActive (false);
      timeCounter = 0;
    }

}
```

Class for Head Ray

```
using UnityEngine;
using System.Collections;
using System;

public class SimpleGazeCursor : MonoBehaviour {
    public Camera viewCamera;
    public GameObject cursorPrefab;
    public float maxCursorDistance = 30;
    int mask;
    public GameObject hitObject;
    public RaycastHit hit;

    private GameObject cursorInstance;

    #region Singleton
    private static SimpleGazeCursor _instance;
    public static SimpleGazeCursor Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject bm = new GameObject("SimpleGazeCursor");
                bm.AddComponent<SimpleGazeCursor>();
            }

            return _instance;
        }
    }
    #endregion
```

```
    // Use this for initialization
    void Start () {
        cursorInstance = Instantiate(cursorPrefab);
    //    mask = LayerMask.GetMask ("Treasure","TriSphereo","Ball");
//        Debug.Log ("mask is"+mask);
    }

    // Update is called once per frame
    void Update () {
        UpdateCursor();
    }

    /// <summary>
    /// Updates the cursor based on what the camera is pointed at.
    /// </summary>
    private void UpdateCursor()
    {
        // Create a gaze ray pointing forward from the camera
        Ray ray = new Ray(viewCamera.transform.position, viewCamera.transform.rotation * Vector3.forward);

        int layerMask = 1 << 8;
        if (Physics.Raycast (ray, out hit, Mathf.Infinity, layerMask)) {
            // If the ray hits something, set the position to the hit point and rotate based on the normal vector of the hit

            hit.collider.GetComponent<HandController> ().SetGazedAt ();
            if (hit.collider.tag == "Ball") {

                cursorInstance.transform.position = hit.point;
                cursorInstance.transform.rotation = Quaternion.FromToRotation (Vector3.up, hit.normal);
                hitObject = hit.transform.gameObject;

                hitObject.GetComponent<HandController> ().SetGazedAt ();
                hitObject.GetComponent<HandController> ().GetPoint (hit.point);
            }
        }else
        {
            Debug.Log(" not hitttttttttttttttttttt");
            // If the ray doesn't hit anything, set the position to the maxCursorDistance and rotate to point away from the camera
            cursorInstance.transform.position = ray.origin + ray.direction.normalized * maxCursorDistance;
            cursorInstance.transform.rotation = Quaternion.FromToRotation(Vector3.up, -ray.direction);

            if (hitObject != null) {
                Debug.Log("hit others");
                hitObject.GetComponent<HandController> ().SetGazedAtNot ();
                hitObject = null;
            }

        }
    }


    public RaycastHit HitPos(){

        return this.hit;
    }
}
```

Class for Air Touch

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class Nail : MonoBehaviour
{

     private bool isExecuting, userIsPointing;
    float detectedDepth, depthAdjustment;
    GameObject nailIllustration,point;
    Vector3 pointPosition;

    private void Start()
    {
        depthAdjustment = 2f;

        ManomotionManager.Instance.SetSelectedHand(SelectedHand.RIGHT_HAND);

            nailIllustration=transform.GetChild(0).gameObject;
    }
```

```
    void Update()
    {

        ControlMovementWithHand();
    }

    void ControlMovementWithHand()
    {
                if (ManomotionManager.Instance.Hand_infos[0].hand_info.gesture_info.mano_class == ManoClass.POINTER_GESTURE_FAMILY)
        {
            nailIllustration.SetActive(true);
            detectedDepth = Mathf.Clamp(ManomotionManager.Instance.Hand_infos [0].hand_info.tracking_info.relative_depth, 0f,1f);
            Vector3 pointerPosition = ManomotionManager.Instance.Hand_infos [0].hand_info.tracking_info.finger_tips [3];
            Vector3 newPos =  ManoUtils.Instance.CalculateNewPosition(pointerPosition, detectedDepth*depthAdjustment);

            this.transform.position = Vector3.Lerp(this.transform.position, newPos, Time.deltaTime * 10);
        }
        else
        {

            nailIllustration.SetActive(false);
                }
    }
}
```

References

[1] Argelaguet, F., & Andujar, C. (2013). A survey of 3D object selection techniques for virtual environments. Computers & Graphics, 37(3), 121-136.

[2] Balakrishnan, R. (2004). “Beating” Fitts’ law: virtual enhancements for pointing facilitation. International Journal of Human-Computer Studies, 61(6), 857-874. Baldauf, M. (n.d.). Markerless Visual Fingertip Detection for Natural Mobile Device Interaction, 539–544.

[3] Brown, M. A., & Stuerzlinger, W. (2016, July). Exploring the Throughput Potential of In-Air Pointing. In International Conference on Human-Computer Interaction Springer, Cham,13-24

[4] Bowman, D., Kruijff, E., LaViola Jr, J. J., & Poupyrev, I. P. (2004). 3D User interfaces: theory and practice, CourseSmart eTextbook. Addison-Wesley.

[5] Bowman, D. A., & Hodges, L. F. (1997, April). An evaluation of techniques for grabbing and manipulating remote objects in immersive virtual environments. In Proceedings of the 1997 symposium on Interactive 3D graphics (pp. 35-ff). ACM.

[6] Bowman, D., Wingrave, C., Campbell, J., & Ly, V. (2001). Using pinch gloves (tm) for both natural and abstract interaction techniques in virtual environments.

[7] Carnahan, H., & Marteniuk, R. G. (1991). The temporal organization of hand, eye, and head movements during reaching and pointing. Journal of Motor Behavior, 23(2), 109-119.

[8] Castellucci, S. J., Teather, R. J., & Pavlovych, A. (2013, July). Novel metrics for 3D remote pointing. In Proceedings of the 1st symposium on Spatial user interaction (pp.

17-20). ACM.

[9] Chan, L. W., Kao, H. S., Chen, M. Y., Lee, M. S., Hsu, J., & Hung, Y. P. (2010, April). Touching the void: direct-touch interaction for intangible displays. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 2625-2634). ACM.

[10] Elmqvist, N., & Tsigas, P. (2007, March). A taxonomy of 3D occlusion management techniques. In Virtual Reality Conference, 2007. VR'07. IEEE (pp. 51-58). IEEE.

[11] Erol, A., Bebis, G., Nicolescu, M., Boyle, R. D., & Twombly, X. (2007). Vision-based hand pose estimation: A review. Computer Vision and Image Understanding, 108(1-2), 52-73.

[12] Hernandez, B., & Flores, A. (2014, January). A bare-hand gesture interaction system for virtual environments. In 2014 International Conference on Computer Graphics Theory and Applications (GRAPP) (pp. 1-8). IEEE.

[13] Gugenheimer, J., Dobbelstein, D., Winkler, C., Haas, G., & Rukzio, E. (2016, October). Facetouch: Enabling touch interaction in display fixed uis for mobile virtual reality. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (pp. 49-60). ACM.

[14] Hand, C. (1997, December). A survey of 3D interaction techniques. In Computer graphics forum (Vol. 16, No. 5, pp. 269-281). Oxford, UK and Boston, USA: Blackwell Publishers.

[15] ISO. (2000). ISO 9241-9 Ergonomic requirements for office work with visual display terminals (VDTs) - Part 9: Requirements for nonkeyboard input devices:

International Standard. International Organization for Standardization.

[16] Ishii, A., Adachi, T., Shima, K., Nakamae, S., Shizuki, B., & Takahashi, S. (2017, May). FistPointer: Target Selection Technique Using Mid-air Interaction for Mobile VR Environment. In Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems (pp. 474-474). ACM.

[17] Jacoby, R. H., Ferneau, M., & Humphries, J. (1994, April). Gestural interaction in a virtual environment. In Stereoscopic Displays and Virtual Reality Systems (Vol. 2177, pp. 355-365). International Society for Optics and Photonics.

[18] Lin, C. J., Ho, S. H., & Chen, Y. J. (2015). An investigation of pointing postures in a 3D stereoscopic environment. Applied ergonomics, 48, 154-163.

[19] Lubos, P., Bruder, G., & Steinicke, F. (2014, March). Analysis of direct selection in head-mounted display environments. In 3D User Interfaces (3DUI), 2014 IEEE Symposium on (pp. 11-18). IEEE.

[20] Slater, M. (2003). A note on presence terminology. Presence connect, 3(3), 1-5.

[21] MacKenzie, I. S., & Isokoski, P. (2008, April). Fitts' throughput and the speed-accuracy tradeoff. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 1633-1636). ACM.

[22] Mine, M. R., Brooks Jr, F. P., & Sequin, C. H. (1997, August). Moving objects in space: exploiting proprioception in virtual-environment interaction. In Proceedings of the 24th annual conference on Computer graphics and interactive techniques (pp. 19-26). ACM Press/Addison-Wesley Publishing Co..

[23] Ni, T., Bowman, D. A., & Chen, J. (2006, June). Increased display size and resolution improve task performance in information-rich virtual environments. In

Proceedings of Graphics Interface 2006 (pp. 139-146). Canadian Information Processing Society.

[24] Pavlovych, A., & Stuerzlinger, W. (2011, May). Target following performance in the presence of latency, jitter, and signal dropouts. In Proceedings of Graphics Interface 2011 (pp. 33-40). Canadian Human-Computer Communications Society.

[25] Piumsomboon, T., Clark, A., Billinghurst, M., & Cockburn, A. (2013, September). User-defined gestures for augmented reality. In IFIP Conference on Human-Computer Interaction (pp. 282-299). Springer, Berlin, Heidelberg.

[26] Poupyrev, I., Ichikawa, T., Weghorst, S., & Billinghurst, M. (1998, August). Egocentric object manipulation in virtual environments: empirical evaluation of interaction techniques. In Computer graphics forum (Vol. 17, No. 3, pp. 41-52). Oxford, UK and Boston, USA: Blackwell Publishers Ltd.

[27] P Powell, W., Powell, V., Brown, P., Cook, M., & Uddin, J. (2016, March). Getting around in google cardboard–exploring navigation preferences with low-cost mobile VR. In Everyday Virtual Reality (WEVR), 2016 IEEE 2nd Workshop on (pp. 5-8). IEEE.

[28] Ramcharitar, A., & Teather, R. J. (2018). EZCursorVR: 2D Selection with Virtual Reality Head-Mounted Display. Proceedings of Graphics Interface 2018, 114-- 121.

[29] Teather, R. J., & Stuerzlinger, W. (2013, April). Pointing at 3d target projections with one-eyed and stereo cursors. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 159-168). ACM.

[30] Teather, R. J., & Stuerzlinger, W. (2014, October). Visual aids in 3D point selection experiments. In Proceedings of the 2nd ACM symposium on Spatial user interaction

(pp. 127-136). ACM.

[31] Teather, R. J., Stuerzlinger, W., & Pavlovych, A. (2014, April). Fishtank fitts: a desktop VR testbed for evaluating 3D pointing techniques. In CHI'14 Extended Abstracts on Human Factors in Computing Systems (pp. 519-522). ACM.

[32] Vuibert, V., Stuerzlinger, W., & Cooperstock, J. R. (2015, August). Evaluation of docking task performance using mid-air interaction techniques. In Proceedings of the 3rd ACM Symposium on Spatial User Interaction (pp. 44-52). ACM.

[33] Wigdor, D., Forlines, C., Baudisch, P., Barnwell, J., & Shen, C. (2007, October). Lucid touch: a see-through mobile device. In Proceedings of the 20th annual ACM symposium on User interface software and technology (pp. 269-278). ACM.

[34] Wobbrock, J. O., Myers, B. A., & Aung, H. H. (2008). The performance of hand postures in front-and back-of-device interaction for mobile computing. International Journal of Human-Computer Studies, 66(12), 857-875.

[35] Yoo, S., & Parker, C. (2015, August). Controller-less interaction methods for Google cardboard. In Proceedings of the 3rd ACM Symposium on Spatial User Interaction (pp. 127-127). ACM.

[36] Zayer, M. A., Tregillus, S., & Folmer, E. (2016, October). PAWdio: Hand Input for Mobile VR using Acoustic Sensing. In Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play (pp. 154-158). ACM.

[37] Zeleznik, R. C., Forsberg, A. S., & Schulze, J. P. (2005). Look-that-there: Exploiting gaze in virtual reality interactions. Technical report, Technical Report CS-05.

[38] Zhai, S., Milgram, P., & Buxton, W. (1996, April). The influence of muscle groups on performance of multiple degree-of-freedom input. In Proceedings of the SIGCHI

conference on Human factors in computing systems (pp. 308-315). ACM.