Interacting with Environment-Driven Forces in Virtual Reality

By Anthony Kouroupis

Supervised by Dr. Christopher Collins and Dr. Mark Green

Ontario Tech University 2021

Abstract

As consumer-grade virtual reality is gaining popularity, one common theme among virtual reality games is sword fighting. In real-world sword fighting there are situations in which your sword is being pushed away by the opponent. Since the user's real hand can't be forced to move to match the result of this force, this causes the real-world hand position to desynchronize with that of their virtual hand.

In this thesis I analyze the cases which would cause this desynchronization in virtual reality. Using fencing as a basis, I will lay out some potential solutions which will help the user recognize and react to those situations, and compare them to existing solutions. Finally, I will describe the design space for a user study which implements our solutions.

1. Introduction

Using sword fighting as the motivation for a general problem in virtual reality interaction, I reached the question of this thesis: how should forces outside of the user's control which interfere with the user's actions be handled?

First I will describe the problem, and explore solutions that exist in current virtual reality games and in the literature. Afterwards I will propose some solutions to the problem in the form of visual, audible, and haptic assistants and discuss how they are meant to address the problem. After which I will discuss some tests that could be run in a user study to evaluate these solutions, along with a subjective analysis of each test and solution's effectiveness at meeting their prescribed goals. Finally, at the end of this thesis I will describe the implementation details of my solutions.

2. Motivation and Problem Statement

Development of sword fighting games in virtual reality began development shortly after the development of consumer grade virtual reality itself. The Kickstarter crowdfunding campaign for the sword fighting game Clang began on June 9th, 2012 [2]; only a month after the kickstarter for the Oculus Rift, which started its Kickstarter campaign on August 1st, 2012 [1].

When considering implementing competitive sword fighting in virtual reality, I identified a problematic interaction that would need to be handled to create an effective sword fighting game. When fencing, it is not an irregular case for one fencer to influence their opponent's blade in ways outside of their opponent's control. A simple example would be a parry. As shown in figure 1 below, it is an act in which a fencer uses their own blade to push their opponent's sword out of the way during an attack. This creates an awkward situation in virtual reality however, as when a user is parried in virtual reality the user's real hand cannot be forced to mimic the position that it should be in. This means that in this situation the user's in-game hand would be in a different position than that of the user's real-world hand. This disconnect may make it

difficult for a user to react as they intend to in such a situation, where precise movements would be required to break free of the parry.

Figure 1 an example of a fencing parry © Marie-Lan Nguyen / Wikimedia Commons / CC-BY 2.5

When considering handling a desynchronized state between the real and virtual hands, two approaches were considered: *recovery* and *avoidance*. Recovery is concerned with what is done in the case that the hand positions are desynchronized, whereas avoidance is concerned with avoiding desynchronization altogether. Avoidance is the most commonly practiced method in virtual reality sword fighting games. Specific ways current games practice avoidance will be examined in section 3.

For this paper, I chose to propose methods of recovery, because in real-world fencing it is impossible to guarantee that a fencer's sword will never be manipulated by their opponent, and as such, there is no way in our basis to have complete avoidance of desynchronization.

Our methods of recovery involve the use of visual, haptic, and audio cues that are intended for the user to react to and solve the situation themselves. These cues seek to satisfy four design goals: clarity, reactability, control, and presence.

Clarity means that the user should know that a desynchronization has occurred, and the cause of the desynchronization should be easy to discern.

Reactability means that the user should be able to react quickly to a desynchronization, as there may be cases in a game's design that will require reflexive action.

Control is concerned with how much control the user has over a situation where their virtual and real hands are desynchronized. The user should be able to take action in the situation that will help them regain control over their virtual hand when external forces are acting against it.

Finally, the solutions themselves should not break the user's feeling of immersion, or presence, in the virtual environment.

3. Related Work

This section details the places I searched when looking for existing solutions to the problem, including the solutions in a selection of existing games, and solutions in the current literature.

3.1. Solutions in the Literature

Looking at existing literature in this problem space, "A constraint-based god-object method for haptic display" [4] is the most popular one I found in the design space. This paper describes the use of a god object for users to interact with virtual environments. In our situation, our god object is the virtual representation of the player's hand. The paper continues to describe the case in which the god object is desynchronized by the user having placed their controller within the volume of a solid object, and the point on the surface of that object at which the god object should rest while the controller remains within that volume.

It is important that I distinguish the difference between this problem and the one I am addressing, which is the origin of the force that caused the desynchronization. This paper addresses the effect of forces driven by the user which cause the hand to move to a position that is impossible for it to be within the virtual environment. Since the user is the cause of this desynchronization in its entirety, they maintain full control of the situation.

The desynchronization which I am addressing originates from a source external to the user. This can be any force acting on the user or an object the user is holding that the user has no control over, such as gravity, or another player's actions. There is no guarantee that these forces will ever stop, or that the user will be able to stop them. These forces outside of the user's control can make their virtual hands, or the object in their hands act in ways the user does not intend regardless of their input.

The paper "Reaction times to constraint violation in haptics: Comparing vibration, visual and audio stimuli" [3], while not concerned with the origin of a desynchronization between the controller and virtual hand, still resides in the study space of desynchronization in virtual reality. This paper studies the maximum distance at which the virtual hand can be desynchronized from the user's real hand before the user notices. The findings of this paper can be useful in deciding the distances at which the solutions in this thesis should be activated.

3.2. Solutions in Existing Games

The first place I looked to find current methods of handling desynchronization was existing virtual reality games. I evaluated the methods these games used by watching gameplay footage, looking for cases in which the hand positions were desynchronized. The games I evaluated are Until You Fall, Swords of Gargantua, Ironlights, Blade & Sorcery, Gorn, Shadow Legend VR, Free Company VR, and Westworld Awakening.

Almost all of these games used methods of avoidance to handle desynchronization. Shadow Legend VR avoids desync by having the weapon pass through obstacles, as seen in this video [7]. The link for that video is timestamped to start at an instance of this event.

Blade & Sorcery on the other hand will allow the desync to happen and gives no notification to the user of the event. In the videos examined, the users did appear to recognize the desynchronization and restore it, but in some cases not without first continuing to try and swing their sword along the same path. In this example [8] at 1:47 (link includes timestamp), the user's arm stops when it collides with the enemy's body, and the user tries to swing their arm in the same way once more to follow up, getting stuck again in the same way.

Free Company VR takes the same approach as Blade & Sorcery in letting the sword get stuck on objects, as can be seen in this video [9a], but includes a method of minimizing this desynchronization; when the enemies block an attack from the player, or their attacks are blocked by the player, the enemies will be knocked backwards from the player, as seen in this video demoing the enemy's blocking mechanics [9b].

From watching gameplay footage for Until you Fall and Swords of Gargantua I was unable to get an adequate understanding of the games' mechanics for handling situations that would cause desynchronization, as such I cannot describe their methods.

The more unique solutions come from Ironlights and Gorn, which each have their own methods of avoidance.

In Gorn, the weapons are flexible. This means if they are forced into a position in which the hand would have to move to maintain the weapon's rigidity, the weapon bends instead, so that the player's hand is not forced to move.

In Ironlights, there is a degree to which desync is required, since fights in Ironlights occur in slow motion. As can be seen in this recording [10], the swords' movement lags behind the player's hand movements due to the slow motion interaction. In this case, the desynchronization is not caused by an external force such as the opposing player, but instead by the game's rules.

I still consider Ironlights' sword interaction a type avoidance regardless of this forced desync, as the path the sword will take will always follow the hand's path unimpeded, and there is no force will ever impede that movement. Additionally, when swords make contact with one another, or

when swords make contact with the opponent, they break, requiring the user to draw a new one by moving their hand behind their body. This aggressive yet effective method of avoidance ensures that there's no case in which desynchronization is caused by an external force, and that the virtual hand will always move along the path intended by the player, as the force difference between the weapons never has to be resolved. In the video demoing the gameplay [10], you can observe the swords breaking by their change in colour. The blades of the swords lose their luminescent property when broken.

Westworld Awakening, while not a sword fighting game, still has cases of desynchronization. In this game, the desynchronizations are user-driven, as described in section 3.1, and occur when taking action such as pushing your hand into space occupied by an object in the virtual environment, or pulling your hand far from a door while gripping the handle as seen in figure 2 below.

When the controller and virtual hand positions are out of alignment in Westworld Awakening, the game will show the user where the controller resides by rendering a transparent hand at the controller's position. This is an example of the solution proposed in section 4.3 being applied to user driven desynchronizations.

Figure 2: Ghost hand in Westworld Awakening during a user driven desynchronization

4. Solutions

As described in section 3.1, the forces I am concerned with act outside of the user's control, and there may not be a way to avoid desynchronization in an interaction. As such, the solutions I developed are focused on recovery, as opposed to avoidance.

For this problem I suggest visual, physical, and audible notifications for significant desynchronization. Significant desynchronization would be defined as an amount of desynchronization such that it becomes a hindrance. This measurement is subjective, but this paper [3] has defined some measurements for the distance at which a desynchronization begins to become noticeable to the user.

A solution activates when the virtual hand's position reaches a set distance away from the controller's position. This distance is measured as a radius from the controller's position. There is an additional trigger that can activate the solutions in the case of a difference in rotation between the controller's orientation and that of the virtual hand, measured in degrees. Each solution can be assigned a unique trigger radius or angle, and any combination of the solutions can be used.

Each solution described below may have variants that are meant to amplify or modify the effect of the solution. I will also discuss how each method relates to the goals defined in section 2 of this thesis in the *hypothesis* subsection of each solution.

4.1. Haptics

The haptic solution makes use of the virtual reality controllers' haptic feedback to provide physical feedback of a desynchronization.

4.1a Variants

Continuous: The controller vibrates continuously until the user restores the controller to a point within the radius. Depending on the strength of the haptic response, this could be a cause of annoyance. Annoying the user into restoring synchrony can be effective for restoring synchrony, but will likely lower the feeling presence in the interaction.

Impulse: A haptic pulse is triggered when the controller leaves the sync area, with no notification of re-entering the sync area, to avoid confusion (i.e., *is this pulse because I lost synchronization, or regained it?*). Since there is no notification of resynchronization this method should be paired with another solution which will communicate when synchronization is restored.

4.1b. Hypothesis

The use of haptics on desynchronization is meant to address the goal of reactability, as between visible, audible, and vibrational feedback, vibration had the fastest response time [5]. Furthermore, studies have shown that users that have experience with haptic feedback devices

showed a faster response to haptic feedback than those without [5]. This leads me to believe that a learning effect can be achieved over time to associate haptic feedback with a desynchronization event. The building of this association would be imperative on the game's design however, as applying the vibration effect to any other interaction in the game will dilute the association.

4.2. Line

When a sufficient desynchronization is reached, a line is drawn between the controller position and the virtual hand's position. The line is rendered along a cylindrical bound with each end connected to either the controller or the virtual hand. The line is rendered as a dashed line, and is animated such that the segments move from the virtual hand's position towards the controller's position, indicating the direction which the virtual hand is being pulled towards by the user's input. In the dashed line, each segment is of equal length, and the space between each segment is equal to the length of the segments.

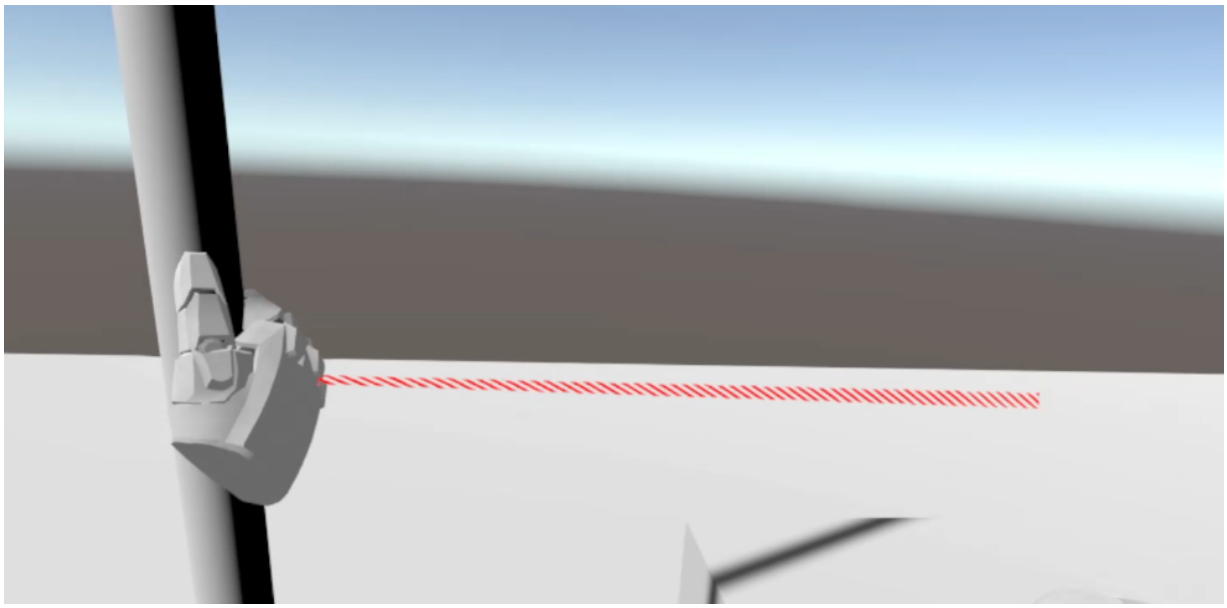


Figure 3: The line solution

4.2a. Variants

The line solution has a few variables which can be altered, though the effect of altering them may be minor, I feel it is still important to take note of them.

1. Segment length: The length of each segment
2. Diameter: The diameter of the line's cylindrical bound
3. Animation direction: The line can animate from the virtual hand to the controller, or the other way around
4. Animation speed: The speed at which the dashed line animates

4.2b. Hypothesis

This solution aims to increase clarity in the case of a desynchronization. That is, it is to notify the user of the distance and direction from the virtual hand to the controller.

In past work, visual indications have been observed to have the slowest reaction times between the haptic, sound, and visual responses [5]. This solution is meant to help the user identify situations in which the origin or direction of a desynchronization may be unclear. This can include cases such as when there is an object obstructing the user's view of the virtual hand during the desynchronization, or where the desynchronization is very large.

4.3. Ghost Hand

The ghost hand solution will render a transparent copy of the hand at the location the controller is actually in. The ghost hand will mimic the pose of the virtual hand (for example, the gripping position), but the rotation and position will be in sync with that of the controller at all times. As such, the ghost hand is capable of passing unimpeded through objects if the controller position passes inside their bounds.

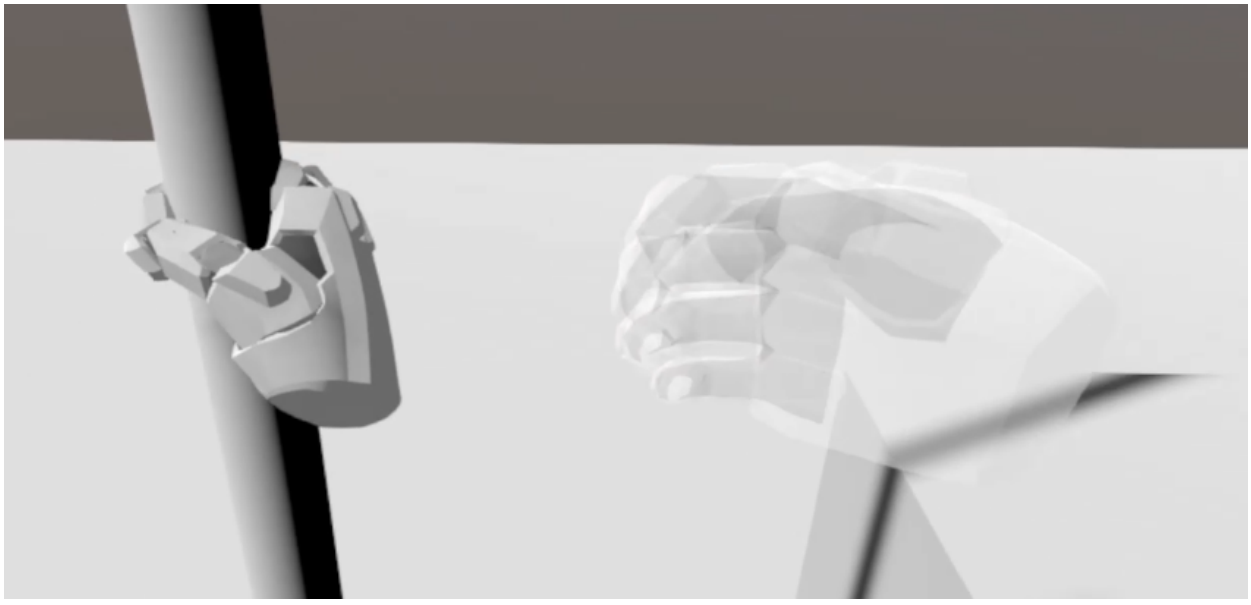


Figure 4: The ghost hand solution

4.3a. Variants

A variant of the ghost hand would be to have the ghost hand be holding a ghost copy of the object being held as well. The ghost object would also be transparent and would be able to pass through objects in the environment as the ghost hand is able to.

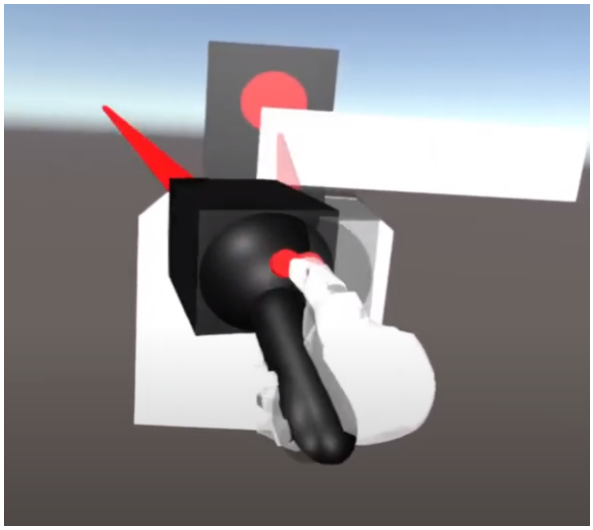


Figure 5: The ghost object variant of the ghost hand solution, sampled in the long stick test case.

4.3b. Hypothesis

The Ghost Hand solution serves two purposes:

First, we want to show the user where their hand actually resides in the virtual environment in the case of a desynchronization. This aims to improve clarity during a desynchronization

The second objective of the ghost hand solution is to give a feeling of presence in the virtual environment. By disconnecting the desynchronized hand from the user's actual hand position, it is my hope that users won't feel that the desynchronization is a flaw in the game's design, and instead feel that what is happening is a part of the game.

The goal of the ghost object variant is that of clarity, control, and presence. For clarity, the user would be able to see the ghost object pass through whatever object is causing the desync, and hopefully would allow them to quickly identify the cause of the desynchronization. Control of the situation may also be improved: by visualizing the object that needs to be moved the user doesn't have to make an assumption about the object's dimensions, for example whether or not the object is too large to fit through a space.

4.4. Sound

A solution involving sound is very simple, that is, play a sound to the user when synchrony is lost and regained. This solution would exist to replace the Haptic solution in the case that haptics are being used in multiple situations in the game, since the quick reaction time of haptics would rely on a training effect causing the user to associate the vibration with desynchronization. Using audio as a cue could be an accessibility issue for those who are hearing impaired, and is not implemented or evaluated in this thesis.

4.4a. Variants

There are infinite variations of this case, since the sound that plays can be changed. Additionally, the volume of the sound can be changed.

4.4b. Hypothesis

A sound cue's goal would be to improve reactability. Sound has been observed to have the second quickest response time between physical, visual, and audible notifications, specifically when it's loud. However sound notifications also have the lowest satisfaction rating among those methods [5].

5. Design Space

In this section I describe a breakdown of the external forces that cause desynchronization, and describe a set of tests with which the effectiveness of my proposed solutions can be measured.

5.1 Problem Breakdown

Since this thesis was designed with fencing in mind, I started defining the problem by examining the types of interaction between fencers (specifically their swords) which would cause a fencer's sword to move in a way that they did not intend, thus moving the hand as well. From this I was able to break down the problem into three different types:

First is persistent forces. This can be defined as the force of one object constantly pushing against another. An example of this outside of fencing would be a football player tackling a dummy, as they continuously press against the dummy, the dummy is experiencing a persistent force from the football player. In fencing, an example of this would be a parry, or using your sword to push the opponent's blade out of the way during an attack.

Next is impulse forces. Impulse forces are the momentum remaining from a momentary contact between two objects, like how a punching bag may continue to swing after being struck.

Finally there is chaotic force. Chaotic force is not a type of force in itself, but is instead the mixture of persistent and impulse forces in no particular pattern. This classification of force is important to acknowledge because it represents the potential unpredictability of a desynchronization.

After identifying the types of forces which may cause a desynchronization, I identified the types of desynchronization that can occur. "Reaction times to constraint violation in haptics: comparing vibration, visual and audio stimuli" [5] which measures the distance at which the virtual and real hand positions can be desynchronized before a user notices, differentiates between the rotation and position of the virtual hand. As such the difference in rotation and the difference in position are tracked as separate values when evaluating synchronization. The

rotational difference is defined as the angle (in degrees) between the controller's reported rotation and the rotation of the hand object in virtual space, whereas the positional difference is defined as the difference between the (x, y, z) coordinates of the two.

5.2 Test Design

For testing our solutions' effectiveness I put together three situations, some of which have variants that affect the condition which the test is evaluating. First, let's lay out some global properties of each test case which I decided would affect the measurement of effectiveness of the solutions.

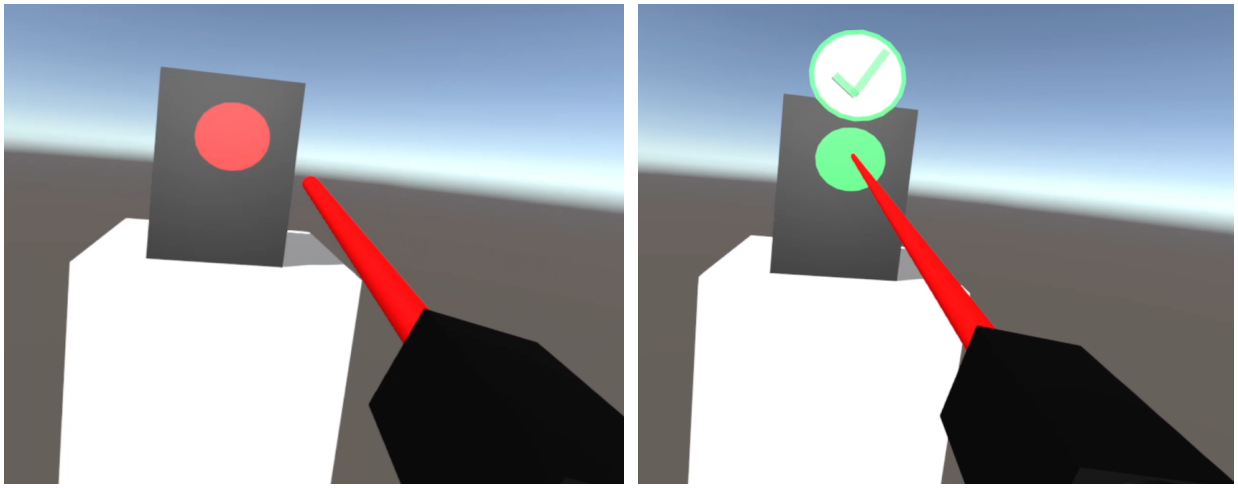
In each task, the user may only interact with the objects using one hand at a time. This hand can either be the left or right hand. This was chosen because the use of a single hand should cause a greater desynchronization, as the amount of force the user can apply with two hands is greater than that of one hand. This also makes the test cases simpler to design.

The user is not allowed to relocate their position in the environment using any industry standard method of relocation such as teleport movement or slide movement. Physically moving about their play area will move them however, as it may cause motion sickness to restrict this movement. The user may not relocate their position since movement in virtual reality is a study space of its own, and the design problem should be as isolated as possible in our test environment.

5.2.1. Long Stick

The first test is the long stick test. In this test the user is standing some distance from a button. The user is provided a stick which can grow in length using some input on the controller. The user is to expand the stick so that they can reach and press the button. Between the user and the button there will be disruptors which will apply force to the stick and cause desynchronizations. This test is intended to test the effectiveness of my solutions in achieving reactability and control.

This case was designed as a caricature of the types of displacements caused in fencing. Forces applied to the object in-hand will be amplified by the distance at which the force was applied from the grip point. This means using a longer stick causes a greater desynchronization.



(a) Start of the test (b) Goal Completion

Figure 6: Completion of the objective in the long stick test

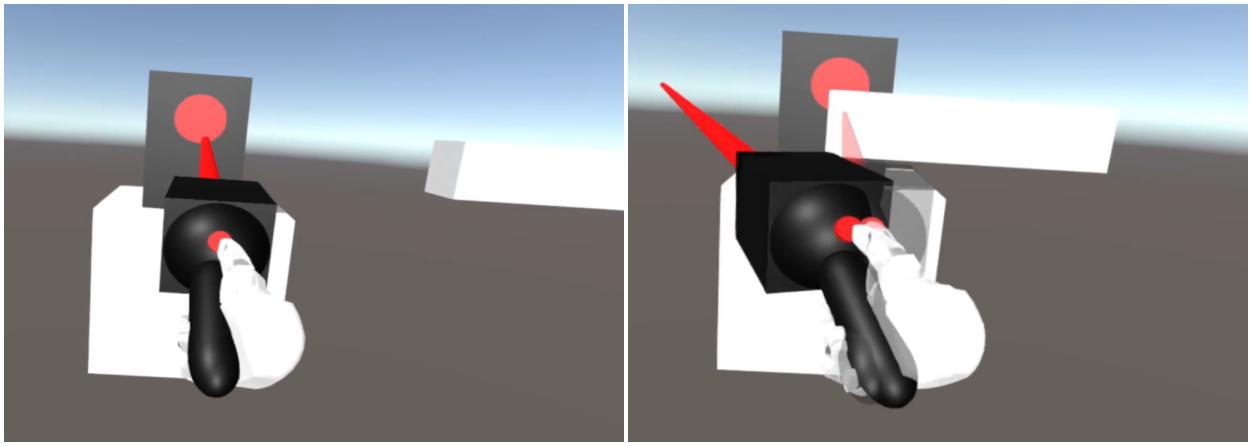
5.2.1a. Disruptors

The Long Stick test case is flexible in the types of forces it can test. By changing the disruptors this test can adapt to be a case of impulse force, persistent force, or chaotic force. In this thesis I propose two types of disruptors. One disruptor applies impulse force and the other applies persistent force. In extending the amount of different disruptors the complexity of the task will also increase, as such there should be as few disruptors as possible to meet the goals of this test.

The first type of disruptor is a piston. If an object passes into a predetermined area in front of the piston, it will shoot out and withdraw quickly, hitting the object in a sudden impact. Once a piston has fired, it will not fire again for some time, regardless of whether or not an object is in its trigger area. This object exerts an example of impulse force.

In this test case, the walls between the user and the button are made up entirely of pistons. Most of these pistons will be inactive, and will not act as disruptors. These pistons exist so the user cannot foresee the incoming impulse forces and avoid them. Pistons will be placed such that there is a guaranteed amount of disruptions between the user and the button.

The next type of disruptor could be a large fan. This fan will turn on and off on an interval. In front of the fan there is an area in which the stick will have a persistent opposing force applied to it, offsetting it from the user's hand. This area is visualized by the presence of wind-streak particles. The fan represents an instance of persistent force. In this thesis I did not implement the fan disruptor.



(a) Before (b) After

Figure 7: Action of the piston before and after firing. The ghost object variant of the ghost hand solution is active.

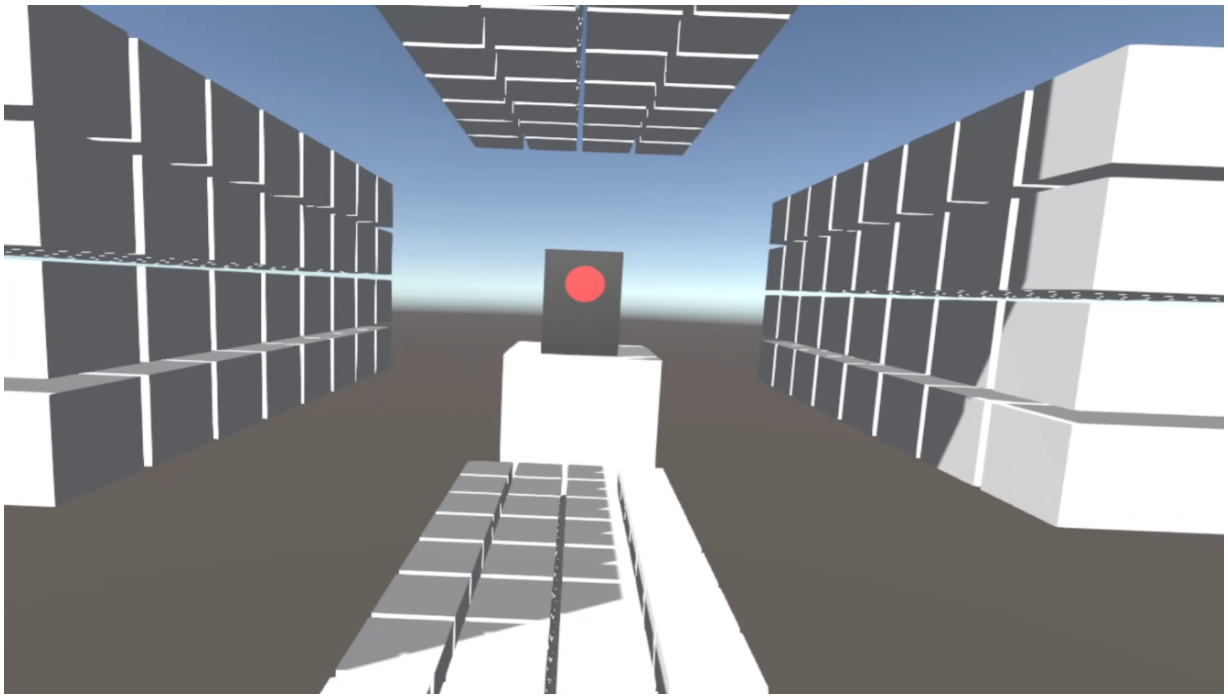
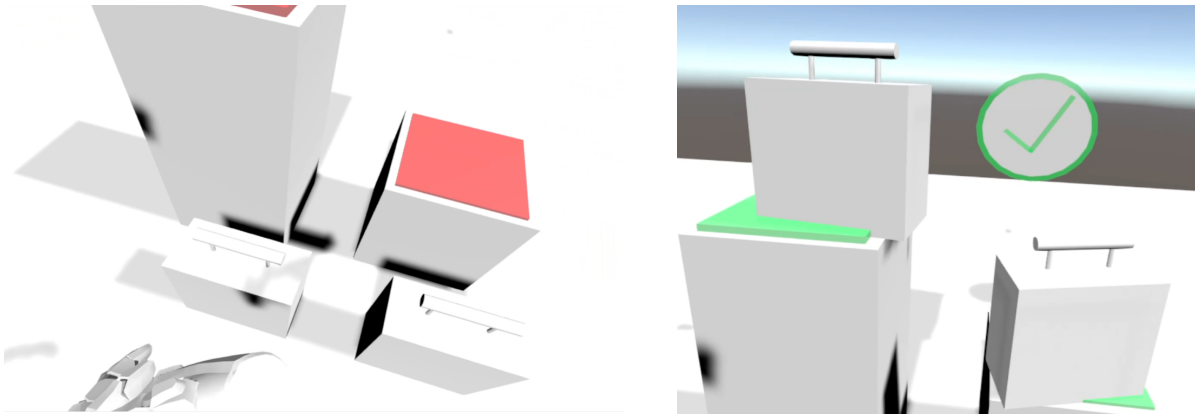


Figure 8: Sample environment in which many active and inactive disruptors are placed between the user and the button.

5.2.2. Heavy Object

In this case the user will be asked to move multiple objects from the ground into specific elevated positions in the play area. The elevation which the object needs to reach should adapt to the player's height to ensure that the player is actually capable of reaching the surface, however each surface does not need to be of equal height. Each of these objects are weighted such that the force of gravity on that object is close to, but lesser than the amount of force the user can apply to lift the object. This test seeks to measure the effectiveness of my solutions in achieving clarity and control



(a) Start of the test (b) Goal completion

Figure 9: Completion criteria of the heavy object test

5.2.3. Maze

In the maze task, the user is asked to drag an item from one end of a maze to the goal point on the other end. Any desynchronization caused in this task represents a traditional case of desynchronization, which is described in section 3.1 as user-driven desynchronization. It is also possible the user will not desynchronize at all while performing this task, though desynchronization can be forced by having sections of the maze outside of the user's direct reach, since they are unable to reposition themselves. This test seeks to measure whether my solutions have a positive, negative, or no effect on user driven desynchronization as well.

The difficulty for this task can be adjusted by increasing the friction of the object inside the maze, which will make the object more difficult to move, and the complexity of the maze. Though this task is named "maze" there should only be one path to the end goal with no dead-end paths in between, as this test is not intended to measure how good the user is at solving mazes.

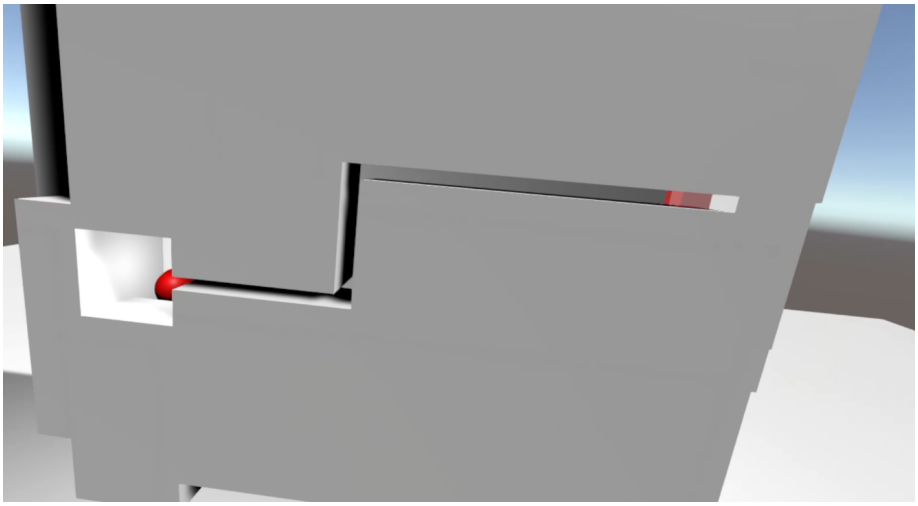


Figure 10: An example maze task at the beginning of the test. The red ball on the left is to reach the red highlighted target area on the right.

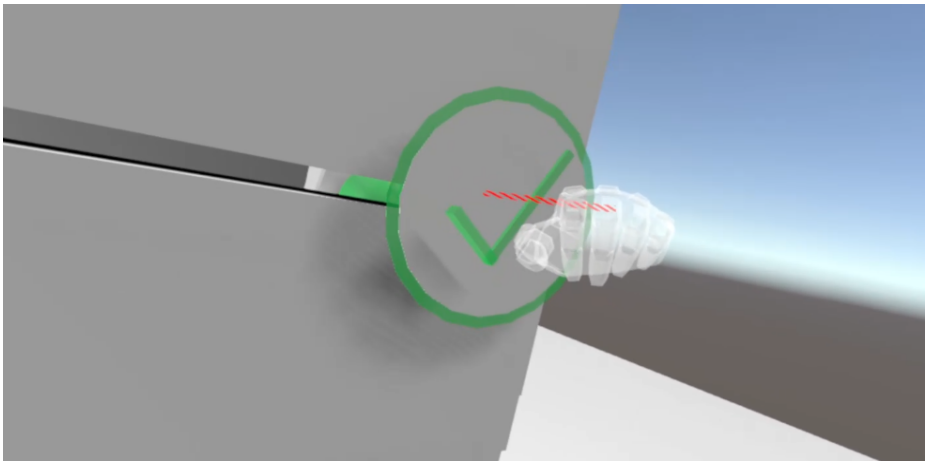


Figure 11: Completion of the maze task

6. Test Procedure

In this section, I will discuss some considerations on executing the tests described in this thesis as a user study.

6.1. Input Ambiguity

In all the proposed tests there are actions which are handled in different ways across different games. For example, every test involves the act of grabbing an object. Some virtual reality games will use the trigger button on the controller as the grab command whereas others will use the buttons on the sides of the controller, pressed by squeezing the controller. For this reason, the user should be allowed to choose their preferred grab input.

6.2. User Training

The test cases described in the previous section all rely on some sort of gamified goal to achieve. Each game has its own rules and disruptors, which have their own rules, as such, the study should provide users with sufficient training to understand the game's rules so that the measurements aren't evaluating how fast the user learns each game, and instead are measuring the effectiveness of the solutions.

6.2.1. Long Stick

The long stick case involves a few different rules with which the user must be familiarized with.

First, the stick itself can grow by pressing a controller button. This may be an unintuitive action, and as such the design of the stick itself may require its presentation to be more elaborate than just a stick, such as having it protrude from a device.

Next there is the act of achieving the goal, which is to extend the stick to press a button.

Finally there are the individual requirements presented by each disruptor in the long stick case; the fan and the piston.

To familiarize the user with the goal of the test, the user should first be given a case in which there are no disruptors. After which the user should be introduced to each type of disruptor independently so they can understand the independent behavior of the disruptors.

In every case, the button should be far from the user's reach to first familiarize the user with the stick's ability to extend to reach the button. Written or visual instructions should exist to inform the user on how to do this.

The pistons of the long stick case should showcase the behavior of both active and inactive pistons, as the walls themselves of the long stick case will be made of pistons both active and

inactive. The user should also be shown that a triggered piston will not trigger again for some time after being triggered. Both an active and passive piston should not be avoidable in completing the objective during training.

The fan is a simple object to showcase as it can be made easily unavoidable, as long as the fan triggers on an interval less than that of the amount of time it would take the stick to grow past the total distance covered by the fan, it is guaranteed that the user will be disrupted by the fan.

6.2.2. Heavy Object

The heavy object case should showcase that the opposing force is gravity. While it may seem to be common sense that heavy objects are in fact heavy, this talk [6] claims otherwise. In this talk, which discusses the troubles of implementing doors in virtual reality, Valve's Kerry Davis claims that users were unaware that they had to turn a doorknob in virtual reality, and instead believed that the door was locked when it did not open from simply pulling on it. Davis asserts from this behavior that the assumptions made in real world interactions do not always apply in virtual reality. The time at which this is mentioned in the talk is around 24:19. From this, I infer that the user will need to be educated of the weight of an object. One way in which the user can be informed of this is to have the objects originate in the air when the scene loads, and have them fall in front of the user before they can interact with the objects. Additional effects such as dust scattering when the objects land, or a heavy "thunk" sound effect upon landing may accentuate the weight of the objects.

6.2.3. Maze

A trial case for the maze task can be the most trivial possible maze, which would be a straight line from the start to the finish point, or a maze which has minimal turns to educate the user on navigating through the maze. The trial runs can be made easier still by increasing the amount of the object the user is able to see in the maze by making the viewport larger, or making the front plane of the maze slightly transparent. With a transparent front plane, care must be taken to not confuse the user as to why the object is stuck in the maze.

6.3. Counterbalancing Solutions

One thing to consider when serving the study is measuring the effectiveness of each solution individually and in combination, covering as many combinations as possible. The performance of our solutions should also be compared to the task performance when using none of the solutions.

In our related work, specifically in current commercially available sword fighting games, the most common way of handling desynchronization outside of avoidance is to not notify the user of a desynchronization, and instead leave the user to figure it out themselves. For that reason the solution which all solutions combinations should be measured against is the case in which the user is not notified of a desynchronization.

For this thesis I chose not to implement the sound solution due to its accessibility issues, and because it shares a hypothesis with the haptic method. This gives us a total of 9 different combinations that would have to be considered, which would cover the cases of using none of the solutions, each individual solution independently, every combination of 2 solutions, and every proposed solution at once.

This number of combinations is assuming that for any solution with variants, you are choosing a single variant to test. Depending on the anticipated number of participants it may be necessary to choose only one variant, as adding variants to the list may increase the required number of participants significantly.

6.4. Training Effect

One thing to consider when running the study would be the training effect throughout the study; or how quickly the user adapts to recognizing desynchronization and reacting to it. If the training effect for any solution is stronger than that of users using no solution, this could be an important indicator to the effectiveness of that solution. As such, the tests should be ordered in a way which encourages this effect.

6.5. Recruitment strategy

While it may be simpler to run this as an in-person user study, that may be impossible depending on the local circumstances of where this study is run. Furthermore, a greater sample size may be reachable by doing an online study, which would allow testing of more solution variants than an in-person study.

There is no shortage of ways the study could be distributed digitally. The simplest way would be to upload the study itself to Steam, which is the largest digital distribution platform for games on the PC platform. Putting the study on Steam would also advertise the study in itself, as Steam will likely advertise the study's software to users who own virtual reality hardware. Digital distribution platforms also allow you to choose which world regions to make the software available to, which may help ensure that the study is distributed to regions which share the same language as the study's consent form, instructions, and surveys.

While it is a valid concern that the study may be drowned out by the large volume of games being distributed by Steam, virtual reality games are still a niche market with a much lower volume of releases than other genres. As such it is less likely the study will be drowned out by competition on a digital distribution platform.

Another source of advertising when doing an online study could be to reach out to leaders of virtual reality communities to be allowed to advertise there. The virtual reality community on Reddit (https://www.reddit.com/r/virtualreality/) alone has 237,000 users, and appears to allow for self promotion as long it is appropriately categorized, and not done in excess.

6.6. Session Recording

While it may be easy to record a user's session in an in-person study, it will be much more difficult to do for a remote study. This does not mean that it is impossible to do so however. By capturing the position and orientation of the user's controllers each frame in which the study's physics engine updates, as well as recording the user's inputs and their corresponding frames, we can re-run the session locally, simulating the user's experience running through the test.

In this thesis' implementation we use the Unity engine, so it would be simple to replay controller positions and orientations, since Unity's physics system updates on a guaranteed fixed interval. It would be more difficult to collect inputs such as the grab command since those happen in a different update window which does not run on a fixed interval, and is very likely to differ in time between machines. For this reason it would be better to bind input timings to their nearest physics update frame, or the time of the input since the first physics update frame.

Session recording will be an important feature in this study as it will allow us to manually verify inferences made from the qualitative data, and identify the nature of any anomalies.

6.7. Evaluation Metrics

Qualitatively, in each test we can measure the time to completion, the number of desynchronization events, and data detailing each desynchronization such as:

- The duration of the desynchronization
- The distance of the desynchronization
- The nature of the desynchronization (positional, rotational)

The qualitative measurements will help evaluate whether the solutions meet the goals of clarity, reactability, and control. However since presence is a subjective goal, it cannot be measured from the qualitative data. For evaluating presence there exists many standardized questionnaires which the user can be asked to complete [11].

7. Subjective Assessment of Effectiveness

In this section I will be assessing the effectiveness of every task and solution proposed in this thesis. These assessments are subjective based on my personal experience while developing and performing the tasks.

7.1. Long Stick

For this test, I tried various interaction methods, including one-handed grip vs two-handed grip, with or without the ghost object variant, and with or without recoil stabilization which is described

in section 8. When implementing this test I did not implement the persistent force disruptor, only the impulse force disruptor; the piston.

In my experience the most comfortable input method was a one handed grip with recoil stabilization, and the most effective solution to implement was the ghost hand solution with the ghost object variant.

The long stick test was drafted to study the effect of my solutions on meeting the goals of control and reactability. Unfortunately in its current conception, it is my opinion that it can't give us an accurate measure of either goal, as at present the best thing to do when the stick is disrupted is nothing. Continuing to aim at the button and grow the stick while waiting for the stick's orientation to return to that of the controller seemed to be the best solution.

One possible improvement that could make this test meet its intended goals would be to add a reactive action the user must perform once the stick has been disrupted.

Another improvement could be to make the button itself move along a looping path, making it impossible for the user to simply keep their hand straight to press the button. This has an additional effect of causing desynchronizations in of itself, as due to the physics in the interaction the stick has more sway in its movement when it is longer.

7.2. Heavy Object

While lifting the object, the path the object takes is a slow, predictable climb as the object lifts slowly towards the hand. However when it comes time to place the object downwards it begins to test control, as during the shift from having gravity oppose your action to having it add to the direction of movement, it becomes easy to overshoot the objective.

I made an additional observation while reviewing recordings of myself completing the test. While there is a limit to how far the virtual hand can desynchronize from the controller position before it stops applying any additional force, I felt the need to raise my hand even higher still to raise the heavy object up to the objective. And while I ran all of the tests in a seated position, in this test I stood up at one point to raise my hand even higher. Having exerted myself in spite of knowing that doing so does not increase the speed at which it rises indicates a strong feeling of presence in the interaction to me.

For these reasons I think the heavy object test meets the goals for which it was designed, and is an effective test in which the solutions can be assessed.

7.3. Maze

When performing the maze task trying to keep synchrony the entire time, or with no solutions active, I found that the task was quite cumbersome, since I had to follow my head along the trail to see where the object was while holding my hand outstretched. When I instead took

advantage of my solutions and forced my hand into a desynchronized state, I was able to approximate the position of the object in the maze using the line solution (see figure 12). This made completing the test much more comfortable.

In my opinion, it won't be difficult to see a positive difference in this test's performance using the solutions in this thesis.

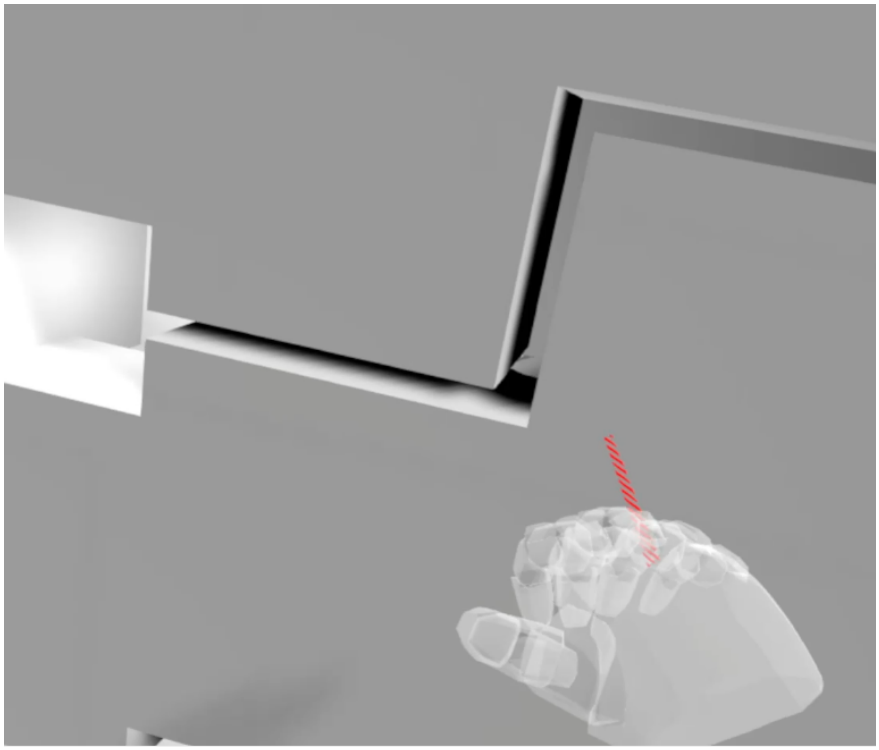
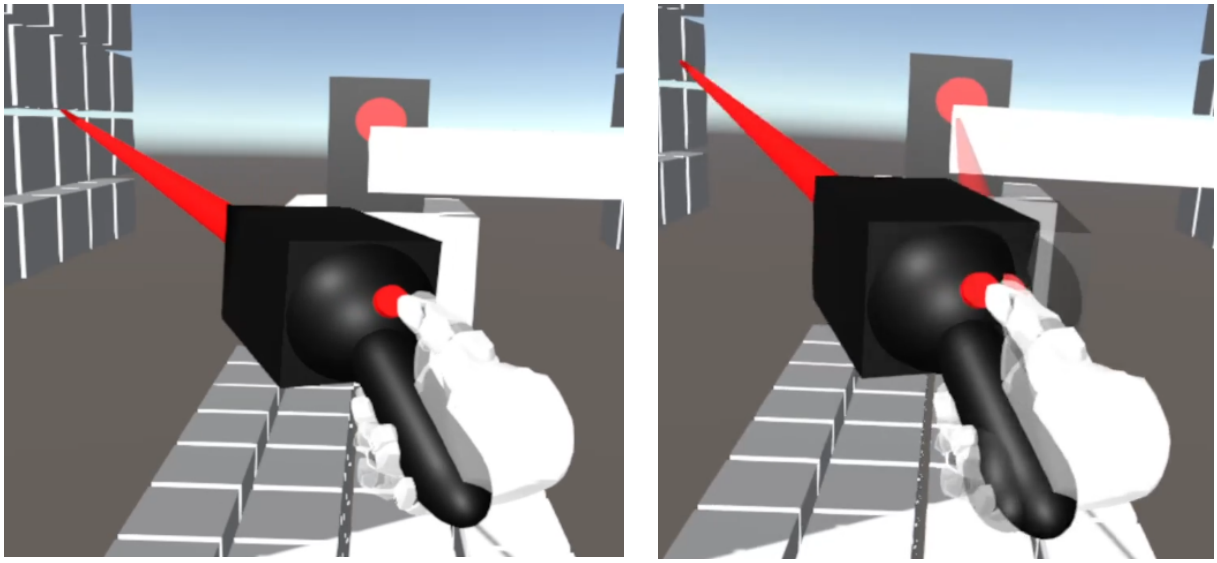


Figure 12: Utilizing the line and ghost hand solutions to approximate the position of the object within the maze

7.4. Ghost Hand Solution



(a) Ghost hand only

(b) Ghost object variant

Figure 13: A comparison of a desynchronized state during the long stick test with utilizing the ghost hand solution and without the ghost object variation

The ghost hand solution was good in situations in which the desynchronization was positional, but in situations where the desynchronization was rotational, I feel it had no effect, as the ghost hand appears on the same position as the real hand. In cases of rotational desynchronization, the ghost object variant of this solution felt much more effective.

During the long stick test, the ghost object variant was comfortable to use, as it showed me where exactly the object was going to end up when synchrony was restored. I think this contributed to the effect described in the long stick test's evaluation in section 7.1, since I could see that I did not have to move my hand to complete the objective.

During the long stick test there would be rotational desynchronizations which had a low magnitude when measuring the difference between the hands, but when applied further down the stick there would be a much larger desynchronization between points at the tip of the stick. In the Solutions section 4 I detailed that there are two types of listeners which monitor positional and rotational synchronization. For the ghost object variant a third type of listener may be useful which measures the distance between the points at the far tip of the object, since for longer objects such as the long stick, the distance between points at the end of the object can become significant with small rotational offsets.

The ghost object variant was less effective in cases where the loss of synchronization was positional such as the heavy object and maze tasks, and I feel it offered nothing notable in those cases. What I took away from this observation is that there may not be a singular best solution or combination of solutions, and the solution of choice may depend on what the expectations of

the interaction are; in the long stick test I felt it was more effective to use the ghost object variant and that the line and ghost hand solutions were ineffective, whilst in other tasks I found other combinations of solutions more effective, as described in section 7.3 and section 7.5.

7.5. Line Solution

The line solution provided excellent clarity and control in situations of positional desynchronization that happen in the heavy object and maze tasks. In the heavy object task it directly shows the path the object was taking on the way up, and in the maze task it allows a more flexible range of control, since you can approximate the object's position in the maze by taking advantage of the line.

Additionally, since the line animates from the virtual hand towards the real hand, it illustrates directly that you are applying a force that is pulling the virtual hand, and clarifies the direction of that force. Though a subtle detail, I believe that this increases the clarity of what is happening during a desynchronization.

However, in situations where the desynchronization is from a difference in rotation such as the long stick test this solution had no noticeable effect. Since there was a low positional difference between the virtual hand and controller in the long stick test, the line was not even visible.

7.6. Haptic Solution

The haptic solution felt effective in notification of a desynchronization, but also held the potential to become annoying after repeated desynchronizations. During the long stick case, the object would often recoil while restoring synchrony, which resulted in multiple desynchronizations and thus multiple controller vibrations. Additionally during the heavy object case when placing the object on the platform, the object would at times bounce in and out of the synchronization bounds. This annoyance could possibly be mitigated by implementing a cooldown period on the notification, or an amount of time that must pass after a haptic pulse before it is allowed to vibrate the controller once more.

8. Implementation

For this thesis, the Unity game engine was used to implement the proposed solutions and tests. This choice was made due to Unity's ease of use, and my own familiarity with the engine. On top of that, Valve's SteamVR plugin for Unity was used to interact with the virtual reality hardware itself.

Additionally this thesis makes use of the AutoHand library to allow the user to interact with items in the virtual environment, and handles the physics behind the user interactions in the indirect influence method described in section 8.2.2.

The first synchronization method described in section 8.2.1, direct-object influence, was implemented using only SteamVR default assets to interact with the environment, with supporting scripts in place to handle the physics of the interaction. However, after finding the AutoHand library, and comparing its method to my own implementation, I ultimately chose AutoHand's solution of indirect influence.

8.1. Restoring Synchrony

There are two different types of synchrony that need to be restored, positional and rotational. Each type of synchrony is restored independently of one another.

8.1.1. Positional Synchrony

Positional synchrony is restored each frame by calculating the distance between the virtual hand and the actual hand, then multiplying that by a coefficient that defines how strong you want the restoration to be. The highest coefficient that can be used for a rigidbody's velocity in Unity is $\texttt{Time.fixedDeltaTime}^{-1}$. Since `Time.fixedDeltaTime` is the time between each physics update in seconds and velocity is in units per second, multiplying the distance between the two objects by this coefficient returns the velocity which will restore positional synchrony in a single FixedUpdate frame. Any value higher than this and the hand will consistently overshoot the restoration point and will never restore synchrony. This method of restoring positional synchrony described in the following sections apply to both the direct-object influence I implemented, and indirect influence implemented in AutoHand, with the main difference between the two being the target which the force is being applied to.

8.1.2 Rotational Synchrony

Rotational synchrony can be restored using the same method, except the way the delta in rotation is calculated is different. Since rotation is in quaternions in Unity, you must multiply the rotation of the controller by the inverse of the virtual hand's rotation. This can be converted to an angular velocity by calling toAngleAxis on the resulting quaternion. The strength of restoration can be adjusted by multiplying the axis vector returned from this function by some strength coefficient. This process will give you a vector that can be applied to the rigid body's angular velocity to restore rotational synchrony.

8.2. Synchronization Methods

Two methods were implemented to restore synchrony when implementing the physics which draw the virtual hand to the controller's position; direct-object influence, and indirect influence. The primary difference between the two methods is the target of the forces which draw the hand and the object being held into place. After developing an implementation of direct-object influence, and comparing that to the indirect influence which the AutoHand library implements, I decided ultimately to use indirect influence.

8.2.1. Direct-Object Influence

With Direct-Object influence, the restorative forces are applied directly to the object being held by the user. If there is no object in the hand, then the forces are applied to the virtual hand instead.

8.2.1a. Pros

Since we're modifying the velocities of the object directly, this method allows us to exert more control over the velocities of the object in hand than Indirect Influence, and the object will track to the controller position more effectively. With this method it is also easier to control the recoil problem mentioned in section 8.2.2c.

8.2.1b. Cons

When modifying velocities of a Rigidbody directly in Unity, you will lose any momentum on the object when you override its velocity. This means any case of impulse force will be overridden. If this result is undesirable then you will need to write scripts yourself to track any instances of impulse force. Since impulse force is naturally dissipated over time by the forces that are applied to restore synchrony, you will also need to dissipate those forces manually.

Further difficulties include point force, or applying the force at the point which the object is being held to restore synchrony. Specifically, rotational synchrony becomes difficult in this case, as the rotation of a rigidbody is about its center of mass. That center of mass may not be the point at which the object is being held, in which case the object will rotate out of the user's hand when applying a rotational velocity. This can be avoided by setting the center of gravity of the held object to the grab point, but that may be undesirable for recreating realistic object interactions.

8.2.2. Indirect Influence

Indirect Influence calculates and implements the restorative forces in the same way as Direct-Object Influence, but instead takes control of the hand object's velocities. Upon grabbing an object, the hand will attach its own rigid body to the grabbed object at the point which the object is being held.

8.2.2a. Pros

By affecting the velocities of the hand object instead of the object in hand, all of the cons of direct object influence [8.2.1b] are circumvented.

This method uses Unity's built in methods of resolving rigidbody interactions to resolve the force difference between the restorative forces and any opposing forces. Furthermore, since the rigid bodies are joined at the point at which the object is being held, the force-at-point interaction is also resolved automatically by Unity's physics engine.

8.2.2b. Cons

Since we aren't taking full control of the held object's velocities, the force generated by attempting to restore synchrony can become greater than the force the hand can exert. This is because the force the hand can produce is calculated as f=mv using the hand's mass, not considering the mass of the object in hand.

The effect of this interaction is that the virtual hand may overshoot the real hand's position or rotation, and will have to swing back the other way to try and restore synchrony. This process can repeat multiple times, resulting in reduced control over the situation for the user, which may be undesirable.

8.2.2c. Managing Recoil

To mitigate the recoil of an object, I implemented a simple script that mitigates the amount which the object in hand will recoil.

Put simply, if the virtual hand is moving towards the controller, and the distance it will travel this frame is greater than the distance between the virtual hand and the controller, then an impulse force is applied in the opposite direction of the virtual hand's current velocity. The magnitude of this impulse is equal to the distance that the virtual hand is expected to overshoot the controller's position in the next physics update.

At first I expected this script would completely negate any sort of recoil, which may be an undesirable effect when considering the fidelity of an interaction. However, since the velocity calculation is only using the hand's mass, the overshoot is only mitigated, not negated. Furthermore, the strength of the recoil compensation can be adjusted by adding a portion of the mass of the object being held to the hand's mass in the force calculation.

9. Conclusion

Three methods of notifying the user of desynchronization were proposed and developed to meet the goals of clarity, control, and reactability. To measure whether these solutions meet their goals three tests were proposed, as well as a potential strategy for performing the tests in a user study. In lieu of a user study, I have given my opinions on the effectiveness of both the solutions and the tests, in which I suggest modifications that may improve the tests such that they can more accurately measure the effectiveness of the proposed desynchronization solutions, as well as improvements to the solutions themselves.

9.1. Future Work

For the immediate future, I would like to test the solutions outlined in this thesis as a user study so I can draw data-driven conclusionsons on the effectiveness of the solutions, instead of supplying anecdotes from my experience.

Looking further ahead, I would like to study larger issues caused by external forces acting upon a player in virtual reality. This thesis is concerned with external forces causing a desynchronization of the hands, but how would a game handle the case in which an external force is pushing the player's whole body, such as a push to the chest? Physically forcing the headset's position to move in the virtual environment whilst their body is not moving would likely cause motion sickness, which means this case would require creative solutions to both inform the player that they are being pushed, and to restore synchronization in the player camera to their virtual body.

10. References

[1] Subutai Corporation, “CLANG”, *kickstarter*, June 9th 2012 https://www.kickstarter.com/projects/260688528/clang

[2] Occulus, “Oculus Rift: Step Into the Game”, *kickstarter*, Aug 1 2012 https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game/description

[3] Zenner, André, and Antonio Krüger. "Estimating detection thresholds for desktop-scale hand redirection in virtual reality." *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 2019.

[4] Zilles, Craig B., and J. Kenneth Salisbury. "A constraint-based god-object method for haptic display." *Proceedings 1995 ieee/rsj international conference on intelligent robots and systems. Human robot interaction and cooperative robots*. Vol. 3. IEEE, 1995.

[5] Peon, Adrian Ramos, and Domenico Prattichizzo. "Reaction times to constraint violation in haptics: comparing vibration, visual and audio stimuli." *2013 World Haptics Conference (WHC)*. IEEE, 2013.

[6] Kerry Davis. “Valve's Own Kerry Davis' HLVR Door Talk” *YouTube*, uploaded by Tyler McVicker Extra, 16 Sep. 2019, https://www.youtube.com/watch?v=9kzu2Y33yKM

[7] “Parry This | Shadow Legend VR”, *YouTube*, uploaded by TeraKnight, 26 Mar. 2020, https://youtu.be/Ip6jasUDPDY?t=449

[8] “Freestyle VR Sword Gameplay - UNCUT SKILLFUL ENDLESS BLADE AND SORCERY”, *YouTube*, uploaded by VirtualRageMaster, 5 Mar. 2019, https://youtu.be/CswwQTXd-9w?t=107

[9a] “Free Company VR - Weapon Physics 3 (WIP)” *YouTube*, uploaded by Pomshine Games, 8 Feb. 2020, https://www.youtube.com/watch?v=ocwADp9BINc

[9b] Pomshine, “Updated my IK blocking system to look more natural and support more weapon types.”, *Reddit*, 22 Jun. 2019, https://www.reddit.com/r/Unity3D/comments/c3vuym/updated_my_ik_blocking_system_to_look_more/

[10] Tetragrammaton, “Here's what "slow-motion melee" looks like in Ironlights (VR multiplayer swordfighting game).”, *Reddit*, 22 Jan 2020, https://www.reddit.com/r/oculus/comments/esez5u/heres_what_slowmotion_melee_looks_like_in/

[11] Schwind, Valentin, et al. "Using presence questionnaires in virtual reality." *Proceedings of the 2019 CHI conference on human factors in computing systems*. 2019.

11. Media Sources & Licences

Figure 1

https://commons.wikimedia.org/wiki/File:Final_Challenge_international_de_Paris_2013-01-26_193155.jpg

Wikimedia Commons: https://commons.wikimedia.org/wiki/Main_Page
CC-BY 2.5: https://creativecommons.org/licenses/by/2.5/

SteamVR plugin for Unity

https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647

AutoHand plugin for Unity

https://assetstore.unity.com/packages/tools/modeling/auto-hand-vr-physics-interaction-165323