Using Image-Based Tracking for Smartphone-Based Interaction in Virtual Reality

by

Honours Project Thesis

Supervisor: Dr. Oscar Meruvia-Pastor

Department of Computer Science

Memorial University of Newfoundland

April 2022

St. John's Newfoundland

Abstract

Some of the barriers preventing Virtual Reality (VR) from being widely adopted are the price and unfamiliarity of VR systems. Our work proposes that the specialized controllers used for VR can be replaced by a regular smartphone, cutting the cost of the system and allowing users to make use of a device they already own and are familiar with. Our solution for using a smartphone as a VR controller involves mounting a camera on top of the head-mounted display (HMD) that is tilted downwards to where the user will be holding their phone in front of them. Then, we use image-tracking to track QR code displayed on the phone screen in order to track the phone's position inside the field of view of the camera.

Our technique allows the user to control a pointer that is visible in the HMD by moving their smartphone and to interact with objects within the virtual environment using both touch screen and tilt controls on the smartphone. We implemented this method along with example applications to test the usability of our technique. These applications include a navigation demo where the user can navigate using teleport points as well as a key-and-lock mechanism application. By testing our implementation using these applications, we found that a smartphone can be used as a controller in this way to conveniently perform common tasks in VR such as navigation and selecting, positioning, and rotating 3D objects.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Figure 1.1: The setup for our technique with the camera mounted on top of the HTC Vive Cosmos Elite HMD.

Virtual reality (VR) refers to a type of immersive graphics systems where the user can navigate around computer-generated virtual environments and interact with 3D objects inside the environment. VR systems are typically sold with a set of specialized controllers that the user can hold in each hand that allow the user to perform actions

in the environment by pointing the controllers and pressing buttons on them. Some systems also require other hardware such as base stations which help to track the location and 3D orientation of the controllers and headset.

One of the barriers preventing VR being widely adopted is the price of VR systems. Our work proposes that the specialized controllers used for VR can be replaced with regular smartphones by allowing the user to use a smartphone for navigation and interaction in VR. Since the majority of people own a smartphone, this would allow users to use a device they already own instead of having to purchase separate controllers, which cuts down on the total cost of the system. The reduction in cost from this would make VR more accessible to the average consumer, as well as allow them to use a device they are already familiar with. Moreover, the smartphones contain sensors and displays which can also be used to improve the interaction capabilities of regular VR systems.

Our solution for using a smartphone as a controller for VR involves mounting a camera on top of the head-mounted display (HMD), which tracks an image displayed on the phone screen. Then, the position of the phone within the camera view controls a pointer that is visible in the HMD and used for interaction within VR. We used the HTC Vive Cosmos Elite with a D435i Intel Realsense depth-sensing camera for our implementation [22, 7]. However, our solution would work with most other current HMDs as well since it does not depend on any specific features of this VR system or the camera.

In order to test the feasibility of this method of using a smartphone as a controller, we also implemented several example applications. The first of these applications is a demo where the user can navigate around the VR environment by selecting footprint-shaped teleport points. These teleport points are visible on the ground and the user can select one by lining the pointer up with it and tapping the phone screen. When the user selects a teleport point, they will be teleported to the point's location and their rotation will be updated so that they are facing the direction the footprint is facing. This example application demonstrates that our method of using a smartphone as a controller can be used to conveniently navigate in VR.

Another one of these applications uses a key-and-lock mechanism for two-factor authentication based on the video game Tetris [18]. In this demo, the user sees a sample Tetris scene in the HMD where there are blocks laid out in such a way that there are one or more empty spaces where additional Tetris blocks can fit in between the existing blocks. The user must select the correct blocks and adjust their position and orientation so that they fit inside the provided empty spaces. Once the user fills each of the empty spaces with the correct blocks, the task is complete.

To do this using our implementation, the user can line the pointer up with the blocks by moving the phone and then tap and tilt the smartphone to perform several types of interactions. These interactions include selecting a block to drag and drop in a new position by holding their thumb against the screen and moving the smartphone, as well as rotating a block by tilting the phone to the left or right. This application

demonstrates that our technique can be used to conveniently perform many common tasks in VR, such as selecting, positioning, and rotating 3D objects.

Chapter 2

Related Work

A few different approaches of using a smartphone or other alternate devices for interaction in VR and AR have been tried previously [2, 3, 20, 12, 4, 23, 5, 26, 8, 9, 13, 6]. Furthermore, smart devices have been used for cross-device interactions in other ways as well [14, 15, 17]. These alternate approaches mainly focus on using the built-in sensors in the smart devices to control a ray caster or pointer that can be used for interaction, with or without some type of external tracking. Table 2.1 presents an overview of this related work, sorted based on relevance to our technique.

Some early work that explores this idea proposes connecting a smartphone to a VR/AR program and allowing the user to interact with 3D objects using the phone's touch screen [2, 3]. More recently, Unlu and Xio [20] built on this idea by using a combination of touch screen input along with external and internal tracking of the smartphone to allow the user to use it as a controller for several types of AR

applications. Their approach uses markers around the edges of a phone screen along with the phone's and HMD's sensors to help track the smartphone's position. One advantage of our approach over this one is that their approach requires a VR system with inside-out tracking, such as the Microsoft Hololens, while ours does not, so this makes our approach more accessible since it will work for cheaper headsets [11].

Similarly, TrackCap proposed by Mohr et al., involves mounting a cap on top of a HMD and using a smartphone camera to track an image printed on the underside of the cap [12]. This approach allowed them to track the position of the phone relative to the HMD and use it as an AR controller. One drawback to this approach compared to ours is that strong backlighting from the ceiling can cause poor visibility of the image on the cap, and users may have to compensate by using an additional light source underneath the cap. Since in our approach the image target is displayed on the phone screen, it does not suffer from this issue since the brightness of the phone screen can be adjusted to work well in different lighting conditions.

Another alternate approach for using a smartphone as a controller for VR was proposed by Hattori and Hirai where they tested using an image-detection method based on "plane finding" [4]. This allows the phone to estimate its own position based on ambient information. However, unlike our solution which also uses image-detection, their approach requires a specific synchronization process and their raycasting mechanism can go out of alignment due to poor synchronization.

Additionally, the Vive Flow is a commercial VR headset that is controlled using a

compatible Android smartphone [23]. It uses the sensors in the phone to implement motion controls and allows the user to tap sections of the screen to interact with the VR environment. Since the phone-tracking is based on the IMU sensors in the phone rather than image-tracking, it can be prone to the well-known issue of drift between the pointer controlled by the smartphone and the smartphone's actual position [5]. Also, using the different areas of the smartphone for different types of interactions can be difficult for some users as it is easy to accidentally touch the wrong part of the screen.

Furthermore, some other work explores using devices that are not necessarily smart devices but that contain IMU sensors [5, 26]. This includes the work by Young et al. where an arm-mounted input device was tested and the work by Hincapié-Ramos et al. which proposed using a smartphone-like device as a controller. Similar to all other comparable approaches which use sensors for tracking, these methods are also prone to some degree of error caused by drift.

Some other related work involves using other smart devices such as smartwatches for interaction in VR [8, 9, 13, 6]. One example is TickTockRay proposed by Kharlamov et al. [8] which uses the built-in sensors in a smartwatch to track the user's arm movements while in mobile VR and uses this movement to control a raycast. Conversely, the solutions proposed by Park and Lee [13] as well as Kim and Woo [9] use hand tracking to allow users to interact with the virtual environment by performing hand gestures in front of depth sensors. One issue with using a smartwatch is fatigue

caused by the user having to keep their arms raised while performing repetitive hand and wrist motions. Also, smartwatches are less popular and less likely to be familiar to users, when compared to smartphones.

Lastly, others have used smartphones and smartwatches for pointing and interacting with large 3D displays [14, 15, 17]. These solutions similarly use the built-in gyroscope and accelerometers in smartphones and smartwatches to control a ray caster that can be used to interact with 3D objects by touching a touch screen or performing some other interaction mechanism.

Key	Relevance	Year	Reference
Unlu2021	Highly related	2021	[20]
Mohr2019	Highly related	2019	[12]
Park2019	Highly related	2019	[13]
Siddhpuria2018	Highly related	2018	[17]
Kharlamov2016	Highly related	2016	[8]
Pietroszek2014	Highly related	2014	[14]
Hattori2020	Moderately related	2020	[4]
Hirzle2018	Moderately related	2018	[6]
Pietroszek2017	Moderately related	2017	[15]
Kim2016	Moderately related	2016	[9]
Aseeri2013	Moderately related	2013	[2]
Budhiraja2013	Moderately related	2013	[3]
Malekmakan2020	Somewhat related	2020	[10]
Weise2020	Somewhat related	2020	[25]
Alaee2018	Somewhat related	2018	[1]
Young2017	Somewhat related	2017	[26]
Hincapié-Ramos2015	Somewhat related	2015	[5]

Table 2.1: An overview of research on related work.

Chapter 3

Methodology

3.1 Raycasting Mechanism

Our solution provides reliable tracking of a smartphone in order to be able to use it as a controller for VR applications. Unlike other similar solutions, we do not rely on using the phone's internal sensors such as the gyroscope and accelerometer to track the phone's position and orientation. Instead, we track an image of a QR code displayed on the screen of the smartphone using Vuforia Engine, which is a software development kit (SDK) that is typically used for creating augmented reality applications [24]. Vuforia provides the functionality to be able to easily track images and display virtual objects on top of them, and we make use of this feature in our solution that we created using the Unity game engine [19]. Furthermore, we used the HTC Vive Cosmos Elite VR system to test our technique. However, this solution

is independent of the HMD since it does not rely on any features specific to this VR system and would likely work with most modern HMDs. For the camera, we used the D435i Intel Realsense depth-sensing camera, but this is also not required as any regular camera with a high enough resolution for it to be used for reliable image-tracking will work.

To facilitate the image-tracking using Vuforia, the camera is mounted on top of the HMD and angled downwards at an approximately 45-degree angle so that it points at where the user will naturally be holding their phone in their hand in front of them (Figure 1.1). As we track the phone's position by tracking the QR code displayed on its screen, we can track where the phone screen is located relative to the camera view of the camera on top of the HMD. We use this relative position of the phone to create a small red square that acts as a pointer that the user can see inside the HMD and can control by moving the smartphone (Figure 3.2).

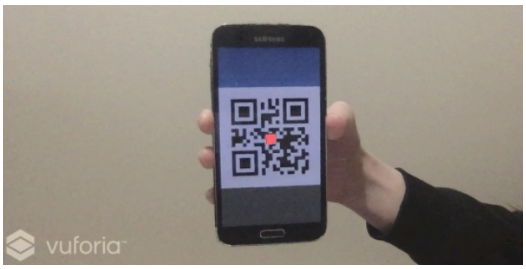

Figure 3.1: Using Vuforia Engine to display a red cube on top of the tracked QR code image displayed on the smartphone.

As shown in Figure 3.1, when the phone screen with the QR code displayed on it is in the camera view, Vuforia creates a red cube on top of the center of the QR code and we can get the position of this object within the camera view. Additionally, in Unity, there is a transparent canvas object that follows the movement of the HMD

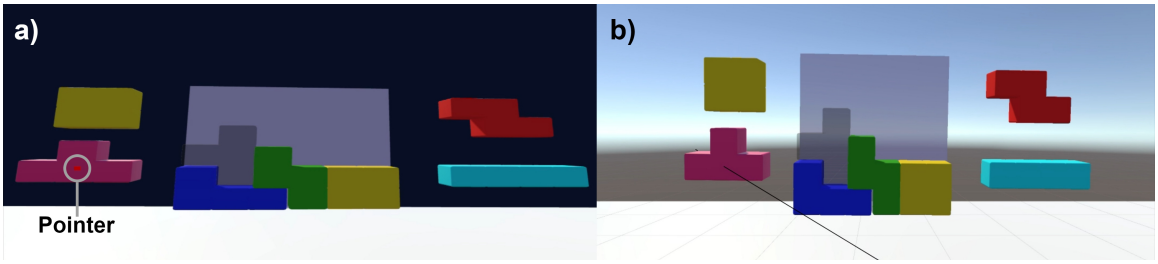

Figure 3.2: Images demonstrating how the raycasting mechanism works. a) The Game view that the player sees while wearing the HMD. Shows the red square pointer highlighting the Tetris block in the bottom left for selection. b) The Scene/Debug view that demonstrates how a black raycast is created starting from the player, towards the pointer, and through the Tetris block.

so that it fills the user's field of view inside the HMD at all times. The position of the cube created by Vuforia in screen coordinates is converted to the equivalent local coordinates within the canvas. These coordinates are then used to display an image of a small red square on the transparent canvas which the user can then see in the HMD and control easily by moving the phone (Figure 3.2).

Lastly, this red square is then used as a pointer for interaction with 3D objects within the virtual environment. This is achieved by creating a raycast from the user towards this pointer image on the canvas. While there are many methods of performing interactions in VR that are suitable for different situations, we decided to use a method based on raycasting since it is a popular method that is proven to be quick and convenient to use in many situations [25, 10]. The raycast extends indefinitely to hit objects in the background of the Unity scene that are any distance from the camera (Figure 3.2). This raycast is invisible to the user and simply allows us to detect which 3D object is in line with the pointer at any given time so that the

user can interact with it.

3.2 Network Communication from Smartphone to VR Application

For our application, there is also a wireless transfer of data between the smartphone and the VR application running on a desktop computer. To implement this networking functionality, we used an open-source networking solution for Unity called Riptide [16]. When the application starts, a server starts running on the desktop computer. Then, when the user opens the client application on their smartphone and taps the screen the first time, the smartphone connects to the server.

The network connection allows the smartphone to communicate with the VR application and permits the user to perform various actions in VR by interacting with the phone screen. In particular, we can send a message to the server when the user presses their finger against the phone screen and another when they release their finger from the phone screen. To be able to detect when the user touches and releases the smartphone screen, in the client application we simply cover the screen with a button displaying the tracked QR code. Then, we can send a message when the button is first pressed down and another when the button is released.

Finally, we can use this functionality to implement various types of clicking and hold-and-release interactions that are commonly used in VR applications. For in-

stance, the user can select a 3D object in a VR application by lining the pointer up with the object and then tapping the phone screen, and this will work by sending a message to the server application that will tell the application that the object has been selected.

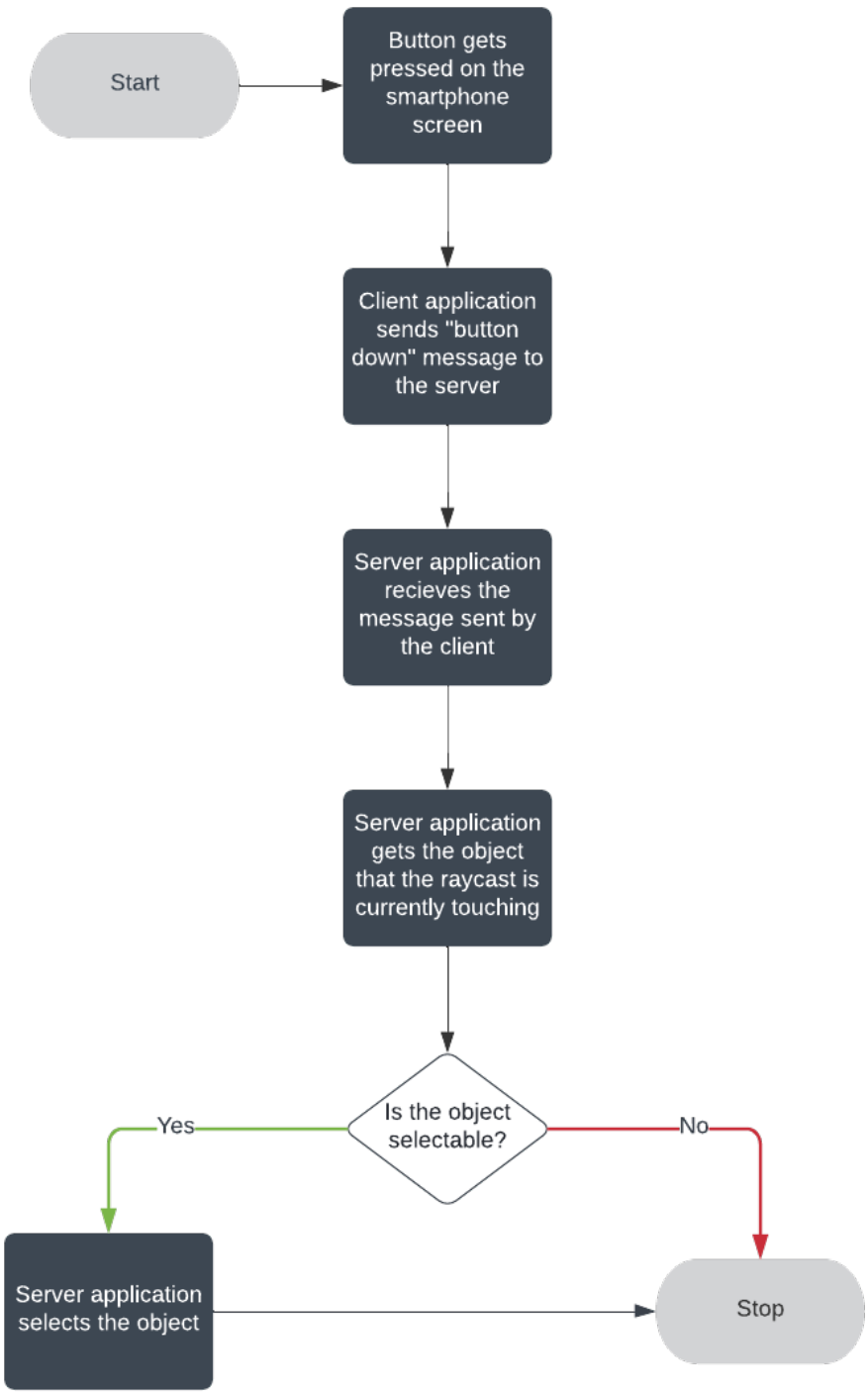

Figure 3.3: A flowchart showing the main workflow of the networking aspect of the application.

Chapter 4

Applications

4.1 Navigation using Teleport Points

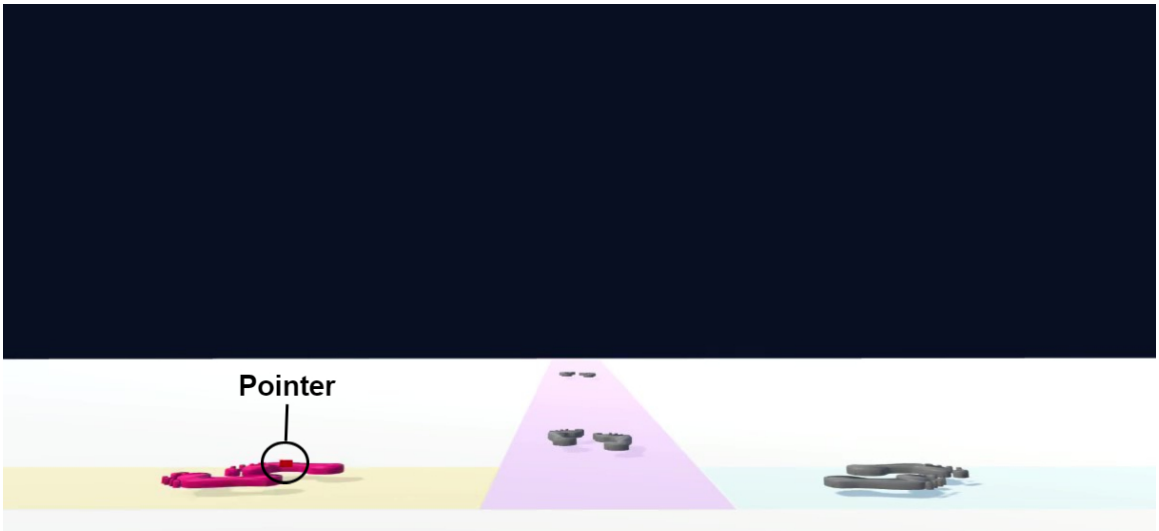

Figure 4.1: An application enabled by our technique where the user can select footprint-shaped teleport points to navigate around the VR environment.

The first of the two applications we created to verify our solution involves allowing the player to navigate around a VR environment using teleport points. In this application, the user can see three differently coloured paths going in different directions (Figure 4.1). On various points along the paths, there are gray 3D footprints just above ground level that represent the teleport points that the user can teleport to. To navigate around the scene, the user must select the closest teleport point to them that is in the direction that they want to go in.

To select a teleport point, the user can use the raycasting mechanism described in the previous chapter. When a user lines up the pointer with one of the footprints, it gets highlighted in bright pink to show that it can be selected (Figure 4.1). While the pointer is in line with the footprint, the user can tap the smartphone screen to select it. The user tapping the screen causes a message to be sent from the client application running on the smartphone to the VR application running on the desktop. Then, when the VR application receives this message, it is notified that the highlighted teleport point has been selected and proceeds to teleport the player to it.

When the player gets teleported, the player instantly moves to the location of the footprint. The x-coordinate and z-coordinate the player moves to are the world coordinates of the footprint object subtract some offset calculated using the world coordinates of the player and the world coordinates of the VR camera, as shown in Algorithm 1. The y-coordinate of the player's position remains constant so that their height within the VR environment is unchanged. Additionally, the user's rotation will

be updated to match the direction that the selected footprint is facing. To achieve this, we rotate the player about the y-axis around the point representing the user's head position in world coordinates by the number of degrees that are necessary to match the footprint's rotation. The player can then select another teleport point to continue moving around the environment.

```
1 Offset = (VRCameraPosition.x - PlayerPosition.x, 0, VRCameraPosition.z -
    PlayerPosition.z);
2 NewPlayerPosition = (FootprintPosition.x - Offset.x, PlayerPosition.y,
    FootprintPosition.z - Offset.z);
```

Algorithm 1: Player position calculation when teleporting.

We decided to develop this application as a test of our solution because navigation is an extremely important task in VR as it is required for many VR applications. There are several different methods that are often used to navigate in VR environments, but using teleportation is one of the most popular since it is beneficial in preventing motion sickness and other potential negative effects from VR. Therefore, it is significant to have shown that our solution can be successfully used for this navigation technique.

4.2 Tetris Key-and-Lock Mechanism

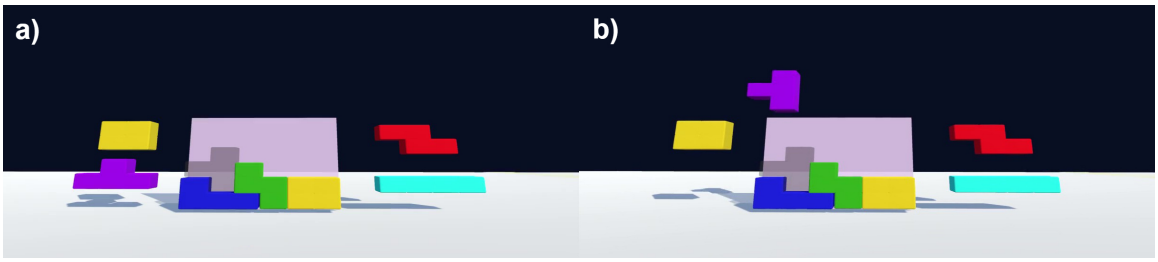

Figure 4.2: Images demonstrating the Tetris key-and-lock mechanism application. a) The initial setup of the application. b) The application while the player is moving the rotating the purple block.

Another application we implemented to test our solution was a key-and-lock mechanism based on the video game Tetris. In this demo application, the user is presented with a Tetris puzzle with one or more empty goal spaces represented by semi-transparent gray blocks where another block can fit (Figure 4.2). To complete the task, the user must select the correct block, rotate it to the correct orientation, and then drag it to the correct position in the puzzle.

To implement this, we placed a variety of different Tetris blocks at different orientations around the puzzle space that the user can select. When the pointer is lined up with one of these blocks, it gets highlighted in a bright pink color to indicate to the user that it is selectable (Figure 3.2). Then, while the block is highlighted, the user can press and hold their finger against the phone screen to select and pick up that block. While the user is holding their finger down, the block will move around with the movement of the smartphone, so the user can drag it to the correct position

in the puzzle (Figure 4.2). This functionality was implemented by getting a world point along the raycast at a z-distance close to the block and changing the x and y-values of the block's world points to match the position of the raycast. This makes it so that the block stays at a constant distance in the z-direction from the user, but it can be moved vertically and horizontally on the screen.

While the user has a block selected and is positioning it, they can also rotate the block. This is done by tilting the phone approximately 40 degrees in either direction. If the user tilts their phone to the right, it will rotate the block 90 degrees clockwise. Similarly, if they tilt their phone to the left, it will rotate the block 90 degrees counter-clockwise.

We detect if a user has rotated their phone by checking the difference in pixels between the y-coordinate of the screen point of each of the two top corners of the tracked image target within the camera view. If the difference between the corners of the image displayed on the phone is greater than a certain threshold, a rotation is triggered. Additionally, only one 90-degree rotation is triggered for each time the phone is tilted. Therefore, if the user wants to rotate the block more than once, they must go back to holding the phone relatively straight and then tilt it again. This helps to ensure that the rotation is purposeful as to avoid triggering unwanted rotations.

Finally, when the user stops holding their finger against the phone screen, the block gets released so it no longer moves with the pointer. When the user releases the block, if it is the correct block, at the correct orientation, and touching the goal

position, then the position of the released block snaps perfectly into the goal position. Otherwise, the position and rotation of the block reset to their original values so that the block goes back to being next to the puzzle space at its original orientation. We achieve this functionality by storing the original positions and orientations of all the selectable blocks as well as the goal block when the application first starts up so that these values can be easily reset.

This application is useful because it demonstrates the use of our solution for selecting, positioning, and rotating 3D objects in VR. This is a semi-complex use case that covers the actions which most VR tasks consist of. Therefore, if these actions can be easily achieved using our solution, then it should work for any similar tasks in VR using these types of actions.

Chapter 5

Limitations and Future Work

While our technique works quite well, there are a few limitations with our approach that are worth mentioning. The first is that if the QR code displayed on the phone screen cannot be tracked due to the phone screen not being visible in the field of view of the camera, there is an interruption in being able to use the phone to control the pointer. When this happens, the pointer remains in the last position it was in before the image-tracking was lost. Then, when the QR code becomes visible again and tracking resumes, the user is able to control the pointer once again. This is somewhat of a limitation since, ideally, there would be little to no interruption when tracking gets lost for a short period of time. This could be resolved by using the phone's internal IMU sensors as a back-up method of tracking the smartphone to fall back on when the QR code cannot be tracked, so that there would be no interruption.

Another limitation of the current implementation of this technique is that, unlike

with regular controllers, there is no way to see the phone while wearing the HMD. With many VR systems, typically there is a virtual representation of the controllers that the user can see while wearing the HMD. This is helpful since it allows them to see where the controls are on the controller so that it is easier to use. To address this limitation, we could create a similar virtual representation of the phone screen that is visible inside the HMD that allows the user to see what is displayed on the phone screen at any given time. Alternatively, we could combine our solution with an approach for Near Range Augmented Virtuality (NRAV) by Alaee et al. [1]. This would involve using a depth-sensing camera to allow the user to see their actual phone screen inside the HMD overlaid over the virtual environment when they bring the phone close enough to the camera mounted on the HMD.

Additionally, our implementation could be further refined by adding more features and developing more applications to use it with. For instance, adding haptic feedback by making the smartphone vibrate during interactions could help make the technique easier to use. Also, we could develop new applications to show how the smartphone can be used to create more unique interactions in VR using our technique. By using common touch screen gestures such as swiping, pinching, or tapping on the screen with multiple fingers, we could allow the user to interact with complex objects in VR in novel ways that would not necessarily be possible using a regular VR controller.

Chapter 6

Conclusion

In this paper, we have presented a technique that allows the user to use a regular smartphone for intuitive interaction in VR. This enables users of HMDs to control and interact with their virtual environment without the need for specialized controllers. The implication of this is that it makes VR more accessible to the average consumer due to a reduction in the cost required to make a fully functioning VR system. Additionally, using a smartphone rather than an unfamiliar controller reduces the learning curve for new VR users.

Through the demo applications we developed to test our solution, we have found that it allows for easy-to-use navigation within VR using teleport points. This is significant because navigation within VR is a highly important task. Additionally, we have shown that our technique can also be reliably used for the selection, positioning, and rotation of 3D objects within VR. This is promising since many interaction tasks

in VR will be made up of some combination of one or more of those tasks.

The code for this project is freely available for download [21]. At the end of this paper, there is an appendix detailing instructions for installing and running the project. This guide includes details on how to install both the client and server Unity projects as well as the Android smartphone application.

Bibliography

[1] Ghassem Alaee et al. “A User Study on Augmented Virtuality Using Depth Sensing Cameras for Near-Range Awareness in Immersive VR”. In: Institute of Electrical and Electronics Engineers Inc., Mar. 2018.

[2] Sahar A. Aseeri, Daniel Acevedo-Feliz, and Jurgen Schulze. “Poster: Virtual reality interaction using mobile devices”. In: https://doi.org/10.1109/3DUI.2013.6550211. 2013, pp. 127–128. ISBN: 9781467360975.

[3] Rahul Budhiraja, Gun A. Lee, and Mark Billinghurst. “Interaction techniques for HMD-HHD hybrid AR systems”. In: https://doi.org/10.1109/ISMAR.2013.6671786. 2013, pp. 243–244. ISBN: 9781479928699.

[4] Keisuke Hattori and Tatsunori Hirai. “Inside-out Tracking Controller for VR/AR HMD using Image Recognition with Smartphones”. In: https://doi.org/10.1145/3388770.3407430. Association for Computing Machinery, Aug. 2020. ISBN: 9781450379731.

[5] Juan David Hincapié-Ramos et al. “GyroWand: IMU-based Raycasting for Augmented Reality Head-Mounted Displays”. In: https://doi.org/10.

1145/2788940.2788947. Association for Computing Machinery, Inc, Aug. 2015, pp. 89–98. ISBN: 9781450337038.

[6] Teresa Hirzle et al. "WatchVR: Exploring the Usage of a Smartwatch for Interaction in Mobile Virtual Reality". In: https://doi.org/10.1145/3170427.3188629. Association for Computing Machinery, Apr. 2018. ISBN: 9781450356206.

[7] *Intel® RealSense™ Depth Camera D435i*. URL: https://www.intelrealsense.com/depth-camera-d435i/ (visited on 04/02/2022).

[8] Daniel Kharlamov et al. "TickTockRay: Smartwatch-based 3D pointing for smartphone-based virtual reality". In: https://doi.org/10.1145/2993369.2996311. Association for Computing Machinery, Nov. 2016, pp. 363–364. ISBN: 9781450344913.

[9] Hyung Il Kim and Woontack Woo. "Smartwatch-assisted robust 6-DOF hand tracker for object manipulation in HMD-based augmented reality". In: https://doi.org/10.1109/3DUI.2016.7460065. Institute of Electrical and Electronics Engineers Inc., Apr. 2016, pp. 251–252. ISBN: 9781509008421.

[10] Morteza Malekmakan, Wolfgang Stuerzlinger, and Bernhard Riecke. "Analyzing the Trade-off between Selection and Navigation in VR". In: https://doi.org/10.1145/3385956.3422123. Association for Computing Machinery, Nov. 2020. ISBN: 9781450376198.

[11] *Microsoft Hololens 2*. URL: https://www.microsoft.com/en-us/hololens (visited on 04/02/2022).

[12] Peter Mohr et al. "TrackCap: Enabling smartphones for 3D interaction on mobile head-mounted displays". In: https://doi.org/10.1145/3290605.3300815. Association for Computing Machinery, May 2019. ISBN: 9781450359702.

[13] Kyeong Beom Park and Jae Yeol Lee. "New design and comparative analysis of smartwatch metaphor-based hand gestures for 3D navigation in mobile virtual reality". In: *Multimedia Tools and Applications* 78 (5 Mar. 2019). https://doi.org/10.1007/s11042-018-6403-9, pp. 6211–6231. ISSN: 15737721.

[14] Krzysztof Pietroszek et al. "Smartcasting: A discount 3D interaction technique for public displays". In: https://doi.org/10.1145/2686612.2686629. Association for Computing Machinery, Inc, Dec. 2014, pp. 119–128. ISBN: 9781450306539.

[15] Krzysztof Pietroszek et al. "Watchcasting: Freehand 3D interaction with off-the-shelf smartwatch". In: https://doi.org/10.1109/3DUI.2017.7893335. Institute of Electrical and Electronics Engineers Inc., Apr. 2017, pp. 172–175. ISBN: 9781509067169.

[16] *RiptideNetworking*. URL: https://github.com/tom-weiland/RiptideNetworking (visited on 04/02/2022).

[17] Shaishav Siddhpuria et al. "Pointing at a Distance with Everyday Smart Devices". In: https://doi.org/10.1145/3173574.3173747. Association for Computing Machinery, Inc, Apr. 2018, pp. 1–11. ISBN: 9781450356206.

[18] *Tetris*. URL: https://tetris.com/ (visited on 04/02/2022).

[19] *Unity Real-Time Development Platform*. URL: https://unity.com/ (visited on 04/02/2022).

[20] Arda Ege Unlu and Robert Xiao. "PAIR: Phone as an Augmented Immersive Reality Controller". In: https://doi.org/10.1145/3489849.3489878. Association for Computing Machinery (ACM), Dec. 2021, pp. 1–6. ISBN: 9781450390927.

[21] *Using Image-Based Tracking for Smartphone-Based Interaction in Virtual Reality Project Download*. URL: https://drive.google.com/drive/folders/1KgzVXKlIRy6nN61HkxM8ihh7-UNvebQF?usp=sharing.

[22] *Vive Cosmos Elite Overview: Vive Canada*. URL: https://www.vive.com/ca/product/vive-cosmos-elite/overview/ (visited on 04/02/2022).

[23] *VIVE Flow*. URL: https://www.vive.com/ca/product/vive-flow/overview/ (visited on 04/02/2022).

[24] *Vuforia Developer Portal*. URL: https://developer.vuforia.com/ (visited on 04/02/2022).

[25] Matthias Weise, Raphael Zender, and Ulrike Lucke. “How Can I Grab That?” In: *i-com* 19 (2 Aug. 2020). https://doi.org/10.1515/icom-2020-0011, pp. 67–85. ISSN: 21966826.

[26] Thomas S. Young, Robert J. Teather, and I. Scott Mackenzie. “An arm-mounted inertial controller for 6DOF input: Design and evaluation”. In: https://doi.org/10.1109/3DUI.2017.7893314. Institute of Electrical and Electronics Engineers Inc., Apr. 2017, pp. 26–35. ISBN: 9781509067169.

Appendix

Instructions for Installing and Running the Project

How to Install and Set Up the Server Unity Project:

- Requirements:
  - Unity version 2020.3.25f.
  - SteamVR.
  - VIVE software, or the equivalent software for another VR headset if you are not using a VIVE system.
- Download the VRServer.zip file and extract its contents.
- Open the Unity Hub and click on the "Open" button.
- Select the folder where you extracted the project. The project should be added to the Unity Hub and open automatically in the Unity Editor.
- Select the Player game object from the Hierarchy on the left side of the screen (Figure 6.1).

- Use the Move Tool to drag the Player game object in front of the demo you want to try. The Rotate Tool can also be used to adjust the direction the Player is facing when the application starts.

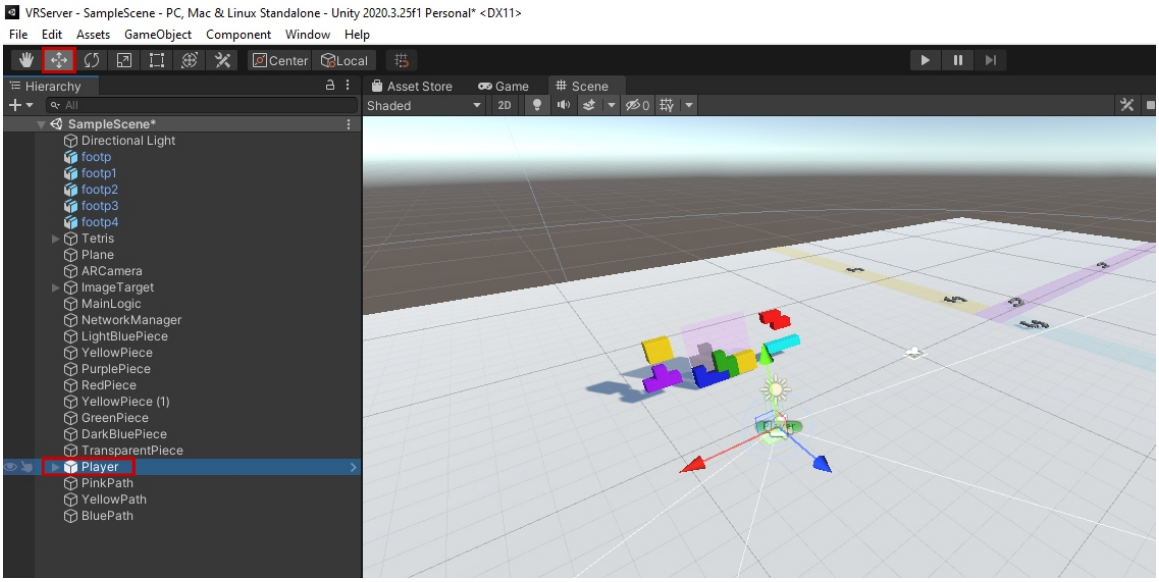

Figure 6.1: An image showing how to edit the Player's starting position in the Unity Editor.

How to Install and Set Up the Client Unity Project:

- Requirements:
  - Unity version 2020.3.25f.
  - An Android smartphone. This application was tested using a Samsung Galaxy S5 but will likely work with any newer Android smartphone.
- Download the VRClient.zip file and extract its contents.

- Open the Unity Hub and click on the "Open" button.
- Select the folder where you extracted the project. The project should be added to the Unity Hub and open automatically in the Unity Editor.
- Select the NetworkManager game object from the Hierarchy on the left side of the screen (Figure 6.2).
- In the Inspector on the right side of the screen, under the "Network Manager (Script)" section, edit the "Ip" field value to be the internal IP address of the host computer (Figure 6.2). The IP address can be found by running the "ipconfig" command using Command Prompt.
- Click on "File" and then "Build Settings..." in the top bar of the Unity Editor.
- Make sure Android is selected as the platform, then click the "Build" button and save the created .apk file.
- The .apk file created in the last step is the Android app. This file must be transferred to the Android smartphone and installed. One easy way to do this is by uploading the .apk file to Google Drive on your computer and then opening Google Drive on the smartphone and tapping on the file to download it.
- Open the VRClient app once it has finished installing.
- When the desktop VR application is running, tap the QR code on the smartphone screen to connect the smartphone to the server.

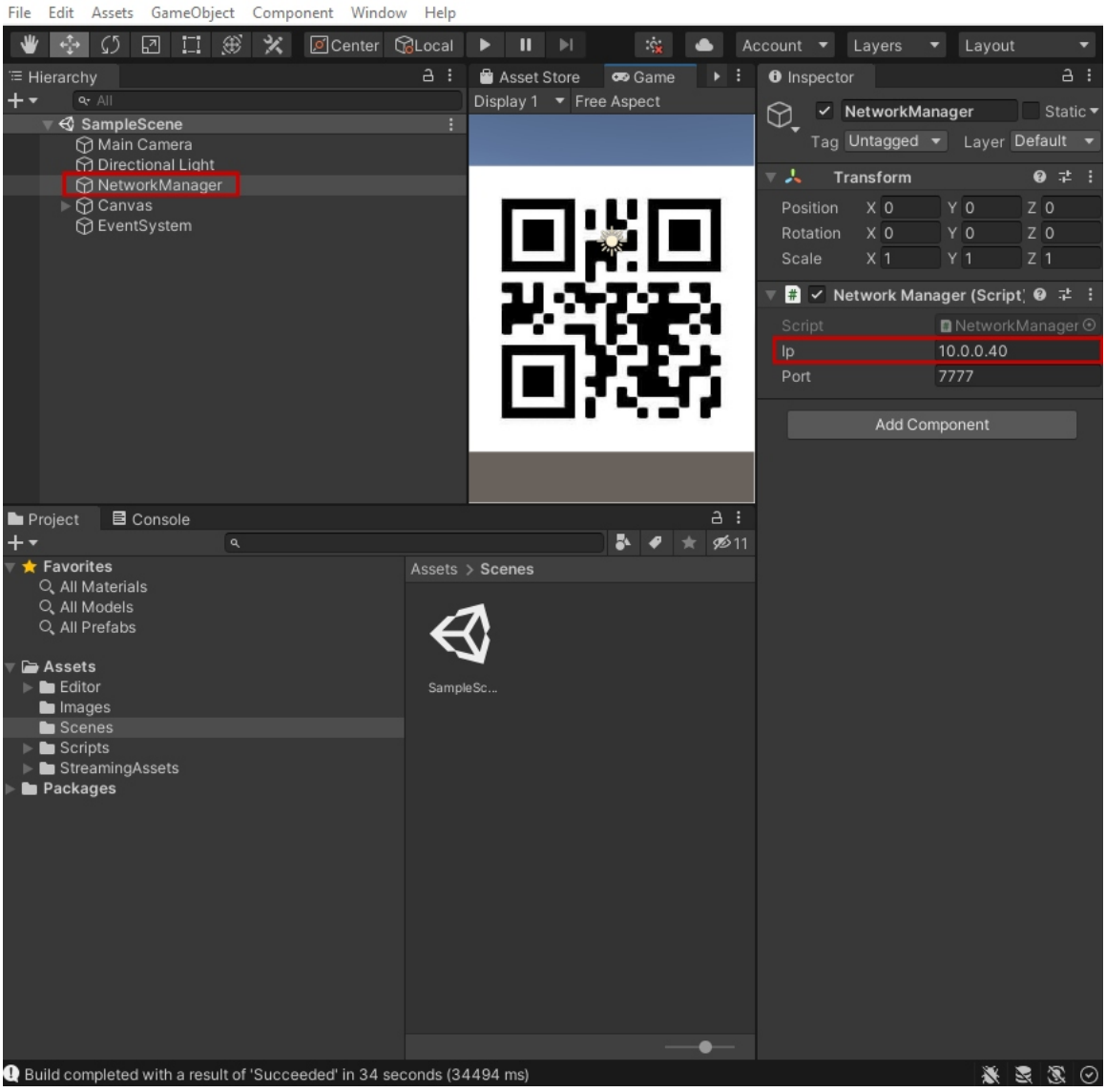

Figure 6.2: An image showing where to edit the IP address in the VRClient project.

Solutions to Potential Issues:

- Network Connection Between Smartphone and Computer Not Working:
  - Make sure the IP address entered for the NetworkManager game object in VRClient is correct.

  - Check your firewall settings to make sure it is not blocking IP address connections.

- Issues Receiving Video Stream from Camera:
  - Search for the "webcamprofiles.xml file" within the VRServer project folder and open it.
  - Add the information for the camera you are using to the "webcamprofiles.xml" file.
  - Open the VRServer project in the Unity Editor.
  - Select the ARCamera game object from the Hierarchy on the left side of the screen.
  - In the Inspector on the right side of the screen, under the "Vuforia Behaviour (Script)" section, click on the "Open Vuforia Engine configuration" button.
  - Under the "Play Mode" section of the Vuforia Configuration, select the camera you are using in the "Camera Device" dropdown menu.

- Pointer is Too High or Low:
  - Certain setups may require a manual adjustment to make the pointer easier to use.
  - Find the Canvas game object under Player → SteamVRObjects → VRCameraParent → VRCamera → Canvas and select it.
  - Move the Canvas object slightly up or down using the Move Tool until the pointer is working better.