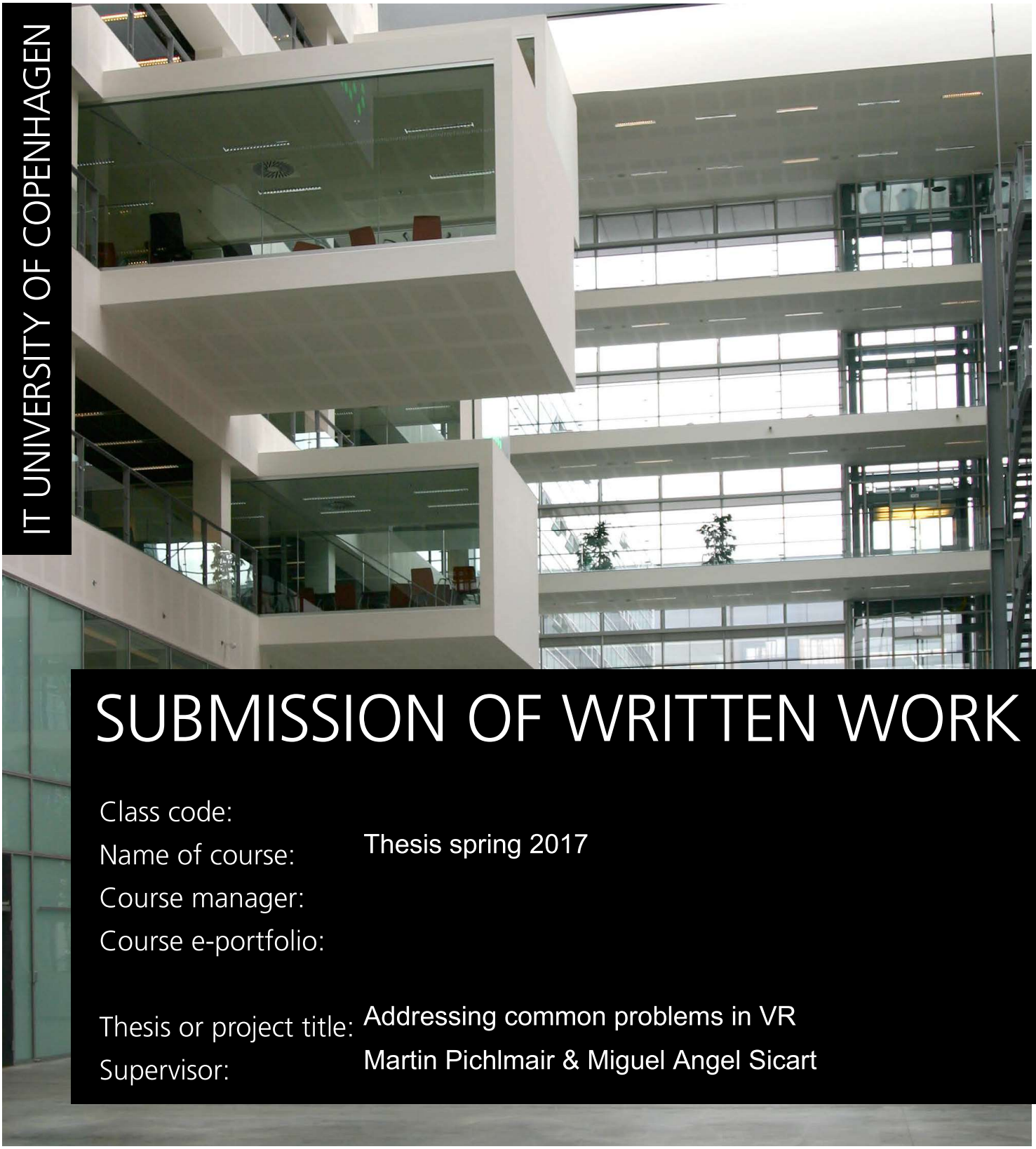
IT UNIVERSITY OF COPENHAGEN



SUBMISSION OF WRITTEN WORK

Class code:

Name of course: Thesis spring 2017

Course manager:

Course e-portfolio:

Thesis or project title: Addressing common problems in VR

Supervisor: Martin Pichlmair & Miguel Angel Sicart

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Óscar Manuel Losada Suárez	30/05-1992	oslo@itu.dk
2.	Phillip Phoelich	26/02-1992	ppho@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Thesis paper

Controlling Hands in Virtual Reality

Authors
Óscar Manuel Losada Suárez*(oslo@itu.dk)*
Phillip Phoelich *(ppho@itu.dk)*

Supervisors
Martin Pichlmair
Miguel Angel Sicart

Date
June 1st, 2017

MSc in Games
IT University of Copenhagen

Abstract

Consumer-oriented Virtual Reality (VR) hardware now allows users to interact with virtual environments with their own hands through the use of hand-held spatially tracked controllers. However, the physics constraints of the virtual environment do not apply to the user's real hands. This implies that if the virtual hands follow the movements of the real hands exactly, the user can make the virtual hands break the expected physics rules of the virtual environment e.g. by moving their real hands inside of virtual objects. This work addresses this problem by prototyping ways of controlling hands in VR that allow the virtual hands to deviate from the tracking data by adjusting the position, rotation and finger placement when there are conflicts between the user input and the virtual environment constraints. The value of this hand control model is assessed by comparing one of the developed prototypes with a hand control model that imitates the hands in the critically and commercially acclaimed VR game Job Simulator in controlled experiments. Much work remains to be done, but the results of the experiments are positive and seem to indicate that the user experience could be improved using this type of hand control.

Keywords: Virtual Reality, VR, hands, hand control, hand presence.

Acknowledgements

We would like to dedicate here a few words in appreciation of the help and support given by our friends and families. Special thanks are due to Anton, Lucas, Peter and Jonathan for sharing their office and good spirits with us, to Sarah, who listened to Óscar ramble about the thesis and let him write from her windowsill, and to Phillip's parents for their hospitality in letting him work from their blooming garden in the countryside. And of course, we thank our supervisors, Martin and Miguel, for their feedback and confidence in us.

Contents

Every motion of the hand in every one of its works carries itself through the element of thinking, every bearing of the hand bears itself in that element. All the work of the hand is rooted in thinking. Therefore, thinking itself is man's simplest, and for that reason, hardest handiwork, if it would be accomplished at its proper time.

Martin Heidegger, "*What is called thinking?*"[1]

[1] Heidegger, 1968.

1 Introduction

In our day-to-day lives, we constantly use our hands to hold, operate and otherwise manipulate objects, tools and devices with precision. We use them to gather tactile information about the world and even to communicate with each other through hand gestures and body language. They are a fundamental part of the human body and our way of interacting with the world around us.

The biggest buzzwords in the Virtual Reality (VR) medium: immersion and presence, perhaps point us in the right direction. VR is meant to bring us closer than ever before to experiencing virtual environments as if we were actually in them. Interaction is a big part of experiencing the places we are in. It should hardly be a point of contention then, that having virtual hands that imitate our own real hands can have a significant impact on VR experiences.

VR game developers and hardware manufactures alike have acknowledged this. Major consumer-oriented VR devices like the HTC Vive, the PlayStation VR with the PlayStation Move Motion Controllers (PS VR + Move) and the Oculus Rift + Touch have controllers that track the position of your hands in space (Valve Corporation & HTC Corporation, 2016; Sony Interactive Entertainment, 2016, 2010; Oculus VR, 2016b). In their About page, Owlchemy Labs, the creators of the critically and commercially acclaimed (Unity Technologies, 2016; Galyonkin, 2017) Job Simulator: the 2050 archives (Owlchemy Labs, 2016) declare:

> *We believe that interaction and using your hands is what truly makes virtual reality the most incredible place to build unique content that blows players minds.* (Owlchemy Labs, 2017)

Predictably enough, this is not the end of the story. Even if we had complete tracking of the hands and individual fingers, the physical constraints of the virtual environment would still not apply to the users' real hands, which leaves us with few essentially different options:

1. Prioritize user input and ignore virtual environment constraints whenever there is a conflict in order to always keep the virtual hands aligned with the real hands to the extent allowed by the tracking data.

2. Separate the virtual hands from the real hands when there is conflict in order to respect the virtual environment's constraints.

3. Use sensory feedback that makes the users either respect the virtual constraints on their own or feel like they are constrained without actually being so.

4. As (Schell, 2015) suggests, design in a way that circumvents the problem, which means restricting the medium's content to experiences that fit exactly to the medium's current affordances (Norman, 2013). This means seeing limitations as features and finding experiences that map perfectly to the current systems.

The work here presented is focused on the second option, which is seen as a directly opposite alternative to the first. We aim to explore how virtual hands can work in this case and investigate whether the user experience can be improved with this approach.

From this point on, we will refer to this problem as the Virtual Constraints Problem, to the approaches fitting the first option as Unadjusted Hand Control Models and to those fitting the second option as Adjusted Hand Control Models.

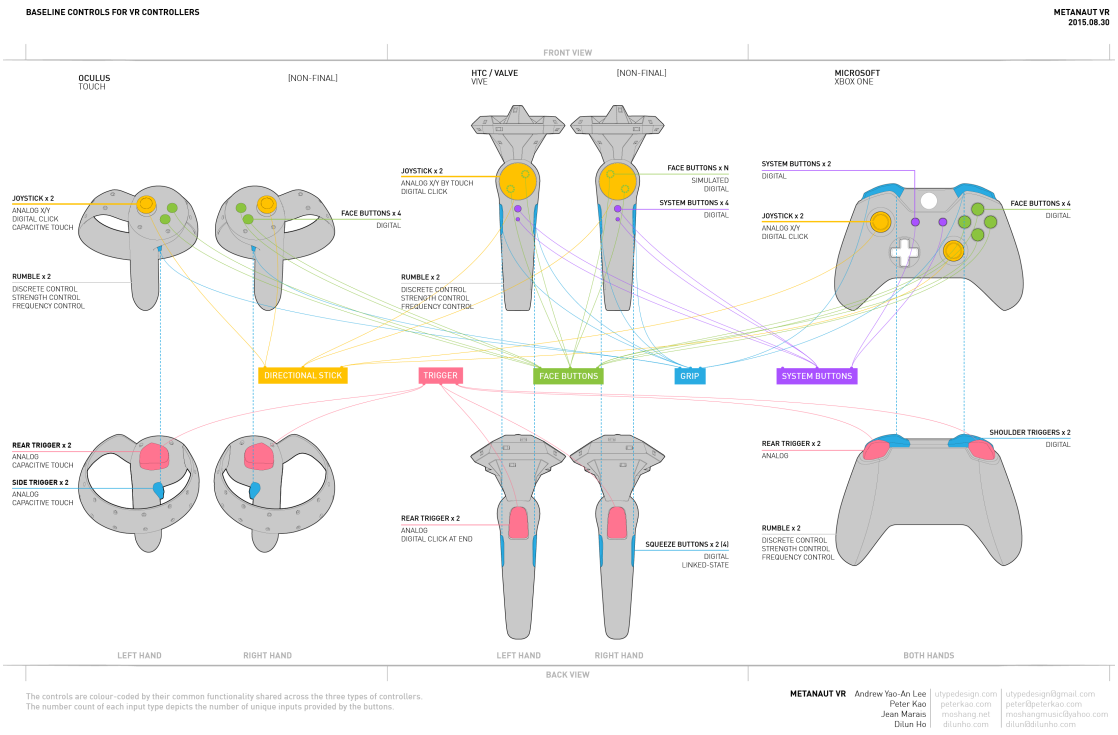


Figure 1.1: Comparison of VR controllers and their different input and output capabilities. Note that the final version of the HTC Vive controllers looks different. Retrieved from(Metanaut VR, 2015) and can be seen in a larger format in Appendix A.

1.1 State of the Art

1.1.1 VR Hardware

There is a considerable amount of devices and accesories that are relevant to VR. This work is mainly applicable to those setups that can spatially track the hands of the user to a similar or greater degree than the HTC Vive allows. We are particularly interested in findings that are relevant to the aforementioned HTC Vive, PS VR + Move and the Oculus Rift + Touch because they are currently the most important consumer-oriented devices with these capabilities (Armstrong, 2017; SuperData LLC, 2017).

There are differences between the Head Mounted Displays (HMD) of these devices, but for the most part, they are not relevant for our area of focus, so they will not be discussed here.

On the other hand, it might be worth briefly covering the differences in terms of spacial tracking of the controllers between the devices. The HTC Vive allows 360 degree tracking within the biggest space volume, the Oculus Rift + Touch with the default setup is designed mainly for 180 degree tracking, but an experimental setup with extra components allows stable 360 tracking (Lang, 2016; Kuchera, 2016).

Figure 1.1 shows the features of the Oculus Touch and the HTC Vive controllers. In the case of the HTC Vive, the most relevant features for our purposes are:

- the analogue triggers with digital click at the end, controlled with the index finger,
- the digital grip buttons, pressed with middle, ring and/or pinky fingers,
- the joysticks or trackpads, with analog x/y input through capacitive touch and digital click, used with the thumbs
- and rumble capabilities.

In the case of the Oculus Touch, the relevant features are:

- the analogue triggers with capacitive touch, controlled with the index fingers,
- the analogue side triggers, pressed with middle, ring and/or pinky fingers,
- the joysticks, with analog x/y input, digital click capacitive touch, used with the thumbs
- and rumble capabilities.

Figure 1.2 shows the input mechanisms of the PS Move Motion Controller, again, the most relevant features are:

- the analogue T buttons, controlled with the index fingers,
- several digital buttons controlled by the thumbs
- and rumble capabilities.

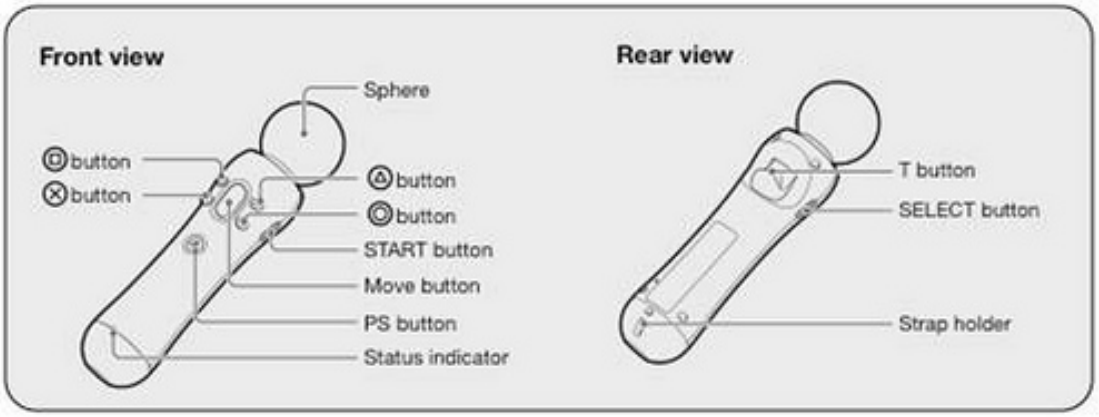


Figure 1.2: Diagram of the PS Move Motion Controller. T button allows analogue input. Retrieved from (Sony Interactive Entertainment, 2017).

Analogue buttons are useful to allow the users to control the motion of the virtual fingers with similar movements with their own fingers - in Norman's terms, they provide a natural mapping (Norman, 2013) for finger control. Similarly, although without the advantage of being analogue, buttons or areas with capacitive sensors can be used to monitor if the user's fingers are resting on the controller or are lifted, which can allow for guessing the position of the fingers.

The Oculus Touch is the controller that takes these input mechanisms further having a capacitive trigger for the index finger, capacitive buttons for the thumb and a normal trigger for the other three fingers. This can be used to detect a variety of gestures like index finger pointing, thumbs up (or down), finger guns... It also facilitates common interactions like pressing virtual buttons with your virtual index finger, operating a grabbed object with the same hand that is holding it (e.g. press gun trigger). Both the HTC Vive and PS VR controllers allow for a lower degree of natural finger control.

Other accessories exist that allow for actual positional tracking of the fingers, such as the Manus VR Gloves (Manus VR, 2016). These would of course enhance the experience, but ultimately would face the same problem because they don't provide a way to impose constraints from the virtual world on the real hands of the user.

1.1.2 VR Games

The most frequent approach to dealing with the Virtual Constraints Problem described at the beginning of this chapter is to ignore these constraints when they conflict with the user input. Job Simulator is a prime example of the unadjusted hand model and the one that we have used in our work as a baseline for comparison.

In Job Simulator the hands never separate from the position of the controller, but this still allows for reasonable physical interaction with small, light objects that can be easily moved by the hands. When the hands collide with movable objects, the object is pushed out of the way. However, when the virtual hands collide with objects that cannot be moved by them, the hands will simply go through them as if they weren't there as shown in Figure 1.3.

Figure 1.3: Sequence showing the Job Simulator hands entering an immovable obstacle. Captured in the HTC Vive version. A gif can be found here: https://tinyurl.com/JobSimTouchStatic .

At the same time, they avoid the problem of grabbing objects with correct finger placement by hiding the hands when they are holding an object as shown in Figure 1.4. This solution is not without merits: it allows the user to see the held object from all angles without the hand obstructing the view, it is simple and surprisingly, it is often not noticed by the users.

Ideally however, we would like the hand to be visible and the fingers to be correctly placed on the grabbed object forming a grip. But without adjusting the virtual hand, this requires the user to be too precise controlling the virtual hands and makes grabbing objects very difficult.

Very recently, a few games have started to show approaches to hand control that fall within the adjusted hand model. One these is Wilson's Heart (Twisted Pixel Games, 2017), released in April this year. Each object that the user can interact with has a predefined hand pose and when certain conditions are met, the hands will snap to the pose as shown in Figure 1.5. These conditions usually have to do with the user pressing

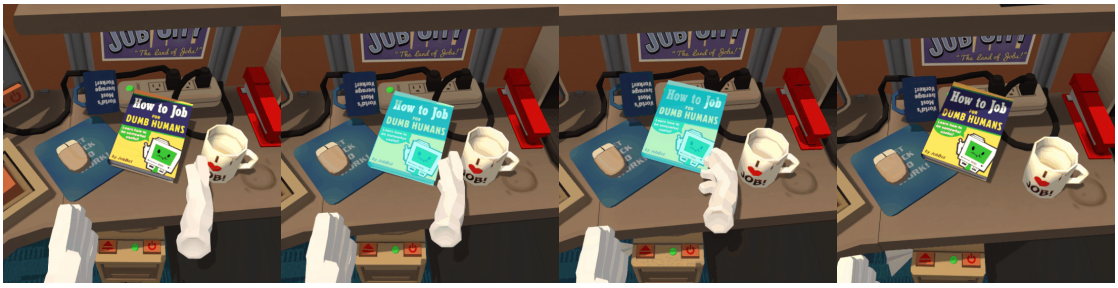


Figure 1.4: Sequence showing the Job Simulator hands grabbing an object. The hands are hidden while holding objects. Captured in the HTC Vive version. A gif can be found here: https://tinyurl.com/JobSimGrabObject .

some of the triggers or buttons that control the virtual fingers although there are also poses that are triggered by moving the hands close enough to the objects.

This technique is interesting because the virtual hands can separate a lot from the position of the real hands and very suddenly too. The underlying idea is that user's want to interact with each object in a certain way and will on their own pose their hands in a similar way to the object's pose, so when the pose snapping happens, the change will be small and, most importantly, aligned with the user's intention.

The pose-snapping method allows perfect placement of the fingers and looks great in pictures, but it requires a lot of manual work to setup every interactible object in the game and it takes away a lot of control from the users, forcing them to interact with each object exactly in the ways that the poses allow.

Figure 1.6 shows that the other issue identified with the Job Simulator hands is mostly unsolved in Wilson's Heart. The hands can still go through immovable objects, but in this case, an X-Ray style effect is used to visualize the parts of the hands that are inside of objects. This communicates the state of the hands to the player, but might not be preferable to other techniques that make the virtual hands respect the virtual world's physics.

The most novel and promising attempt within the adjusted hand model that we are aware of however, is an upcoming game called Lone Echo (Ready At Dawn Studios, n.d.). In this game, the virtual hands procedurally adapt to the virtual environment. Once again, the underlying principle is adjusting the virtual hand in a way that doesn't conflict with the user's intentions, but in this case instead of using authored poses the use of a real-time algorithm can make the system extremely flexible. At its best, this approach could empower the user and make them feel more control over their virtual hands by not forcing the interaction with objects to one specific pose. At its worse, there might be situations where the algorithm doesn't behave in a way that matches

Figure 1.5: Sequences showing the pose-snapping grab in Wilson's Heart. Two gifs showing the pose-snapping can be found here: https://tinyurl.com/whGrabPoseMirror and https://tinyurl.com/whGrabPoseCuffs .

the user's natural way of interacting with their hands which would have the opposite effect.

This approach will inevitably come at the cost of having some edge cases that don't look as good authored poses, but in terms of development work, scales much better with the amount of objects in the virtual environment and these edge cases might be so rare that they barely impact the experience.

Figure 1.7 tries to show the procedural hand posing shown in some of the available footage from the game, but in this case particularly - and in general for the other frame sequences included in this work - we encourage the reader to watch the linked videos and .gifs because a few frames can't possibly do them justice.

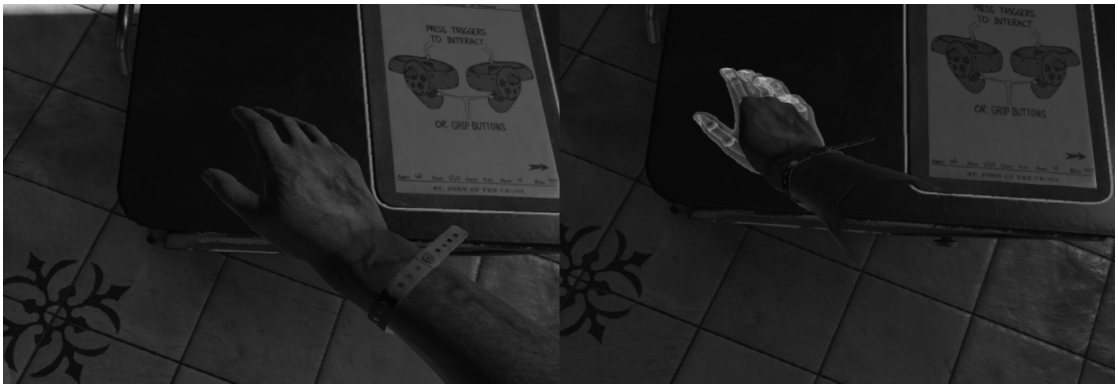


Figure 1.6: Sequence showing the hands in Wilson's Heart entering an immovable object. An X-Ray effect shows the part of the hand inside the object. A gif can be found here: https://tinyurl.com/whPenetration .

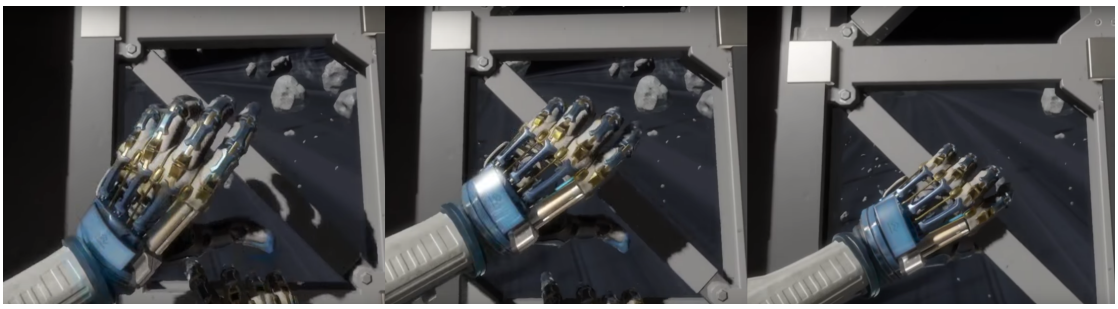


Figure 1.7: Sequence showing the procedural hand posing system in Lone Echo. Captured from video (Ready At Dawn Studios, 2017). The video can be found here: https://tinyurl.com/loneEchoVideo .

1.2 Research Questions

This work explores ways of dealing with the Virtual Constraints Problem previously described: how can we deal with the conflict between wanting the virtual hands to imitate the user's real hands and at the same time respect the virtual environment?

We consider this problem to be specially prominent in two use cases:

1. The user places his real hand in a way that would make the corresponding virtual hand be inside of another virtual solid object.

2. The user wants to grab a virtual object but doesn't position the controller and otherwise produce the input that would in a strict simulation allow their virtual hand to lift the object.

These properties should be pursued without losing the degree of control, consistency and intuitiveness that existing unadjusted hand models provide. In sum, the degree of "hand presence" (term commonly used in the VR industry (Bye, 2016)) achieved by this model cannot be lost in favor of these goals. The question this work addresses is whether "hand presence" can be improved by dealing with the virtual environment constraints problem.

As secondary objectives, we are interested in finding ways of conveying a sense of the weight of virtual objects when interacting with them and improving the tactility of the virtual hands, improving the feeling of touching virtual objects, which are also aspects currently unsatisfactorily addressed.

Therefore, we are trying to reduce the existing gap between the user's real body and their virtual body and compensate for imprecise user input. In terms of the user's perceptual, sensory experience, it might be possible to improve how hands are controlled in VR.

We focus on experimenting with different ways of adjusting the virtual hands' pose, exploring possible adjusted hand models. We will use the Job Simulator hand model, a great example of an unadjusted hand model, as a reference and baseline to compare our experiments with.

2 | Methodology

Our method of research is based on prototyping as a way to gain knowledge about our research topic, in Koskinen et al.'s words:

> *"When researchers actually construct something, they find problems and discover things that would otherwise go unnoticed. These observations unleash wisdom, countering a typical academic tendency to value thinking and discourse over doing."* (Koskinen, Zimmerman, Binder, Redström & Wensveen, 2011)

We use prototypes as "design-thinking enablers", "to discover problems and explore new solution directions" (Lim, Stolterman & Tenenberg, 2008), but also as tools to evaluate and show the potential of different techniques to deal with a specific problem: the Virtual Constraints Problem.

Although we do not prototype games, our main area of interest is games and the ways of controlling hands in VR that we prototype would constitute game mechanics, understood as "methods invoked by agents for interacting with the game world" (Sicart, 2008), in a wide range of VR games.

Our process resembles the experimental game design research approach described by (Waern & Back, 2015). Through the thought-enabling process of developing prototypes, we developed our understanding of an initially less formally defined problem which was originally identified through observation of current VR games.

Using the prototypes we created, we conducted a controlled experiment (Waern & Back, 2015) to compare an approach to hand control found in existing commercial games with an alternative version that partly deals with the problem under scrutiny in order to assess whether the direction we explored in our prototypes shows promise and should be further explored.

Waern et al. warn about common pitfalls of controlled experiments in experimental game design research, but the nature of our area of focus make the controlled experiments approach appropriate. The first of the problems they point out is that when trying to compare variants of a game, having to develop a game for this purpose entails a great overhead, but our work is limited to a very specific mechanic that can be tested with minimal context, so developing a full game was in principle not necessary.

For the same reason, although comparing variants of a core mechanic in a game could indeed be problematic since games are built around their core mechanics and thus, either the other features of the game remain constant, and then the comparison might

be unfair because the other features might work better with one of the variants, or if the other features of the game aren't constant, they can easily become confounding factors in the evaluation.

Finally, results obtained with tests in one single game are not necessarily generalizable. In this regard, we clarify that our intent is not to prove one variant to be superior to another, but to assess whether one of the variants can be useful for VR games and other VR applications where "presence" is regarded as a desirable quality.

Using the gained insight on the problem, we found connections to an existing corpus of research that justifies further research in the direction that was taken and provides us with vocabulary, structure and concepts that can be used to analyze the problem.

3 | Theoretical Analysis

In Chapter 1 the terms "immersion", "presence" (and "hand presence") and even "virtual reality" are used in a loose manner. However, these terms are problematic because of the wide range of fields in which they are used, with very specific, and not always similar meanings. By defining them here, we make them useful as tools to talk about the problems we are dealing with. Here we will frame the Virtual Constraints Problem with existing theories and ground our use of these terms in the academic literature.

3.1 Virtual Reality

Virtual Reality is frequently thought of in terms of a collection of hardware devices commonly associated with the medium: head mounted displays, tracking devices, etc... One such definition focused on the technology is Coates':

> *"Virtual Reality is electronic simulations of environments experienced via head mounted eye goggles and wired clothing enabling the end user to interact in realistic three-dimensional situations."* As cited in (Steuer, 1992).

However, for the purposes of various fields of knowledge such as communication research and for software developers, (Steuer, 1992) argues that the medium needs a definition with an experiential focus instead of a technological one. Steuer works his way from Gibson's definition of presence, "the sense of being in an environment" (as cited in (Steuer, 1992)), through telepresence as "the experience of presence in an environment by means of a communication medium" (Steuer, 1992) and finally arrives at a definition of virtual reality independent of any technological specifics as:

> *A virtual reality is a real or simulated environment in which a perceiver experiences a sense of being in that environment by means of a communication medium.* (Steuer, 1992)

3.2 Immersion

Slater provides a definition of immersion based on "the actions we know to carry out in order to perceive" or Sensorimotor Contingencies (SC), from behavioural and brain

scientist O'Regan and Noë (as cited in (Slater, 2009)). Examples of SCs are turning our head and moving our eyes to look in a certain direction, reaching out with our hand to feel a surface with our sense of touch or turning our head sideways to align one of our ears with the direction that a sound seems to come from in order to hear it more clearly. Note that SCs relate to all of our senses. Slater's definition of immersion is:

> *Immersion is a property of the actions that the user can take that result in changes in perception (Valid Sensorimotor Actions), or changes to the environment (Valid Effectual Actions) within a system.* (Slater, 2009)

This means that immersion is an objective property of virtual reality systems that can be used to compare them. A system can be said to be more immersive than another system if the first system's set of Valid Actions (Valid Sensorimotor Actions + Valid Effectual Actions) is greater than the second system's set.

3.3 Place Illusion and Plausibility Illusion *in lieu* of Presence

In the context of Virtual Reality, the term presence is usually used meaning the "sensation of being in the virtual world" (Schuemie, Van Der Straaten, Krijn & Van Der Mast, 2001). To avoid possible confusion due to the multiple definitions and theories related to the term "presence" (see (Schuemie et al., 2001) for an overview), we will instead use the terms Place Illusion (PI), "the illusion of being in a place in spite of the sure knowledge that you are not there", and Plausibility Illusion (Psi), "the illusion that what is apparently happening is really happening in spite of the knowledge that it is not", proposed in (Slater, 2009).

These definitions are however rather unspecific on their own and require further explanation to be useful. In an interview, Slater describes PI as "the result of using your body to perceive in the way that you would normally" and Psi as "the result of the observed coherence of the world and the extent to which it matches our expectations about how it works" (Slater & Bye, 2015).

Considering only these new descriptions, one might wonder if having a virtual body is in fact connected to PI at all. Without having a virtual body, would it still be possible to perform the same actions with our real body to produce the same changes in our sensory displays? We could turn our real head to look around, but when looking down, we wouldn't see our body. We could reach out with our real hand and feel an object when we touch it, but our hand would not occlude the view of the object we are touching. Our body is a constant in our perception, the actions we perform to perceive the world are always conditioned by the fact that our body is always there. Therefore, we consider that having a virtual body that synchronously follows the movements of

our real body is part of PI insofar as this makes our "virtual senses" perceive it when they normally would in reality. In fact, Slater points at the body as being related to both PI and Psi:

> *"The body is a focal point where PI and Psi are fused. As we have argued the action involved in looking at your own body provides very powerful evidence for PI (your body is in the place you perceive yourself to be). However, this virtual body is not yours, it is a representation of you. In principle you have no control over what it does. Now suppose you move your limbs and you see the limbs of this virtual body move in synchrony. This is a very powerful event in the external world that clearly relates to you – a correlation between proprioception and visual exteroception. Further, it is likely that there would be some degree of ownership over this virtual body – it comes to 'really' seem to be your body (even though you know it cannot be)."* (Slater, 2009)

The connection between the body and Psi that Slater is describing here is subtle. In (Slater, 2009) he points out that a key component of Psi is that events in the virtual environment and over which you have no direct control, refer directly to you. A straightforward example that Slater himself uses is virtual characters acknowledging you and acting in a believable way towards you. But the virtual body you see in the space you occupy is not your own, it is also part of the virtual world and by reacting to your movements it establishes a relation to you that includes you in that virtual world. Since you are real, your inclusion in the virtual world makes it more real.

Interestingly, Slater doesn't explicitly relate the physics of the virtual world with either PI or Psi. We would argue though, that because of our knowledge of how real physical objects work, our expectation will be that the objects that we see in virtual reality behave the same way, therefore we consider physics to be related to Psi. This includes how our virtual body should interact with the virtual environment. A dissonance between the way the real world works and the way the virtual world works would remind us of the irreality of the experience, which would be a break of Psi. If the world we perceive doesn't work like the real world, it cannot be the real world.

We should emphasize that both PI and Psi are subjective experiences. In the same virtual reality, a user that only looks around in a virtual environment will have a very different degree of PI than one that moves around a lot and runs into a real wall, tries to touch a virtual object and realizes they can't feel it with their sense of touch... In regards to Psi, the user's expectations of the world will be different based on their knowledge of it. For instance, a chemist will have very different expectations about a virtual chemistry lab than a computer engineer.

Slater's work is not developed specifically for VR games. His research is concerned with how to elicit realistic behaviors in virtual situations, which is extremely relevant

in order to use VR for therapeutic purposes or to explore moral issues, among other things. The degree of PI and Psi that games might be interested in producing might be different. Some genres, like perhaps horror games, might benefit from having realistic virtual environments that produce a high degree of Psi, while other genres might have no interest in such realism and in those cases, the effort will have to be in producing a suspension of disbelief (Ferri, 2007) in the player and maintaining a coherent world that matches the expectations set by the tone of the game.

3.4 Sense of Embodiment

In dealing with virtual hands, we are considering what connection one might experience towards a virtual body. (Kilteni, Groten & Slater, 2012) propose the notion of Sense of Embodiment (SoE) to address the issue of "whether it is possible to experience the same sensations towards a virtual body as toward a biological body":

> *"SoE toward a body B is the sense that emerges when B's properties are processed as if they were the properties of one's own biological body."*

Kilteni et al. further explain SoE in terms of three subcomponents:

- *Sense of Self-Location: "one's spacial experience of being inside a body."*
- *Sense of Agency: "the sense of having global motor control, including the subjective experience of action, control, intention, motor selection and the consious experience of will"* (Blanke and Metzinger's definition, as cited in (Kilteni et al., 2012)*, resulting from the comparison between the predicted sensory consequences of one's actions and the actual sensory consequences.*
- *Sense of Body Ownership: "one's self attribution of the body, [...] implying the sense that the body is the source of the experienced sensations."*

The Sense of Self-Location in the virtual body should be produced when the virtual body synchronously aligns with the real body. Kilteni et al. indicate that self-location should not be confused with the spatial experience of being inside a world, unlike Place Illusion, the Sense of Self-Location cannot happen at all without a virtual body. If the virtual body executes our intentions (most likely expressed through the movements of our real body interpreted in the context of the virtual world), we will feel a Sense of Agency towards the virtual body. Finally, the Sense of Body Ownership is correlated with the previous two, but also results from the suppression of the stimuli from the

real world (Schubert, Friedmann & Regenbrecht, 1999) and the synchronization of the virtual stimuli with the state of the virtual body, in other words, if the sensations that the user experiences happen when the interaction of the virtual body with its environment would produce them.

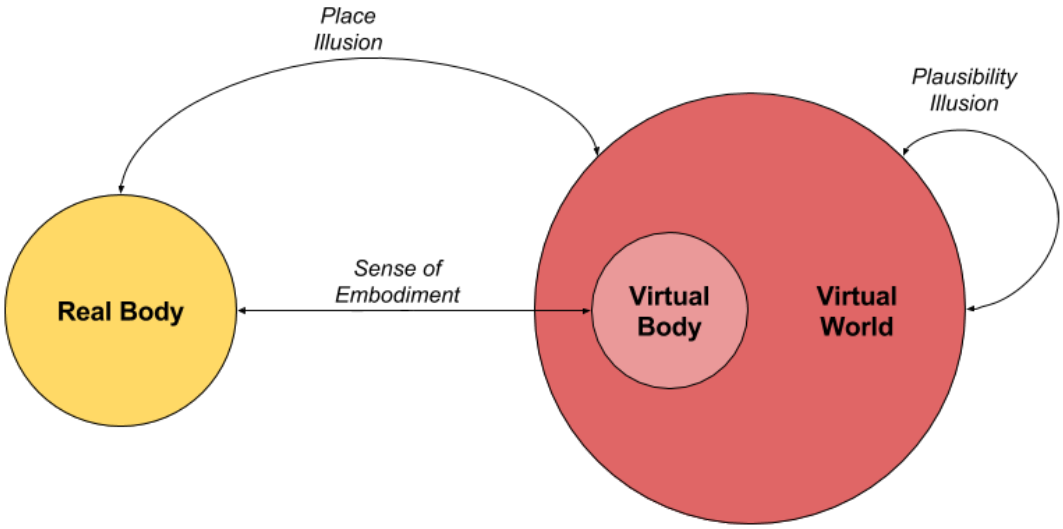


Figure 3.1: Summary of the relationships between real body, virtual body and virtual world defined by PI, Psi and SoE. The virtual Body is understood to be part of the virtual world. PI has to do with how the sensory displays update, conveying information about the virtual world, with the movements of the real body. Psi can be understood as the result of the user's evaluation of the virtual world's coherence and the interaction between its elements. SoE has to do with the relationship between the real and the virtual body. It is produced by the perceived spatial alignment between the virtual body and the real body (Sense of Self-Location), by how the real body controls the virtual body (Sense of Agency) and by whether stimuli are perceived to be correlated to the virtual body or the real body (Sense of Body Ownership).

3.5 Sensory Perspective

In real life, our body determines the "perspective" of our senses, we sense from our body: what we see is restricted by the position of our eyes, we can only feel an object with our touch if we can reach it with our hands. In Virtual Reality one has the option to restrict our senses either with the real body or with the virtual body.

An example focused on the sense of sight to illustrate the difference: Imagine the case where a user moves their real head to a position that corresponds to somewhere inside a virtual object. If the virtual body is constrained by the virtual environment, it will separate from the real body at the surface of the virtual object. At this point, if the sense perspective is restricted to the virtual body, the user will see from the perspective of the virtual body, but if the sense perspective is restricted by the real body, the user will see from outside of the virtual body, seeing instead from where their real body is: the inside of the virtual object.

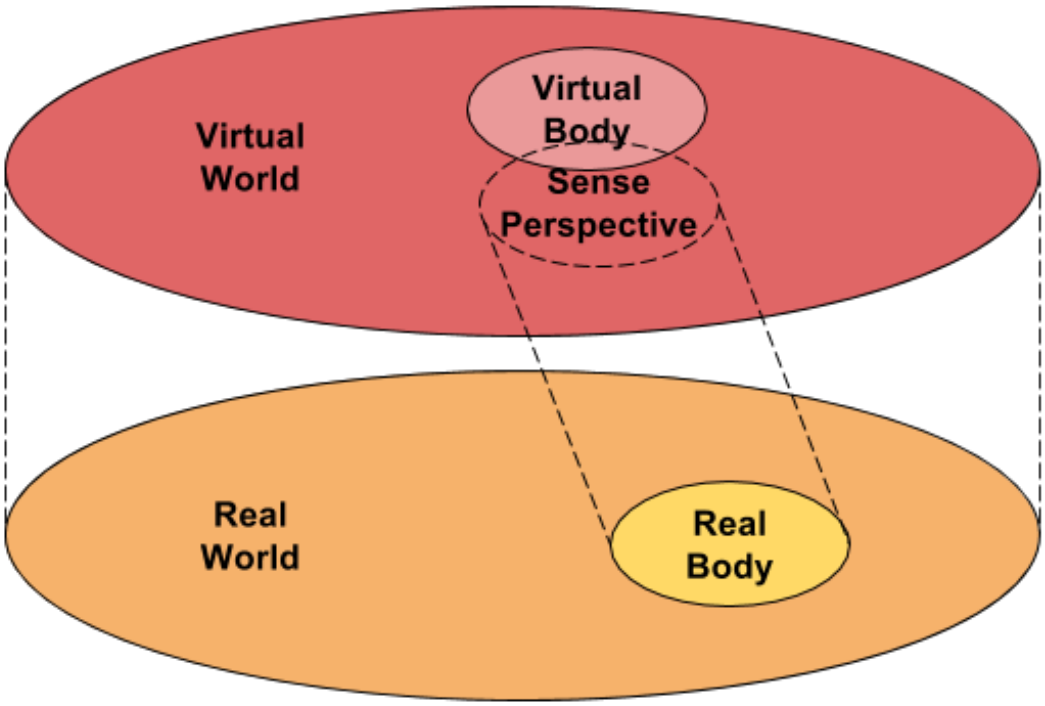


Figure 3.2: The 3D diagram represents the virtual world overlayed on the real world. The sense perspective need not be restricted to the real body. This can be a design choice - as described in the text - or a limitation of the system (e.g. insufficient or imprecise tracking). Likewise, the sense perspective could be made to not coincide with the perspective of the virtual body in cases when the body is restricted by virtual obstacles that make it not be able to align with the real body.

The first behavior is the most coherent from the virtual world's side and could benefit the Sense of Ownership of the virtual body because it connects the virtual senses of the user to it instead of to their real body. At the same time, the Sense of

Self-Location would be challenged by the conflicting signals between our proprioception and our sight.

The second behavior could produce a break in Psi because seeing from inside of a solid should not match anyone's expectations about the worlds possibilities. It could also be argued that it would impact PI negatively, since from that situation we might be able to see our body from outside of it, which is not something our normal SCs would allow.

We would remiss if we didn't mention that at least in regards to the sense of sight, the first behavior might fall under what is generally considered a bad practice by the VR software development community. Specifically, it is discouraged to move the camera in any way that doesn't match the user's head movements because it is thought to produce physical discomfort in the user (Unity Technologies, 2015; Oculus VR, n.d.). This would be an issue that would have to be considered in much greater detail if the focus of this work wasn't only on the control of the hands.

3.6 The Virtual Constraints Problem

We now seek to describe the problem that we are addressing in this work, that is, the Virtual Constraints Problem, using the concepts that we have just defined. Furthermore, we shall use the reviewed theory and existing related research to argue why the adjusted hand control model is an approach worth exploring.

The Virtual Constraints Problem has two conflicting goals:

1. Make the virtual body respect the virtual environment physics.

2. Preserve the connection between the user's real body and the virtual body.

The first goal is mainly related to Psi. As previously discussed, the user knows how real bodies interact with the environment and if their virtual body does not interact with the environment like they know real bodies do, this dissonance will remind the user of the irreality of the virtual experience. If the world we are perceiving doesn't work like the real world, the world we are perceiving cannot be the real world, thus what is apparently happening is not really happening, which implies a break of Psi.

The second goal has to do with the Sense of Self-Location, the Sense of Agency and PI. If the virtual body separates from the real body, the user might stop perceiving themselves to be occupying the space where the virtual body is, which implies losses in Sense of Self-Location in the virtual body. In the case of Sense of Agency, the loss would happen if the separation of the virtual body from the real body happens in a way that makes the user stop feeling in control of the virtual body. Lastly, the effect on PI is harder to anticipate. When the real and virtual bodies have a different pose, the movements that the real body can do to alter the sensory displays are different from

the ones that the virtual body should be able to do (assuming that the virtual body is restricted as if it had the same bone structure, etc... as real bodies) and in this way, PI might be reduced.

We can see that the conflicting goals of the Virtual Constraints Problem bring to light conflicting requirements of Psi and some of the components of SoE and even possibly PI.

3.6.1 The Rubber Hand Illusion

Considering Kilteni et al.'s explanation of SoE, one would think that using adjusted hand control models would make SoE difficult. In terms of the aforementioned components of SoE, one might expect that separating the virtual hand from the real hand would have a direct negative impact on the Sense of Self-Location because the virtual hand is not where the user feels their own hand to be through proprioception. It would likely also affect the Sense of Agency, by making the virtual hand not follow the user's movements exactly, we are taking part of the control away from them in comparison with a 1 to 1 control method. Thirdly, in regards to the Sense of Ownership, it would be harder for the user to believe that the virtual hand is theirs if they don't see it where they feel their own hand to be.

The well-known Rubber Hand Illusion, originally reported in (Botvinick & Cohen, 1998), and other related experiments suggest that these issues might not be a problem after all. Experiments demonstrating this illusion typically have the following setup: the test subject places their right arm behind a panel that hides it from their view, a fake rubber arm is placed within their view, aligned with their real arm. The fake and real hands are synchronously stroked with paintbrushes in the same way for a period of time. When the subjects are asked to point towards their right arm with their eyes closed, typically they point toward the rubber arm - effect known as proprioceptive drift. In questionnaires after the experiment, subject reports showed a degree of attribution of the rubber hand (Botvinick & Cohen, 1998) and when the rubber hand was injured subjects reacted as in anticipation of pain (Armel & Ramachandran, 2003). The reported attribution of the fake hand and the anticipation of pain signify a Sense of Ownership towards the fake hand.

(Sanchez-Vives, Spanlang, Frisoli, Bergamasco & Slater, 2010) replicated the illusion with a virtual reality setup by making the virtual hand follow the movements of the subject's real hand, instead of using the sense of touch to induce the illusion, and found both proprioceptive drift and a degree of attribution of the virtual hand. These illusions map with the concepts of Sense of Place and Sense of Ownership that we pursue with our virtual hands.

(Fourneret & Jeannerod, 1998) reports on experiments where subjects were asked to draw a straight line between to points shown on a screen by drawing with a stylus without seeing the drawing hand. While the subjects drew, a random bias was introduced to the

result and the subjects automatically corrected the error and were largely unconscious of the deviation from their actual drawing. We take this lack of conscious monitoring of motor performance to indicate that the Sense of Agency might not be lost when the virtual hand's movements show relatively small deviations from the actual movements of the real hand.

(Kilteni et al., 2012)'s components of Sense of Embodiment seem to be found in the the Rubber Hand Illusion family of experiments and their similarity to the virtual hand - real hand separation produced by adjusted hand control models suggests that the results are transferable to our problem.

3.7 Critiques of Movement Interaction

Norman rejects the claim that current gestural interfaces are Natural User Interfaces (Norman, 2010): "user interfaces where the interaction is direct and consistent with our 'natural' behaviour" (Hvas Mortensen, 2017). Norman reminds us that the strength of a user interface is determined by how easy it is to remember the actions that are possible and how to invoke them. He points out that due to the ephemeral nature of gestures, when a gesture does not produce any response or produces an undesired one, the user lacks the necessary feedback in order to be able to learn to perform the gesture 'correctly'.

Gillies argued in (Gillies & Kleinsmith, 2014) that if movement interfaces are non-representational, that is, if they implement the Sensorimotor Contingencies that we know from the real world, Norman's arguments might not apply. In (Gillies, 2016) however, he acknowledges that when the SCs implemented in the interfaces are only similar to the real-world SCs - as is often the case - they easily become representational again and Norman's arguments regain their weight. An example of this provided by Gillies in the context of VR is "walking in place" as a locomotion control method.

In regards to unadjusted hand control models, since the mapping between the real movements of the user and the virtual movements is 1 to 1, these arguments need not be a concern, but this is less clear when it comes to adjusted hand control models because the virtual hands can separate from the user's real hands, which could be thought to make the control less intuitive. However, we would argue that because adjusted hand control models can still provide a sensorimotor feedback loop (Gillies & Kleinsmith, 2014) that matches our knowledge about how real bodies are restricted in their movement by physical objects around them, the issues that have been pointed out should not be a problem.

In a broad sense, the tracked controllers that the users hold in their real hands and use to control the virtual hands are part of the interface. In Chapter 1 we covered the differences between the controllers of the HTC Vive, the Oculus and the PS VR and we noted that the Oculus Touch controllers were the ones that had the most input

mechanisms that could be used to control virtual fingers with movements of the real fingers. However, the user has to perform reduced or exaggerated movements with their real fingers that don't exactly correspond to the movements of the virtual fingers. in the case of the triggers, which have analogue input, a continuous mapping between the movement of the real fingers and the virtual fingers is possible, which allows the sensorimotor feedback loop mentioned by Gillies, but in the case of the capacitive sensors, the user doesn't have a way of feeling that their fingers are about the trigger a state change until the state has triggered. These problems would of course not exist with the use of data gloves like the Manus VR Gloves since the gloves can track the movement of the fingers completely.

On the software side, both Facebook Spaces (Facebook, 2017) and Oculus First Contact (Oculus VR, 2016a), which use and showcase the possibilities of Oculus Touch controllers, use representational movement interaction. Buttons, both when they are diegetic and when they are part of a GUI, can only be pressed with the index finger, and only when it is extended. The virtual hands only collide with virtual physical objects in certain poses, like with a closed fist. Because the movement control in these games is more representational, it requires more attention from the user and even simple hand gestures can become rather complicated when thought of in terms of combinations of input. The problem is that we want to minimize the amount of attentional resources that the user needs to focus on the controllers because they are part of the real world and not of the virtual reality, which is where we want the users to have their attention. Although we prefer to use PI and Psi instead of presence, it is worth noting that Witmer and Singer relate presence to attention allocation and defined presence as partly determined by involvement, "a psychological state experienced as a consequence of focusing one's attention on a coherent set of stimuli or related activities and events" (Schuemie et al., 2001).

3.8 *Encore*: Current VR Games

Armed with a theoretical framework, we can revisit the games discussed in Chapter 1 and evaluate them in terms of PI, Psi and SoE, mostly focusing on the hands[1]. The reader should bear in mind that these are subjective experiences and that what we present here are evaluations based on a combination of our own individual impressions and rational analysis. In order to obtain more reliable assessments, one should conduct tests with larger groups of subjects and use perhaps some of the existing questionnaires, and tests that measure physiological and behavioral responses.

Job Simulator is our example of an unadjusted hand control model. It produces high PI and SoE because its 1 to 1 mapping of the hands to the controllers means that

[1]Lone Echo was discussed in Chapter 1 but will not be discussed here because it hasn't been released yet and we are unable to test it ourselves.

the movements that the user wants to do with their virtual hands are exactly the same as the ones that they have to do with their real hands (within the immersion limits of the hardware system). Where one might take issue with Job Simulator's hands is in regards to Psi. The hands can penetrate immovable virtual objects that the user will easily recognize from real life and expect to not allow their hands to go through (recall Figure 1.3 in Chapter 1). For analogous reasons, the fact that the hands disappear when holding objects in Job Simulator can also produce breaks in Psi.

It should be noted however, that the experience of playing Job Simulator is hardly objectionable even though from a rational perspective its world doesn't work like the real world. Its cartoonish, low fidelity visual aesthetics and humor elicit the required level of suspension of disbelief and set the right expectations in the user (see (Nowak & Biocca, 2003; Argelaguet, Hoyet, Trico & Lécuyer, 2016) on how the visual fidelity level can influence user's expectations and affect their experience in VR). In fact, the game frequently surprises players with the level of detail in its world in terms of interactibility. (Schwartz & Reimer, 2017) tell that a large part of the development time was spent adding the interactions that testers expected the virtual world to allow.

In connection to this, it seems appropriate to mention the high correlation found by (Schubert et al., 1999) between their interpretation of presence as embodied presence, which "develops from the representation of navigation (movement) of the own body (or body parts) as possible action in the virtual world", and preditibility ("the ability to predict and anticipate what will happen next") and explorability ("the possibility to explore and actively search the virtual environment").

Wilson's Heart's hands don't quite fit in the adjusted hand control model, but the way they work provides useful information that is relevant for the Virtual Constraints Problem. As discussed in Chapter 1, in this case the virtual hands follow the real user's hands without separating from them until an interaction with a virtual object is triggered, generally by input produced by the user's real fingers using the analogue triggers or the capacitive sensors in the Oculus Touch. When said interactions happen, the hands snap to a pose that is predefined for each object. In most cases, these transitions from following the tracking devices exactly, to largely ignoring the tracking data are quite natural and the separation wasn't noticeable, so in terms of Sense of Self-Location this works well. In terms of Sense of Agency, however, the fact that the interactions with virtual objects were mostly limited to the predefined poses for each object did have a negative effect. While there are no interactions that override the user input, the Sense of Self-Location and specially the Sense of Agency and Sense of Body Ownership, are enhanced by the amount of control over the fingers that the Oculus Touch controllers allow.

The physical interaction between the virtual hands and the environment is - aside from the predefined interactions - far more limited than in Job Simulator: the hands go through many objects or have to be in specific finger poses to be able to physically

impact the objects (similarly to what was described in 3.7), which works against producing Psi because the user is constantly reminded about these differences with the real world. This is aggravated by the fact that Wilson's Heart, although restricted in colors to greyscale, has a much more realistic aesthetic. In the same way that having a low fidelity style worked in favor of Job Simulator, this could be working against Wilson's Heart.

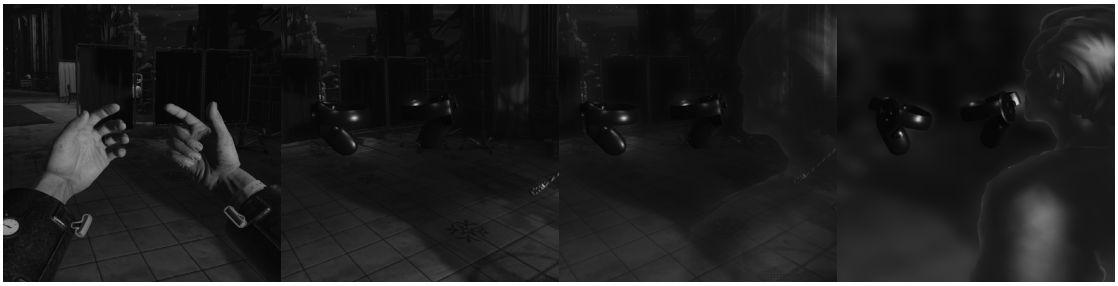


Figure 3.3: Sequence showing an "out of body experience" in Wilson's Heart: the sense perspective is not restricted by the virtual body. A gif can be found here: https://tinyurl.com/whOutOfBody .

Wilson's Heart provides a great example of sense perspective restricted to the real body instead of the virtual body. If the player moves too far from the virtual body, they will be disembodied, the world becomes blurry, the virtual body turns transparent and the Oculus Touch controllers become visible. This approach opts to break the SoE more abruptly in order to clearly establish the limits of what the player is allowed to do.

4 | Experimental Analysis

This chapter delves into the development of five prototypes, each portraying a different type of behavior for hands in VR. These different prototypes approach the Virtual Constraints Problem in different ways and to different degrees. The implemented grabbing system, visualisation of the real hand and our use of haptic feedback will also be detailed below. The following sections will present these topics, including a categorization of approaches that can be taken when implementing hands in VR, how our prototypes have been implemented and discussion weighing the different options based on their pros and cons. The implementation of our baseline hand (based on the hands from Job Simulator) will also be detailed in this chapter.

All five prototypes have been developed using the Unity engine and the HTC Vive with default controllers for input. The space we've explored during the development of the prototypes has been both enabled and restricted by these choices. Unity has a set of APIs which we as users of Unity have available to us. These APIs have shaped what has been possible for us during implementation of the prototypes[1]. Furthermore, building the prototypes around using the HTC Vive's controllers as the input method makes our methods and approaches more applicable to hardware that has the same affordances as these. As for the input used to control the hands the position and orientation input is gathered from tracking the controllers and the controllers' trigger buttons are used to indicate how much the fingers should bend, referred to as the closed value (how closed the hand should be)[2].

These three inputs are the base for different categories of approaches. Each of these inputs can be filtered, by which is meant that the input can be modified or ignored. Filtering one of the inputs can be seen as using that input as the parameter for a function, which returns a new result. A hand can have a filter for either none of the inputs (Unadjusted Hand Control Models) or one of the inputs or more (Adjusted Hand Control Models). Different filtering combinations will give the hand a different behavior in the virtual world and might result in the hand seeming more realistic when interacting with its environment.

The five hand prototypes are named as follows: *Rigid Hand*, *Sliding Rigid Hand*, *Finger Rigid Hand*, *Physics Hand* and *Rotation Hand*. They differ in the way they filter

[1] Unity's scripting reference is available online (Unity Technologies, 2017b).
[2] The closed value goes from 0 (open) to 1 (closed).

the player's inputs which gives them a different feel. The detail of which filters they use and their implementation will be given in the sections below.

4.1 Categorization of Approaches

The three player inputs mentioned above (position, orientation and how closed the hand should be) are the base of the categorization of approaches. Each of these inputs can be filtered, by which is meant that the input can be modified or ignored. Filtering one of the inputs can be seen as using the input as the parameter for a function, which returns a new result which will be used instead of that input. A hand can be implemented to filter several inputs at once and can have different filters depending on the context. Different filter combinations will give a different behavior for the hand in the virtual world and can result in the hand seeming more realistic when interacting with its environment. The hand prototypes will be mentioned in the sections about the three filter variables below, but their implementation will be described in detail in Section 4.2.

Position	Rotation	Finger position
X		
	X	
		X
X	X	
X		X
	X	X
X	X	X

Table 4.1: Filter variable combinations.

4.1.1 Position Filtering

A position filter using the definition above is a function, which when enabled, takes the player's positional input from a controller and returns a new value to be used in its stead. This means that the position of the hand in the virtual world will deviate from the controller position.

One of our hypotheses is that it's possible to retain or increase the degree of control, consistency and intuitiveness seen with unadjusted hands while simulating a more realistic behavior for hands around objects in the world. One of the first steps here is to not allow the hands to penetrate objects. By using position filtering to deviate the hand from the controller position when trying to penetrate an object, the hand can interact more realistically with the world.

The following two figures (Figure 4.1 and Figure 4.2) show an image sequence of a hand with no input filtering and an image sequence with a hand that uses position filtering. The second figure shows the hand stopping when it reaches the object, whereas in the first figure the hand moves through the object.

Figure 4.1: Sequence showing hand without filters entering obstacle. A gif can be found here: https://tinyurl.com/FiltersNone .

Figure 4.2: Sequence showing hand filtering the position. Notice that while the hand stays the controller continues to move into the obstacle. A gif can be found here: https://tinyurl.com/FiltersPosition .

Implementing position filtering for hands is about determining how to make the hand follow the controller. All of our five prototypes use position filtering, but the implementations of the filters differ. The first distinction is in the degree to which they rely on the physics engine. The Physics Hand is the only one of our prototypes that relies heavily on the physics engine. The way the Physics Hand is moved towards the controller is to set the velocity each frame so that the hand will reach the controller within the next frame using the physics system. Using velocities and the physics system to move the hand has several positive aspects. The main gain of using the physics system to move is that it adheres to the rules of the system, including collisions, which means that the hand will not penetrate objects in the world that have colliders. One of the downsides of using the physics system to handle collisions and more is that we as the developers have no direct control over the inner workings of the system and therefore lose some control of how the hand behaves and feels.

As for the prototypes that are implemented, which rely less on the physics engine, their filters are all based on three different approaches to stop the hands from penetrating objects: Sweep and place, skip movement in direction and depenetration. These three methods fall into two groups; preventive methods and corrective methods. The first two methods fall into the preventive group, because they try to avoid penetration of objects before it happens, whereas depenetration, which falls into the corrective methods group, will correct the penetration after it has happened. In the different prototypes these three methods are used alone or in combination to create the behavior wanted from that prototype. The simplest of these prototypes is the Rigid Hand. Here only depenetration comes into play. Depenetrating the hand from an object means finding the direction and distance in which to move the colliders of the hand in order for the hand and the object not to be overlapping anymore. The direction and distance are found by calculating the minimal translation required to separate the hand from the object. When moving the hands in this prototype, their position is simply set to the

target position on the controller each frame after which they are depenetrated from any objects they would be overlapping. Since the hand is always depenetrated to the closest surface it will jump between surfaces, when the controller is moved around inside immovable objects. If the hand is closer to one surface in one frame and another surface in the next, depenetrating the hand from the object will lead to the hand switching surfaces, creating a jump.

The problem of jumping between surfaces when using depenetration is what the Sliding Rigid Hand attempts to solve. It employs the other two methods, sweep and place and skip movement in direction, in order to combat the jumping . The main idea behind this prototype is to try avoiding penetrating an object altogether or at least to only penetrate a small amount so that depenetrating the hand would always select the surface from where it entered the object. Sweeping is about calculating how far in a direction the hand can move before it hits something. Before moving the hand to the controller position we can sweep and see if it would hit an object on the way. If an object was hit then the hand can be placed there instead of placing it at the controller. This way the hand will not penetrate the object although the controller is inside or on the opposite side of it. In Unity sweeping in a direction will not detect an object if the rigidbody, which does the sweeping, already touches the object. Therefore, in subsequent frames after placing the object on the surface, another method is needed for the hand to not penetrate the object. Skipping movement in a direction is one way to alleviate this problem. When the hand is touching the surface of an object, it shouldn't move in the direction towards the surface, because that would lead the hand to penetrate the object. Therefore, we skip all movement along that direction, essentially allowing the hand to only move on a plane. Moving on a plane works wonders when interacting with cubes, but has its downsides when dealing with objects with other shapes. Sliding on a plane might lead the hand to penetrate parts of an object that point out or penetrate other objects. Depenetration helps correct the hands in the situations where sliding over the surface of one object led them to penetrate another object. These three methods combined should lead to hands that don't penetrate objects, but also don't jump between surfaces due to depenetration.

The two remaining prototypes, the Finger Rigid Hand and the Rotation Hand, both use the methods mentioned above to achieve their position filtering. The Finger Rigid Hand uses the exact same position filtering method as the Rigid Hand and the Rotation Hand uses a filter similar to the Sliding Rigid Hand with the exception that it doesn't do any depenetration.

4.1.2 Rotation Filtering

To use a rotation filter is to deviate the hand's rotation from the current orientation of the controller. In certain contexts it can be beneficial to filter rotation for the hand to seem more alive and realistic. Hands can adapt their rotation in several ways and

different approaches might be taken depending on the context. When approaching a wall with their hand a player's intention in the real world might be to place their hand flat on the wall (palm first). This behavior, where the rotation is adjusted to make interacting with walls more natural, is one of the use cases of rotation filters.

Another reason to use rotation filtering is to avoid penetrating objects. Besides being an example of how the hands could feel more natural when approaching walls, the reason for rotating the hand to be palm first when getting close to the surface could also be to avoid penetrating the wall. When the hand rotates, the fingers are rotated away from the wall as well, letting the hand get closer to the wall before it starts penetrating it. Using rotation filtering as a means of avoidance can also reduce the amount of position filtering needed, by allowing the hand to get closer to the controller position, while still being constrained by objects in the world. Rotation filtering is also a big part of pose snapping, which was mentioned Section 1.1. Pose-snapping can be used to show available interactions and how to perform them. An example of pose-snapping being used to indicate how to perform an interaction can be seen in Figure 1.5 from Wilson's Heart, where the hands snap to a pose when the player presses a trigger near the suction cups attached to the player's head. Pose snapping of course works much better in unison with both finger position filtering and position filtering. None of our prototypes have explored pose snapping.

The first of the two figures below (Figure 4.3) shows an image sequence of a hand using only rotation filtering and the second figure (Figure 4.4) shows an image sequence of a hand using both position and rotation filtering where the rotation filtering is implemented as in the example from above, palm-first towards the surface. It should be noticeable in the last frame of the first sequence that the hand has entered the obstacle. Using rotation filtering allowed the hand to move closer to the surface before it started penetrating it, but it doesn't stop it completely. Combining the rotation filter with a position filter has the effect of completely stopping it from penetrating the obstacle, while having a more natural look and allowing the hand to get closer to its target.

Figure 4.3: Sequence showing hand filtering the rotation. In the last frame the hand has penetrated the obstacle. A gif can be found here: https://tinyurl.com/FiltersRotation .

Figure 4.4: Sequence showing hand filtering both position and rotation. In the last frame the hand stays on the surface. A gif can be found here: https://tinyurl.com/FiltersPositionRotation .

Although all our hand prototypes use position filters, only two of our five hand prototypes (Physics Hand and Rotation Hand) use rotation filtering.

For the physics hand, we don't use any explicit rotation filtering. The physics system decides the rotation of the hand when interacting with surfaces. While the controller is within an object, the physics simulation can determine to rotate the hand to a flat position (either facing the surface with the palm or with the back of the hand) in order to reduce the position deviation between the hand and the controller. Due to the velocities used to move and rotate the Physics Hand being high the rotation happens as a jump and not as a smooth movement towards the surface resulting in a less natural feel during the jump. To improve upon this, explicit rotation filtering could be added to smoothly rotate the hand to lay flat on the surface, which might remove the jump altogether, but at the same time introduce complexity to the implementation.

Contrary to the Physics Hand, the Rotation Hand uses explicit rotation filtering. The goal of the rotation filtering for this hand is to always approach a surface palm-first. The main example to explain this choice is when the hand approaches a wall. When a player's hand approaches a wall we assume that the most likely desired interaction is to place the palm of the hand against the wall. Having an open hand and moving the hand fingers-first wouldn't make much sense. It would not give support if the reason for touching the wall was to lean against it, for instance. With this as the main idea behind the rotation, the implementation then had to support this in several cases. The easy case is when the player approaches the wall with an open hand and the palm facing the surface and the back of the hand pointing away from it. We use the distance to the surface to blend between the orientation defined by the controller and the desired palm-first pose. Another case is what to do when the hand is closed into a fist. In this case, we assume that the most likely desired interaction is to punch the object, which means that the palm shouldn't be facing the surface. When the hand is closed into a fist it shouldn't rotate to face the object palm-first but rotate to let the fist face the surface of the object.

4.1.3 Finger Position Filtering

Above is mentioned that the player can control the fingers of the hand by pressing the trigger button on the controller. The hand prototypes all have animations that allow them to be either open or closed into a fist. When filtering the finger positions the target positions of the fingers, or where we want to place the fingers, deviates from the location indicated by the closed value of the hand.

There are two main reasons for wanting to filter the finger positions. The first reason is relatable to the reason to filter the position; we want the fingers and the hand in general to avoid penetrating objects. By bending the fingers when approaching the surface of an object the hand can move closer to the surface before it starts penetrating. In reality, this would be much less of an effort than to move or rotate the hand or arm and might therefore seem like a natural way to interact. Furthermore, the fingers might also bend individually, which can be especially noticeable around edges of objects. Our Finger Rigid Hand and Rotation Hand uses finger position filtering to avoid penetrating objects. The other reason for wanting to filter finger positions is about indicating whether an object hovered over is interactive or not. In Job Simulator, the fingers bend slightly when the hand is within range of an object that can be grabbed, but they all bend the same amount, though, resulting in a different feel from when the fingers are bend individually. The fingers can of course be filtered individually, which can put the hand in a pose that looks like the hand could be stroking the surface or getting a grip of the object. Moving the fingers individually makes the hand look much more realistic in its interaction with objects. This approach has been explored with the Physics Hand, which filters finger positions to show when the hand is hovering over a grabbable object[3] and is close enough to grab it as well. Furthermore, by bending the fingers towards the surface of the grabbable object when approaching it, the Physics Hand allows the player to do things like petting or stroking across object surfaces in a way that looks more natural than with a completely open hand, as can be seen in Figure 4.6.

[3]A grabbable object is an object which has our grabbable component attached to it. The grabbable component will be described in further detail in section 4.5 where we cover our grabbing system.

Figure 4.5: Sequence showing hand filtering both position and fingers. A gif can be found here: https://tinyurl.com/FiltersPositionFingers .

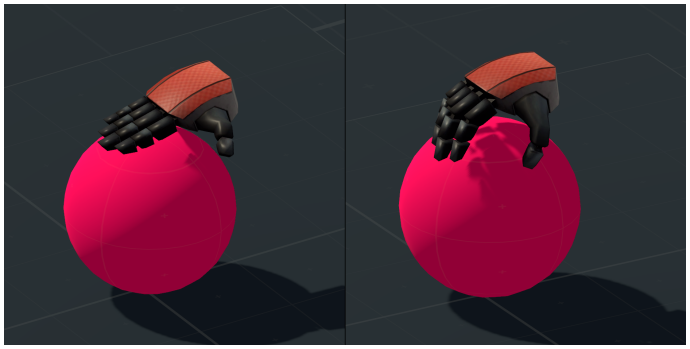


Figure 4.6: Example of petting or stroking object. To the left without and to the right with finger filtering.

The three hand prototypes that implement finger filtering (the Finger Rigid Hand, the Physics Hand and the Rotation Hand) do so in two different ways. The method used for the Finger Rigid Hand tries to find a suitable closed value for each finger. Each finger can then be animated using this value and its corresponding animation mask. Animation masks are used to isolate a part of the animation data, which makes it possible to use the same animation as for the other hands, while being able to animate each finger individually. The other two hand prototypes make use of an inverse kinematics system (or IK system). Inverse kinematics make use of the kinematics equations to calculate positions for a joint system where we have desired positions for the end effectors. This maps nicely to the movement of fingers, since a finger is a system of joints, where we know the root position of the finger and we'll set the desired location of the end effector: the tip of the finger. The IK system can then calculate how to bend each of the finger's joints to make the tip reach the desired position. To make sure the fingers bend naturally, restrictions can be set for how the joints can bend. The IK system will try to get the tip of the fingers as close to their desired locations

as possible without violating the restrictions for the joints. The restrictions for the fingers are set up in such a way that the joints can only bend around a single axis which means that they cannot bend sideways, for instance. The axis around which they bend is chosen such that bending the fingers closes the hand into a fist. All fingers bend around the same axis, except for the thumb, because of its placement on the hand. We have used a Unity asset pack called Final IK (RootMotion, 2017), which implements several components and methods that can be used to setup and run IK calculations. To move the fingers using the IK system, we move the IK targets, which are the desired locations for the fingertips.

4.2 The Hand Prototypes

This section will go into depth about how each of the five hand prototypes have been implemented, discuss the pros and cons of the different approaches and finally a step-by-step process for creating the behavior of the prototype will be presented. The table below (4.2) stands as a reminder of which filters each of the hand prototypes implements.

	Rigid Hand	Sliding Rigid Hand	Finger Rigid Hand	Physics Hand	Rotation Hand
Position filter	X	X	X	X	X
Rotation filter				X	X
Finger position filter			X	X	X

Table 4.2: The hand prototypes and the variables they filter.

The hands all have a couple of things in common. First, they require that colliders are set up for their models. We use one box collider for the palm of the hand and one for each part of the fingers, except for the Rotation Hand, which only has colliders on the fingertips and the palm. In Unity we made all game objects in the hand be in the same layer (but not the default layer) and made the physics system ignore interactions between objects within that layer. For the physics hands this is taken one step further where each hand, left and right, has their own layer. The second thing we've used for all hands is Unity's rigidbody component. This component lets the hands interact with the world through the physics system. In Unity a rigidbody component can be kinematic or non-kinematic. A kinematic rigidbody is not affected by other physics objects which

means it's not affected by collisions, for instance. A non-kinematic rigidbody is affected by other physics objects and by gravity and its motion will therefore be influenced by collisions. All of our hands use kinematic rigidbodies with the exception of the physics hands. The last thing the hands all need is an animation to close the hand. Our animations consist of two frames; one where the hand is fully open and one where the hand is closed. We can blend between the two frames using the trigger value from the controller, which therefore controls how closed the hand is. Some of the hands have different considerations depending on whether the hand is open or closed. For all hands we consider the hand closed when the trigger value - which can be between 0 (not pressed) and 1 (fully pressed) - is equal to or greater than 0.5.

The placement of the hand compared to the controller is a decision that impacts how the user will move the controller around to interact with objects and how the hand feels to use. We chose to place the target of the hand compared to the controller in the virtual world in the same way the user's real hand is placed according to the controller in the real world. As shown in Figure 4.7, the right hand is placed to the right of the controller in a position that could grab and hold the controller. The left hand goes on the left side of the controller but is otherwise the same as the right hand. Our reasoning behind this choice was to place the hand as closely as possible to the position of the user's hands in the real world. From now on, when the controller's position and orientation are mentioned, it's the position and orientation the hands displayed in Figure 4.7 that we're referring to.

Figure 4.7: Two images showing the placement of the hands compared to the controllers.

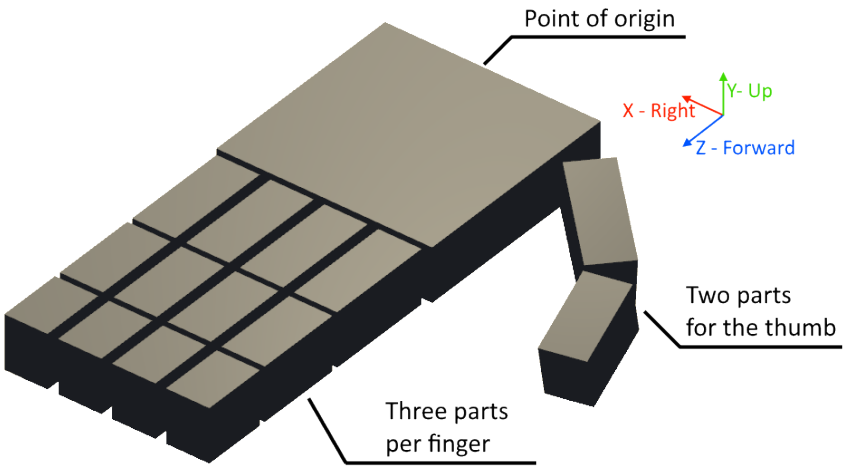


Figure 4.8: Diagram showing the setup of the hand.

4.2.1 The Rigid Hand

As mentioned above in section 4.1.1, the Rigid Hand moves by setting the rigidbody's position and rotation to the selected position and rotation on the controller. This is done using the $Rigidbody.MovePosition$ and $Rigidbody.MoveRotation$ methods exposed by Unity. Using these methods instead of setting the position and rotation properties directly is the prescribed way of moving and rotating non-kinematic rigidbodies to obtain a better physics simulation (Unity Technologies, 2017c). In the case of moving the hand freely around without any collisions occurring, the hand just follows the controller. In the case where the hand has penetrated one or more objects we will depenetrate it from the object. To detect if the hand needs to be depenetrated, we need to check if there is an overlap between the hand's colliders and any other colliders in the world. To reduce the number of colliders to check, we first find all colliders within a radius of the hand. Having these colliders we can now check if any of the hand's colliders have penetrated one or more of them. Unity's Physics API allows us to compute the penetration[4] between two colliders, which gives us the depenetration distance and direction. The direction and distance are defined by computing the minimal translation needed to move the collider out of the penetrated object. Iterating over all colliders in both the hand and the penetrated object, we can use the information gathered to move the hand out of the object, correcting the penetration error. The translation and rotation of the hand happens in Unity's fixed update step, which happens at a fixed interval and before the physics simulations. The depenetration of the hand happens in the update step, which happens after the physics simulation. This allows the physics

[4]To compute the penetration we use Unity's $Physics.ComputePenetration$ method.

simulation to act upon the situation, where the hand is inside a dynamic object and push it away from the hand. Even though the Rigid Hand is kinematic and is not influenced by outside forces, the dynamic objects are and will therefore try to move out of situations where their collider overlap with others'.

Position and rotation:

1. Move the hand's rigidbody to the target controller.
2. Orient the hand's rigidbody compared to the target controller.
3. Find all colliders within a small radius of the hand.
4. Compute if any of the found colliders are penetrated by one of the hand's colliders.
5. Depenetrate the hand based on the penetration information computed in the previous step.

Figure 4.9: Step-by-step process for placing the Rigid Hand.

The Rigid Hand's benefits lie in its simplicity. All the difficult calculations are taken care of by Unity's $ComputePenetration$ method, resulting in a simple implementation but suboptimal solution. No matter the orientation of the hand or the position of its fingers, the hand as a whole is moved outside of any penetrated objects so that it is just touching the surface. This makes for a very stiff look, when the hand is moved over a surface, hence the name. Furthermore, as mentioned above, the compute penetration method returns the direction towards the closest point on the surface, but not necessarily a point on the same side of the object as where the hand entered the object. This leads to the hand jumping between different sides of for instance a cube, when the controller is moved between points that are closer to different surfaces. Other drawbacks are linked to the use of a kinematic rigidbody. Using a kinematic rigidbody means that outside forces do not affect the Rigid Hand. Besides having to do collision checking manually, which the depenetration is used for, it has implications for things like pushing objects and is especially noticeable when pushing dynamic objects into immovable objects, which will react in a very unstable and jumpy fashion, because they are trapped between two objects which which can't be moved away. Pushing objects will also give no feeling of mass without additional implementations. Usually, when pushing an object, the hand applies a force to it and an equal but opposite force would be applied to the hand. This force is used to calculate the acceleration of the hand, which moves it away from the object being pushed. But since the hand doesn't react to outside forces, this movement doesn't happen and difference in weight is not felt.

The Rigid Hand is used as the base for the Sliding Rigid Hand and the Finger Rigid Hand, which both explore additional filtering to improve the overall feel of the hand. The Sliding Rigid Hand tries to do away with the jumping between surfaces occurring due the compute penetration method always returning the direction to the closest point on the surface and the Finger Rigid Hand implements finger filtering to reduce the

amount of position filtering needed by allowing the hand to move closer to the surface by bending the fingers.

4.2.2 The Sliding Rigid Hand

The Sliding Rigid Hand is state based, with two different states: Outside and touching. The two states let the hand behave differently and are used in different contexts. In each frame one of these states will be executed. The state to be executed will be determined in the beginning of the frame depending on information from the last state executed and from the current context.

The outside state is executed when the hand is not touching an object. The hand is first rotated to the orientation of the controller target. After rotating the hand, we check if it's inside an object and we depenetrate it if it is. If it was inside an object, we move the hand a bit further along the depenetration direction to ensure that the sweep which happens next will hit the object, if it's in the path between the hand and the controller. If neither it or any other objects are in the sweep path the hand is moved towards the controller, but if an object is blocking the path we position the hand so that it is touching the object. Unity's $Rigidbody.SweepTest$ method gives us the necessary information to do so: direction and distance to the object that was blocking the path[5].

There is only one transition from the outside state to the touching state; if the sweep detected an object in between the hand and the controller while executing the outside state, the hand will have been placed on the surface of the object. In addition to moving the hand, we also need to save the normal of the point where the hand touches the surface, which is needed in the touching state. When we use Unity's $Rigidbody.SweepTest$ method we gain some information about the object hit, if any. This includes the normal of the point of the object hit by the sweep. In the case an object was hit we switch to the touching state. If nothing was obstructing the path the hand stays in the outside state.

As with the outside state, the first thing that happens in the touching state is that we rotate the hand so that it matches the orientation target on the controller. Because we rotated the hand while it was touching an object, we need to check two things; if the rotation has resulted in the hand having penetrated the object or if the rotation has caused the hand to be removed from touching the surface of the object. First, we check if the hand is inside an obstacle and depenetrate it if necessary. This is done in the same fashion as mentioned for the Rigid Hand (Section 4.2.1). If the hand wasn't inside any object we try to sweep towards the controller position to check if the hand is

[5]The direction and distance found by the $SweepTest$ method is between the closest points between the two objects: The direction and distance between the point on the hand that is closest to the closest point on the object hit.

currently not touching any object. If the sweep resulted in hitting an object in the path towards the controller, we move towards the location hit on that object, like mentioned for the outside state above. The normal of the collision point, either saved during the outside state or gathered from the sweep just mentioned, is needed for the next part: Sliding is where we skip movement in a particular direction (and its opposite), effectively moving the hand on a plane. In this case, we ignore movement along the normal and its opposite. An example of sliding in use is when the hand is interacting with a flat surface and the controller is inside the surface. When moving around the controller inside the object, the hand should slide on the plane created by the normal of the contact point, which is effectively the same as sliding across the surface, if the contact point is updated frequently. To implement the sliding we first find the vector between the hand and the target on the controller. This vector is projected onto the plane defined by the normal. Moving the hand along this new projected vector reduces the distance to the controller without penetrating the object. As a final step for the touching state, the hand is depenetrated again to make sure it's not inside an object after sliding, which might occur if other objects are near the object the hand is currently sliding across. To summarize, the touching state rotates the hand and then checks if the hand has now been removed from or moved into an object and if in any of those cases, it's moved to the position where it's touching again. Afterwards the hand slides to the position on the plane created by the normal which is the closest to the controller.

There are two transitions from the touching state back to the outside state. The first transition checks if the controller has moved back outside the object compared to the surface the hand is currently touching. The direction from the hand's current position to that of the controller is computed. If this direction is in the same general direction as the normal of the surface[6], it means that the controller was moved back outside the object in the direction from where it entered. In this case the hand goes to the outside state. The other transition checks if we can sweep with the hand to the controller. If we can sweep all the way without hitting any objects it means that the hand has slid over an edge and the controller is currently free of any objects. Due to how Unity's $Rigidbody.SweepTest$ method is implemented, it's necessary to move the hand slightly away from the object's surface when sweeping. This is due to the fact that when the object is touched, sweeping doesn't recognize the object being hit. We therefore move the hand slightly back in the direction of the normal of the contact point, before sweeping, but reset the position again when the sweep has been performed. In the case where we hit something, the hand is still touching a surface and we stay in the touching state otherwise the hand transitions into the outside state.

[6]To determine if the direction vector points in the same general direction as the normal, we use the dot product of the two. If the dot product is positive they are pointing in the same general direction.

Position and rotation:

1. Transition to new state if necessary.
    a. If the last state executed was outside, transition to the touching state if:
        1. the sweep during the last outside state execution hit an object.
    b. If the last state executed was touching, transition to the outside state if:
        1. the controller was moved away from the currently touched surface (in the general direction of the normal of the contact point), or
        2. the sweeping towards the controller didn't hit any object.
2. Execute selected state behavior.

The outside state:

1. Orient the hand's rigidbody compared to the target controller.
2. Depenetrate the hand, if it's inside an object.
3. If the hand was inside an object, move it a bit further along the depenetration direction to ensure that the sweep will hit the object, if it's in the direction of the controller.
4. Sweep towards the target controller.
5. If no obstacles in the path between the hand and the controller, move the hand's rigidbody to the target controller.
6. Else move to the position where the sweep hit the obstacle.

The touching state:

1. Orient the hand's rigidbody compared to the target controller.
2. Check if inside object and depenetrate if so.
3. If the hand was not inside an object, sweep towards the target controller.
4. If the sweep hit an object, the controller was not touching. Position the hand on the object hit using the sweep information.
5. Use the normal of the contact point to define a plane on which to slide to minimize the distance to the controller.
6. Check if sliding the hand has led it to penetrating an object and depenetrate if so.

Figure 4.10: Step-by-step process for placing the Sliding Rigid Hand.

What the Sliding Rigid Hand tries to solve is to remove the jumping created by using only depenetration. This flaw is very evident with the Rigid Hand while the controller moves around inside objects. Since depenetration always moves the hand to the closest point on the surface of the object it's depenetrating from, the main idea of the Sliding Rigid Hand is to never allow the hand to penetrate deeply enough to select another surface than the one it entered from. Because the rotation of the hand never penetrates objects very far and and sweep and place never penetrates at all, the outside state penetrates objects very seldom and only very little. The touching state also only penetrates the objects slightly for the same reasons. It uses depenetration and sweep and place, but never places the hand at the controller directly. Because both states don't allow the hand to penetrate far into the object the selected depenetration direction points towards the entry surface even though the controller might be closer to another.

This approach should work well for convex surfaces, but it's more or less precise depending on the update rate of the normal. When the normal is not updated often enough the hand will slide further along the same plane, which might move it away from the surface, for instance when it's moved around a sphere. When the update rate is high enough this approach approximates the optimal solution, which would be for the hand to exactly follow the surface of the object. In our implementation, the normal is only updated upon sweeping, which happens in both states, but only when the hand has been found to be away the object's surface. The information gathered during depenetration doesn't help much, since the methods for computing the penetration in Unity doesn't give any information about the normal on the point of the surface to which the hand would depenetrate, which puts us in a position where we don't update the normal often enough to handle situations like curved shapes properly. This could be improved by finding ways to reliably update the normal of the current contact point more often, which could include sweeping every frame to find the normal during the touching state or to raycast towards a point of the surface found from the depenetration data, for instance. Besides these problems, the same troubles concerning the kinematic rigidbody are inherited from the Rigid Hand, leaving us with a hand that still has unresolved cases, which restricts the use cases in which the Sliding Rigid Hand would perform satisfactorily.

4.2.3 The Finger Rigid Hand

The Finger Rigid Hand is based upon the Rigid Hand's way of moving and rotating, but tries to improve upon it by adding finger filtering on top. Where the Rigid Hand moves and rotates using the rigibody's $MovePosition$ and $MoveRotation$ methods and then depenetrates the hand from possible overlaps with objects in the world, the Finger Rigid Hand adjusts the fingers in between the placement and the depenetration.

Before continuing with how the Finger Rigid Hand and its fingers are placed, we need to mention that the animation for this hand is a bit different from the other prototypes. The animation for the Finger Rigid Hand has a mask for each finger, which means that the fingers can be animated individually even though we use the same animation frames as for the other prototypes, namely one open and one closed hand frame.

As mentioned above the first thing the Finger Rigid Hand does is to place itself at the controller's position and orientation by using its rigidbody's $MovePosition$ and $MoveRotation$. When the hand has been placed, we save the current closed value of each finger. The closed value here isn't necessarily the same as the trigger button value[7], as the closed value is how closed the finger was after the adjustment last frame.

[7] As mentioned above, the trigger button on the controller is mapped to the closing of the hand, by using its value to blend between the open and closed animation frames.

After saving the closed values all the fingers are opened fully using the animation. We need to do this before the next step, which is to calculate the average up vector of all the finger parts for each finger. So for each finger, the average over all its parts' up vectors is saved. When the averages have been calculated, we reset the fingers' closed value to the value saved before and the finger positions are updated according to their closed value using the animation. The next step is to depenetrate the hand from any nearby objects, which is done as previously described. This step is important because we're going to raycast from the fingers later and we need to make sure the origins of the raycasts aren't inside any objects we would like to raycast towards[8].

With all this legwork done, we're now set to start adjusting the fingers. The general idea is that we want to find out how the fingers are oriented relative to the surface they are interacting with. There are three cases we consider for each finger: If the hand is not near any objects, the finger isn't adjusted, if the hand is positioned such that bending or stretching the finger will not allow it to move away from the surface (the rotation axis is parallel to the surface normal), we skip adjusting the finger and lastly, if the rotation axis is not parallel with the surface normal, we use the angle between the normal and the rotation axis to decide on how to adjust the finger.

Before we go into the calculations for the individual fingers, we first calculate the distance between the hand and the controller. All the fingers use this value to calculate how much to adjust. With the distance calculated, the next steps are performed per finger. To find out if a finger's rotation axis is parallel to the surface normal, we first find the colliders within a radius of the hand[9] and find the one closest to the finger. If no collider was found near the hand, we skip the finger adjustment and lerp its closed value towards the value determined by the player's input. If a collider was found, we use raycasting to find the normal on the closest point of the collider[10]. Since we defined the rotation axis of the finger ourselves and we found the normal of the surface, we can take the dot product of the two to see if they are parallel with each other. If they are, the finger cannot bend away from the surface and like with the case of no nearby colliders the finger's closed value is lerped towards the closed value defined by the player's input. In the final case where the normal and the finger's rotation axis are not parallel, we use the dot product between the normal and the average up vector we calculated earlier, and the distance from the hand to the controller to find the value with which to adjust the finger's closed value by. We don't use these two values directly, but feed them each to one of Unity's animation curves[11]: One that evaluates the distance and one

[8] In Unity, raycasts don't hit backfaces of object models, so to be sure that the raycast hits the object at the correct position, the origin needs to be outside of the object.

[9] Unity's physics API exposes the method $Physics.OverlapSphere$ for finding all colliders within a certain radius from a point.

[10] The closest point on a collider can be found using Unity's $Collider.ClosestPoint$ method.

[11] Unity's animation curves are functions that can be visually defined and takes an input value and returns an output value.

that evaluates the dot product. The two resulting values are then multiplied with each other, giving us the adjustment for the finger. After calculating the adjustment value for the finger, we set the finger's closed value to the sum of the closed value defined by the player's input and the adjustment value. When all the fingers' closed values have been set, they are used with the animation masks to update the animation of each of the fingers.

As the final thing after having adjusted the fingers, we again want to move the hand to the controller's position[12] and then depenetrate a final time. We do this because having depenetrated and adjusted the fingers might lead to the hand not being positioned at the controller while also not touching the surface of an object. The hand should only be separated from the controller's position when the hand can't reach it because there's an object in the way. By moving once more and depenetrating we make sure that the hand is placed at either the controller's position or on the surface of the object the controller is inside.

[12]We only need to move the hand, not rotate it, because depenetrating the hand doesn't rotate it.

Position, rotation and finger positions:

1. Move the hand's rigidbody to the target controller.
2. Orient the hand's rigidbody compared to the target controller.
3. Find the average up vector for each finger:
    (a) Open the finger.
    (b) Take the average of the up vectors of each part of the finger and save it.
    (c) Move the finger back to the previous position.
4. Depenetrate the hand from any object it might have entered after moving and rotating.
5. Calculate the distance from the hand to the controller.
6. Calculate the adjustment value for each finger and apply it:
    (a) For each finger, find the closest collider within a radius from the hand.
    (b) If no collider found within range lerp the finger's closed value towards the closed value indicated by the player's input and skip the rest.
    (c) Find the closest point on the collider found.
    (d) Find the normal on the closest point of the collider by raycasting to the closest point.
    (e) If the normal found and the finger's rotation axis are parallel, lerp the finger's closed value towards the closed value indicated by the player's input and skip the rest.
    (f) Else calculate the dot product between the average up vector of the finger (calculated in one of the previous steps) and the normal.
    (g) Normalize the distance between the hand and the controller by dividing with a max distance value defined for each finger and clamping the result between 0 and 1.
    (h) Feed the dot product and the normalized distance into each their animation curve and multiply the results with each other to get the adjustment value for the finger.
    (i) Set the finger's closed value to the closed value indicated by the player's input summed with the calculated adjustment value.
    (j) Update the finger's position using each of the finger's closed value and its corresponding animation mask, which allows for animating the fingers individually.
7. Move the hand's rigidbody to the target controller (after depenetration it might have moved away from the controller).
8. Depenetrate the hand after adjusting the fingers and having moved it again.

Figure 4.11: Step-by-step process for placing the Finger Rigid Hand and its fingers.

As mentioned above, filtering the finger positions can make the hand feel less stiff when interacting with objects, but it can also be used with the purpose of avoiding penetration. The main goal of this prototype was to experiment with the latter. As described above, the fingers bend when the hand approaches an object and the direction they bend in depends on the hand's angle compared to normal of the nearest point on that object. Bending the fingers like this enables the hand to move in closer to the object, before it needs to filter the position to avoid penetrating the object. Another benefit of bending the fingers when interacting with objects is that the hands look a lot less stiff than hands that don't filter the finger positions, making them feel more organic and natural. Since the Finger Rigid Hand's focus is on the improvement of the

feel of the fingers it doesn't solve any of the problems that we mentioned about the Rigid Hand, on which it is based. The Finger Rigid Hand still jumps around an object, when moving the controller around inside the object, due to depenetration selecting the closest point on the object to move to and the troubles introduced by using a kinematic rigidbody also still exist. The addition of adjusting the fingers also introduce some new cases where the hands aren't stable. This prototype doesn't handle edges and corners very well. The fingers behave in strange ways since the sweep of the hand still hits the object even though some of the fingers are free to open fully. At corners the hand can't always decide on the side to depenetrate to, because depenetrating to one side, make the fingers bend in a certain way so that when the hand is depenetrated the next time it's closer to another point on the object, resulting in the hand jumping between several positions around the edge or corner. Again, like with the Sliding Rigid Hand, the Finger Rigid Hand leaves a lot of unsolved cases but shows that there is potential for filtering the finger positions to achieve different feelings in the interaction.

4.2.4 The Physics Hand

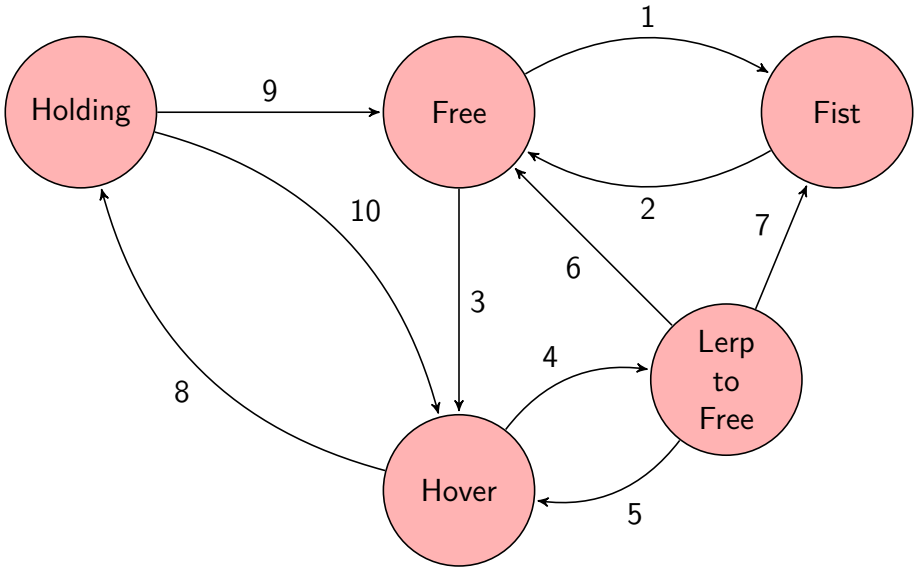
The main difference between the Physics Hand and the other prototypes is that it is using a non-kinematic rigidbody and is therefore fully affected by the physics simulation. This means that it can be pushed around by other objects and the physics system will handle collisions. To properly work with the physics system, the hands need to be moved and rotated differently from the other hands. Instead of setting their position and orientation directly, velocities are applied which make the hand move and rotate to where they need to go. This means that we calculate what velocity is needed for the hand to move the distance between itself and the controller target and the angular velocity needed for the hand to rotate to the controller orientation in one frame. In Unity it's important that these velocities are set in the fixed update method, because they need to be applied before the engine's physics calculations takes place[13]. The Physics hand's movement is based on the concept seen in the NewtonVR asset pack (Tomorrow Today Labs, 2016), but with a few modifications.

Before applying the velocities described above, we adjust the fingers of the hand. As mentioned above, the fingers of the Physics Hand are moved using an IK system. The Physics Hand has several states that indicate whether the finger positions should be filtered or not. The states are named as follows: Free, Fist, Hover, Holding and Lerp to Free. When the hand is not near any objects and the player hasn't closed the hand using the trigger button on the controller the hand will be in the Free state. In this state, the fingers will be positioned according to the animation and the trigger value. The trigger value is used to blend between the open and closed frames of the

[13]Unity's order of execution can be seen from a chart in their online documentation (Unity Technologies, 2017a).

hand's animation. The animation is used to determine the IK targets' locations. The hand can transition from the Free state into the Fist state if the player presses the trigger enough for the hand to be considered closed[14]. In the Fist state the fingers work exactly like in the Free state, dictated by the animation and the trigger value. The reason to have the closed state is to disallow certain transitions that are otherwise possible from the Free state. The Fist state can transition back to the Free state when the player again opens the hand. Besides transitioning to the Fist state, the Free state can also transition to the Hover state. This happens when there is a grabbable object inside the hand's grab area. The grab area is a trigger collider attached to the hand which indicates the range at which the hand can grab objects in the world. When a grabbable object enters the grab area trigger and the object is below the hand, as in the object is on the side the palm of the hand is facing, the hand enters the Hover state. In the Hover state the finger positions will be filtered in a way that tries to let the fingers reach and touch the object. This filtering will be described further below. From the Hover state the hand can transition to either the Holding state or the Lerp to Free state. The Holding state is entered when the player has grabbed an object. In the Holding state the fingers are not updated, effectively fixating them to their last positions, which is the position where the grab happened, resembling a grip. The hand transitions from the Hover to the Lerp to Free state when there is no grabbable object inside the grab area trigger anymore. The Lerp to Free state is inserted in between the Hover and the Free / Fist states to make the transition smooth. In the Lerp to Free state the fingers will lerp from their hover positions back to the positions indicated by the animation and trigger button values. When the lerping has finished, the hand will automatically transition to either the Free or the Fist states depending on whether the hand is closed or not. The Free to lerp state can be interrupted if the hand is again hovering over a grabbable object in which case it transitions back to the Hover state.

[14] As mentioned above, in our prototypes we consider the hand closed when the trigger has been pressed half way, meaning that the trigger value is 0.5 or above.



1. Hand is closed.
2. Hand is opened.
3. Grabbable object in grab area.
4. No grabbable object in grab area.
5. Grabbable object in grab area.
6. Lerping fingers to closed value has finished and the hand is not closed.
7. Lerping fingers to closed value has finished and the hand is closed.
8. When grabbing object.
9. Object dropped and no grabbable object in grab area.
10. Object dropped and grabbable object in grab area.

Figure 4.12: State diagram for the Physics Hand's fingers.

As mentioned above, the fingers' positions are only filtered in three of the five states: The Holding state, the Hover state, the Lerp to Free state. The Free and Fist states don't filter the input but let the fingers follow the animation and trigger button value directly. In the Holding state the finger positions are filtered by ignoring the input completely; the fingers are fixed while holding objects. The Hover state is where the most interesting filtering happens. It has several steps to find the positions for the fingers, but first we see if the there is a need to adjust the fingers at all by checking if the grabbable the hand was hovering over is still within the grab area and if it is we first place the fingers in accordance to the animation and trigger button value.

The Hover state tries to find suitable positions on the surface of the grabbable object on in which to place the fingers. To find the best location for each finger, we try three different option in order. The first of the three option is to find the point

on the grabbable that is nearest to the fingertip[15]. The next option is to find a point that is directly below the fingertip by raycasting along the negative up vector[16] for the finger. The final option is to find some direction in between the two directions found by through the other options: the negative up vector of the fingertip and the direction towards the closest point on the grabbable object. In order to find this middle direction we interpolate between the two direction vectors with respects to a middle interpolation value[17]. The last two methods are not guaranteed to hit the grabbable object at all. The order in which we try these options is as follows: First we see if moving along the direction which is in between the negative up and the direction to the closest point (the middle direction) hits a point on the grabbable object. If it does we use that point as the position the finger needs to move to. If it doesn't hit we try with the negative up from the fingertip and if it hits, we use that hit point as the position to move the finger to. If none of the above methods result in a point on the grabbable object then the closest point on the object is used as the target for the finger. The reason for this prioritization has to do with how good the fingers look after adjusting them for certain placements of the hand compared to the grabbable object and the grabbable object's shape. If the closest point was always used and the hand was hovering over a grabbable cube with the palm on one of the surfaces and the fingers sticking out over the edge, the two furthest joints in the fingers would both bend about 90 degrees without the joint between the fingers and the palm bending much, which is not a natural pose and is therefore not desirable. Figure 4.13 displays the three directions in a diagram. The same will occur when hovering over thin objects like sticks or bats. The closest position on the grabbable from the fingertips will be close to the palm resulting in the weird pose. Using the negative up vector of the fingertips will also not work in a lot of cases. If we again look at the example with the grabbable cube. If the hand is completely open and the fingers are sticking out over the edge, moving along the negative up vector will not lead to a hit on the cube. Because of this, having a middle point between the closest point and the negative up can help in these cases by bending the fingers towards the grabbable object, but with a more natural pose than what results from the closest point. When the positions for the fingertips have been found by using one of these methods we lerp the IK targets of the fingers towards the found positions, to smooth out the movement, after which we call the IK system to solve for the new position.

[15]Unity's API allows us to find the closest point on a collider from a given point by using the $Collider.ClosestPoint$ method on the collider in question.

[16]The negative up vector is the opposite of the up vector which for an object in Unity is defined as a normalized vector parallel to the local y-axis pointing in its positive direction translated into world space.

[17]This value is a value between 0 and 1, where 0 returns the start lerping vector and 1 returns the end lerping vector. Using different values moves the position found for the fingers in between the two given direction vectors. We use the value 0.86 for our Physics Hand.

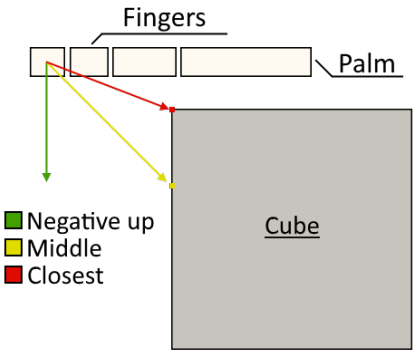


Figure 4.13: Diagram showing the hand from the side above a cube with arrows pointing in the directions used for finding the locations to place the fingers.

Position and rotation:

1. Calculate the angular velocity needed to rotate the hand to the orientation of the controller in this frame and apply it to the hand's rigidbody.
2. Calculate the velocity needed to move the hand to the position of the controller in this frame and apply it to the hand's rigidbody.

Figure 4.14: Step-by-step process for placing the Physics Hand.

Finger positions:

- If in the Hover state:
    1. Check if there is a grabbable object in the grab area of the hand below the palm. If not, skip the rest.
    2. Move all fingers to the location indicated by the animation and the trigger button value as a starting position.
    3. For each finger, get the negative up vector from the fingertip.
    4. Get the closest position from the fingertip to the grabbable object and create a direction vector starting from the fingertip and pointing at the closest point.
    5. Calculate the middle vector as a vector which resides between the negative up and the closest point direction. This can be done by lerping. The amount by which to lerp is a variable which needs to be tweaked.
    6. Raycast towards the middle direction with the max distance being the radius of the grab area. If the grabbable object was hit, set the hit point as the finger's target.
    7. If the previous raycast didn't hit the object, raycast in the negative up direction with the same distance. If the object was hit, set the hit point as the finger's target.
    8. If none of the raycasts hit, use the closest point on the grabbable object as the finger's target.
    9. Set all fingers to their open position using the animation as a base. This is to start the IK solving from the same location every frame.
    10. Lerp the finger's IK targets towards the calculated finger targets.
    11. Let the IK system solve for the new IK targets and move the fingers.
- If in the Lerp to Free state:
    1. For each finger, save its current IK target position.
    2. Get the position that each finger would have according the the animation and trigger button value.
    3. Find an intermediate position for the fingers interpolating from the saved IK target position towards the animation position.
    4. Set the IK target of the finger to the intermediate position found for it.
- If in the Holding state:
    1. Skip all finger movement.
- All other states:
    1. Set each finger's IK target to the position indicated by the animation and trigger button values.

Figure 4.15: Step-by-step process for placing the Physics Hand's fingers.

The previously mentioned prototypes all have a lot of unsolved cases due to their use of a kinematic rigidbody. Collisions must be handled manually and that creates a lot of cases that need to be solved before the hands feel great when interacting with objects of different shapes. Unity's physics system already handles most of these cases when two physics objects interact with each other. Therefore, the main idea behind the Physics Hand is to let the physics system handle all the difficulties that arise due to

collision handling. But besides handling just collisions, using a non-kinematic rigidbody enables the Physics Hand to do a couple of extra things out of the box, one of them being to be able to "lift" objects. We use the word lifting to describe grabbing and moving an object around without using an artificial grabbing mechanism[18], which in our case is activated by pressing the trigger until the hand is closed. An example of how to lift an object using the Physics Hand is to place the left and right hand on either side of an object, move the controllers towards each other (into the object) and then move them upwards. The physics system will use the friction between the materials, the mass and the forces that the hands apply on the object to enable the lifting of the object. This also leads to the sense of weight. Since all interactions, collisions included, are now handled by the physics system, it means that the objects the hands interact with are also applying forces to the hands. It will be impossible for players to lift objects that are to heavy, since the hands will slide across the surface instead of lifting the object. The sense of weight also applies to the context of pushing, where the hands will be separated more from the controller position, when pushing heavier objects than with lighter objects, which gives a different feel to pushing objects of different weight.

Now that we've covered some of the better traits of the non-kinematic rigidbody, let's delve in to parts of its negative side. While both the position and rotation are filtered by the physics system and the position filtering is close to what we're looking for, the rotation filtering has a few cases where it doesn't act in a satisfactory way. When placing the hand flat on the surface of an object and then turning the controller, the hand stays flat on the surface, which feels very natural. The opposite of this is when the hand is pushed towards the surface, fingers first. At first, the hand will look a lot like the Rigid Hand, staying outside the object, but when the controller is far enough from the hand, it jumps to a position where the hand is flat on the surface which feels unnatural. One way to avoid this jump would be to create explicit rotation filtering to handle the cases, where the hand would rotate in a frame and smoothly rotate the hand instead. This problem isn't seen very often though, since the hand rotates more smoothly when it approaches the surface at an angle instead of perpendicularly. Along with the rotation jump, there are also problems that are related to the moving of the fingers. When the fingers bend, the rigidbody moves or rotates slightly, which might make is look like it is shaking. This shaking, which stems from the internal workings of the physics system, can be reduced by increasing the drag and angular drag of the hand's rigidbody to the point where the movement and rotation caused by bending the fingers aren't noticeable any more.

[18]All hand prototypes implement a grabbing mechanism, which when activated allows the grabbed object to follow the hand. The different implementations are described below in Section 4.5.

4.2.5 The Rotation Hand

The Rotation Hand is the only prototype we implemented that explicitly filters all three variables: position, rotation and finger positions. It has two modes of movement. First, we check if there are no objects in the hand's vicinity. We can do this by using Unity's physics API which exposes the method $Physics.OverlapSphere$ for finding all colliders within a certain radius from a point. If there are no objects nearby the position and orientation of the hand is set to the position and orientation of the controller. If there are objects nearby we act upon the closest collider. We first want to rotate the hand. The goal of the explicit rotation filtering is to make the palm face the surface of the object, when the hand is open and make the front of the hand face the surface, when the hand is closed into a fist. These two orientations can be seen in Figure 4.16. The rotation should be more or less complete depending on the distance to the object meaning that if the hand is touching the object, the palm should be facing directly towards the surface and if the hand is far away it should orient itself like the controller target and blend when in-between. To begin, the hand's orientation is first set to that of the controller target. This is used as the starting position from where we apply additional rotation. To calculate the additional rotation to apply to the hand, when approaching an object, the normal of the object's surface is required. Depending on if the hand is open or closed we also want the palm's up vector or negative forward vector respectively. In the case of the open hand, we want the palm's up vector to rotate towards the normal of the surface, when the distance is reduced. Unity provides a lerp method to linearly interpolate between two vectors[19], which is used to find the direction vector to which the palm's up should point depending on the current distance to the surface. To increase the amount of control we have over the rotation, we don't use the distance directly. We first pass it through a function which we define using one of Unity's animation curves. This way we can control if the hand should rotate faster between certain distances, for instance. When the direction vector has been calculated, we use it together with the current up vector (or negative forward in the case of the hand being closed) to define a rotation[20], which we add to the hand's current rotation.

[19] In Unity the $Vector3.Lerp$ method can be used to linearly interpolate between two vectors.

[20] The rotation between two vectors can be calculated using Unity's $Quaternion.FromToRotation$ method.

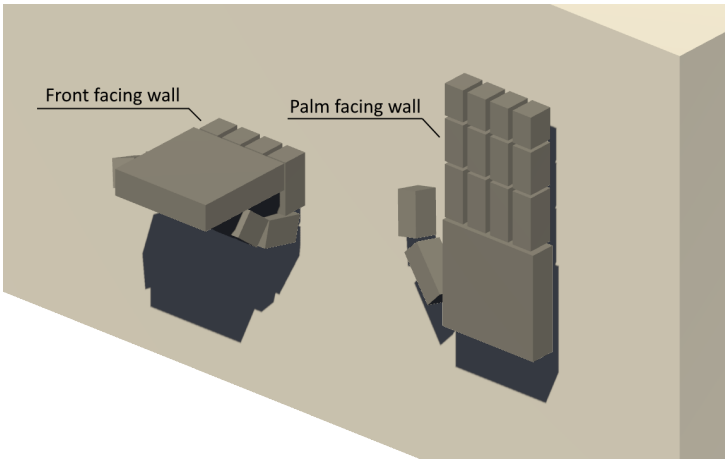


Figure 4.16: Diagram showing the two facing directions: The palm facing the surface and the front of the hand facing the surface.

We move the hand after rotation has occurred. In order for the hand not to penetrate the object it's moving towards, we sweep from the hand towards the controller position. When sweeping, we don't want the fingers to be taken into account, so we don't have colliders on the fingers except for the fingertips. These colliders are disabled at all times, except during the calculation of the finger positions, which is described below. The reason to not want the fingers to be taken into account is that the we want the normal on the point of the surface hit by the palm of the hand and never were the fingers hit. As mentioned before, Unity's $Rigidbody.SweepTest$ method doesn't detect objects that the hand is already touching, so we move it back slightly in the direction of the surface normal before sweeping. If the sweep hit an object, the hand is moved to the hit position, after which it's slid across the surface to reduce the distance to the controller position. The sliding works in the same way as for the Sliding Rigid Hand (Section 4.2.2). The normal is used to create a plane upon which the vector which spans between the hand position and the controller position is projected upon. This new vector is then what determines where to move the hand. If no object was hit during the sweep test, the hand is moved straight to the controller position.

The fingers of the Rotation Hand are moved using inverse kinematics. When we want to move the fingers, there are two situations we consider; either the hand is near objects that the fingers might interact with or it isn't. In both cases we start off by setting the fingers' targets to the positions defined by the controller's trigger value. This position is determined by the hand animation, which is blended between an open and closed animation frame. In the case that there aren't any nearby objects, nothing more is done to the fingers. When the plater presses the trigger they will therefore close the hand and when they release the trigger the hand will open. In the other case, where the hand is close to one or more objects, the fingers will be adjusted so that

they don't penetrate any objects. To do this, all colliders within a radius are found using the same method described above. For each of the found colliders we check if the fingertips are inside by computing the penetration between the fingertip collider and the found collider. Before this step, we enable the fingertip colliders, which are normally disabled. After computing the penetration we disable the fingertip collider again. The distance and direction gathered from computing the penetration is then used to move the finger's IK target. This means that the desired location the finger is trying to move to is moved outside of the object.

Position and rotation:

1. If no objects are within a radius from the hand, move towards the position and the rotation of the controller and skip the rest.
2. Else find the orientation to rotate towards:
   (a) Get the normal of the closest point on the closest object.
   (b) Get the current palm direction vector as the palm's up vector if the hand is open and the palm's negative forward vector if the hand is closed.
   (c) Calculate target direction vector by lerping between the current palm direction vector and the normal vector.
   (d) Calculate the rotation between the current palm direction vector and the target direction vector.
   (e) Rotate the hand towards the sum of the controller orientation and the rotation calculated in the previous step.
3. And then move the hand:
   (a) Move the hand a tiny bit in the direction of the normal (so the hand is not touching the object, when sweeping).
   (b) Sweep the hand towards the controller position.
   (c) If the sweep hit an object, place hand at hit point and slide to minimize the distance between the hand and the controller.

Finger positions:

1. Set each finger's IK target to the animation's finger tip position.
2. Find all colliders within a radius from the hand.
3. If no colliders were found, skip the rest.
4. Else enable finger tip colliders.
5. Compute each finger tip collider's penetration with the colliders found in step 2.
6. Move each finger's IK target using the distance and direction found by computing the tip's penetration.
7. Disable the finger tip colliders again.

Figure 4.17: Step-by-step process for placing the Rotation Hand and its fingers.

As mentioned, the Rotation Hand is the only one of our prototypes that implements explicit rotation filtering, although it's a very simple version and only handles two cases. Both cases, the hand being open and the hand being closed, interact with all objects

similarly. No matter the angle of approach, the size of the object or the type of object the Rotation Hand does the same thing. When open, it rotates so that when touching the surface, the hand is placed flat on the surface with the palm first and when it's closed, the front of the fist is placed against the surface. The rotation filtering works as intended for most surfaces, but could still be improved upon. One thing that feels a bit odd with the current implementation is that the hand also rotates to being palm front when approaching objects with the back of the hand. Approaching an object with the back of the hand being closer than the front could be added as an additional case, where it might not rotate at all or perhaps rotates to let the hand be flat on the surface, but with the back of the hand against it instead of the palm. Other behaviors might also be preferable to the currently implemented cases for other object sizes and types.

While the rotation filtering works as expected in most cases, the fingers have some flaws that are quickly evident. The first problem with the fingers is due to the recurring problem with depenetration; that it selects the closes point on the surface, which isn't always the best suited point. With the fingers, the problem can arise when the hand is placed such that the palm is touching one side of a cube, but the fingers are pointing over the edge. When the hand is in the closed pose and the palm is placed against the surface, the fingertips might depenetrate to the same side of the cube as the one the palm is touching. This pose looks a lot less natural than a pose where the fingers are bend around the edge and the fingertips are placed on the other side, letting the fingers have an overall bend of around 90 degrees. As mentioned, the fingers can also end up being placed partially inside the cube, which is another problem. This might even happen, even though the fingertips are placed on the correct surface in the cube example. This is the result of not depenetrating the hand as a whole. The hand is placed after which the fingers are adjusted, but only the tips of the fingers are depenetrated to find the location for the fingers' IK targets. The IK system doesn't take collisions into account, when moving the fingers to solve for the new target positions. This often results in the middle part of the fingers partially penetrating the objects even after adjusting them, especially around edges and corners. As a final note, it's worth mentioning that this hand also uses a kinematic rigidbody, inheriting the same problems that come with that, as mentioned for the other prototypes above.

4.3 Real Hand Visualization

When the hand is separated from the controller as happens with our prototypes when the controller is moved inside an object, it can be hard to understand exactly where one's real hand is positioned in the virtual world. Not knowing what the actual input to the hand is can lead the players to not understand why the hand is positioned as it is in the virtual world or to believe that the game is experiencing a bug. To visualise the actual input, hoping to make it clearer for players what is happening, we've implemented

a wireframe version of the hands, which is displayed at the position of the controller when the controller is separated more than a defined distance from the hand. The wireframe hand is placed such that it mimics where the virtual hand would have been placed if it hadn't been obstructed, in essence showing where the players real hand is in the virtual world as far as the tracking data allows us to know. In order to not obstruct the player's view too much by showing the wireframe hand immediately upon separation, the wireframe fades in as the distance grows. This is implemented using one of Unity's animation curves as explained above, which fades the hand in by adjusting the alpha value of the visualization from 0 towards 1 as the distance between the controller and virtual hand grows[21]. The result is that after the hand is separated from the controller by a certain distance the wireframe hand starts fading in and becomes more opaque when the distance increases further.

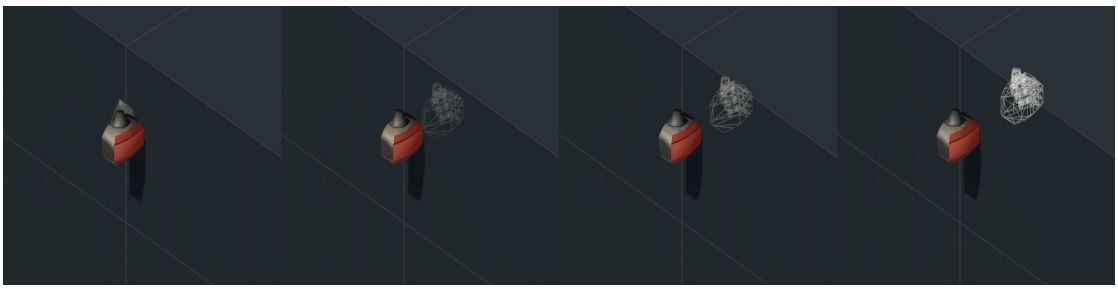


Figure 4.18: Sequence showing the real hand wireframe visualization. A gif can be found here: https://tinyurl.com/HandVisualization .

4.4 Haptic Feedback

Haptic feedback is about giving the user a sense of touch when interacting with a system that otherwise doesn't allow for this. Examples of this can be buttons on touch screens, where haptic feedback can be applied to fill in the missing tactile feedback that is normally received from pressing a physical button. Haptic feedback works by applying forces, vibrations and motions to the user.

The HTC Vive controllers include hardware that makes haptic feedback possible. To access this feature from Unity we used the SteamVR plugin by Valve (Valve Corporation, 2017), which can be found on the Unity asset store. SteamVR functions as a high level interface for the HTC Vive and makes the setup and use of features easier[22]. SteamVR

[21]The alpha value indicates how transparent or opaque an object or image is. 0 means that it's completely transparent and 1 indicates that it's completely opaque.

[22]SteamVR can also be used for other Headsets like the Oculus Rift.

allows a pulse to be triggered on the controller with a pulse length in microseconds as the parameter. The length here is not used to make the vibration take more time, but to make it feel stronger. Features like vibration duration and pulsed vibrations (vibrations with gaps) are up to the developers to implement. We implemented an intermediate layer, based on examples from the SteamVR plugin and from a plugin called VRTK (Sysdia Solutions, 2017) which can also be found in the Unity asset store. It allows us to do define the mentioned vibration duration and pulsed vibrations with a single method we call Rumble. This method allows us to start a vibration for a controller with a given duration, intensity and interval. The duration is the duration of the vibration in seconds, the intensity is the pulse length described above and the interval is the gap in seconds between pulses. For each vibration, these values can be set differently giving a different kind of feel. Setting these values, however, isn't as trivial as it might sound. Especially the intensity and the interval have a lot to say in how the vibration feels to the player and they affect each other, meaning that changing the intensity might require the interval to be changed to make the vibration feel right again.

Our main use of haptic feedback is when interacting with objects in the virtual world, for instance when hitting an object with the Physics Hand, which is the only one of our prototypes that makes use of haptic feedback. To streamline the process of adding haptic feedback to objects in the world, we created the Rumbler component. The Rumbler component can enable haptic feedback in four different scenarios: When the object is hit by the hand, when the hand slides over the surface of the object, when a joint attached to the object breaks while it's grabbed by the hand and when the object is dropped by the hand. Each of these features can be enabled or disabled for the object individually and have their own settings for determining strength and duration of the vibrations. The first three of these use Unity's animation curves to define the intensity of the vibration. When hitting objects the animation curve's input is the magnitude of the impulse resulting from the collision. When sliding over objects, the input for the animation curve is calculated using the velocity of the hand then multiplying it with the average friction values from the hand's and the object's physics materials[23]. For the joint break, the evaluation value is the break force, which is the force applied to the joint, which caused it to break. These are all normalised before being used with the animation curve by dividing them with a set max value and then clamping the result to be between 0 and 1.

When interacting with objects that have the Rumbler component attached, the settings set for the Rumbler's four interaction types dictate the vibration caused in the controller. When hitting a wall, for instance, the wall's attached Rumbler's settings for hitting dictate what values to use for the duration, intensity and interval, when calling the Rumble method described above. In the case of hitting, the intensity is calculated

[23]Unity uses physics materials to store values such as dynamic and static friction and bounciness and other settings used by the physics system. Physics materials are applied to colliders.

based on the animation curve set up for the Rumbler component and the magnitude of the impulse of the collision.

4.5 The Grabbing System

The grabbing system is first and foremost about grabbables. Grabbables (or grabbable objects) are objects that can be grabbed with the hand when the trigger is pressed past a certain threshold. In our implementation, the threshold is when the closed value of the hand is 0.5. We've implemented three different grabbable variants, which all have the same goal, but have different approaches and thus different pros and cons: The Parenting Grabbable, the Fixed Joint Grabbable and the Velocities grababble. When a grabbable object has been grabbed it follows the hand as long as the trigger is held pressed. The grabbable can then be dropped or thrown by releasing the trigger again. The Fixed Joint Grabbable and the Velocities Grabbable can also be dropped if a large enough force is applied to the grabbed object, which can occur if the grabbed object hits something, for instance. Grabbable objects also all have a rigidbody component attached to them so they interact with the physics system and finally, as mentioned above, the Physics Hand uses the fact that it's hovering over a grabbable to indicate that the object can be grabbed by filtering the finger positions.

The Parenting Grabbable does what the name suggests. It makes the hand the parent object of the grabbable object when grabbed. When a parent object is moved or rotated its children are moved and rotated as well. This makes parenting one of the easiest ways of making one object move another, but it also has its downsides. The Unity documentation states that when an object is under physics control, which objects with a non-kinematic rigidbody are, they move semi-independently of the way its transform parents move (Unity Technologies, 2017d). This is an issue, because the grabbable objects all have non-kinematic rigidbodies attached. If nothing is done about this, there could be problems with the collisions and other calculations when the parents are moved and objects would still fall due to gravity. Furthermore, sweeping with the parent rigidbody wouldn't include the colliders of the child rigidbody as well, resulting in wrong placements of hands while grabbing objects. Since rigidbodies can't be disabled directly, we solved this problem by saving all the rigidbody values, removing the component and then when the grabbable object was released, we added back a new rigidbody component and reinstated the saved values. Furthermore, throwing of objects needs to be implemented manually, when using the Parenting Grabbable, because the velocities of the hand aren't applied automatically when it's removed as a child of the hand and it has received its new rigidbody component. To implement throwing we save a history of the velocity and angular velocity of the controller over a certain number of frames. The averages of these velocities are then applied to the grabbable's rigidbody after separating it from the hand. This results in a direction and

velocity for the grabbable that feels satisfactory. Using the grabbed object to push other objects, however, has certain problems which are tied to the use of kinematic rigidbodies (which of course doesn't apply to the Physics Hand). When the grabbable is a child of an object with a rigidbody, it becomes part of that rigidbody. So while an object is grabbed, it's part of the hand's rigidbody. One of these problems occurs when trying to push a dynamic object into an immovable object, in this case using a grabbed object to do so. The object being pushed will be locked between two bodies that aren't affected by the physics system, so the object in the middle can't push the object on either side away, resulting in the squeezed object being unstable and jumpy.

The Fixed Joint Grabbable is implemented using Unity's joints. A joint is used to attach a rigidbody to another or to a fixed point in space. Different joint types exist that have different behaviors. The Fixed Joint Grabbable, as the name states, uses a fixed joint to lock the grabbable to the hand, when grabbed. By using a fixed joint, the grabbable has forces applied to it, which makes it move to the position and rotation indicated by the joint. This target position and rotation is defined when the joint is created and is set as the position and orientation the grabbable object has relative to the hand at that time. Besides the problem of heavy objects dragging the hand along when moving around the controller a spring-like effect can also be seen on the grabbed object when the hand stops abruptly after moving fast. These problems are what the Velocities Grabbable tries to solve, although if explored further and utilized in a more reasonable way these problems could be turned into features that might give artificial grabbing a sense of weight. Another effect of using a joint is that the grabbable can be pushed away from its target location, by forcing it into other objects, which can lead to situations where the hand is suddenly not facing the grabbed object from the same direction as before or the hand is in an unrealistic position to be able to hold the object. Joints also have options that enable special effects like the breaking of the joint when an applied force (or torque) is larger than a certain threshold. We make use of this to allow the behavior of dropping grabbed objects when making them hit immovable objects, for instance. When an object has been grabbed and the object is smashed into a wall, depending on the object, dropping it can add a feeling of realism. In our implementation each object can define their own break forces so they can be customised to fit with their use and expected behavior.

The Velocities Grabbable is moved along with the hand in the same way the Physics Hand is moved. When grabbed, the current position and orientation relative to the hand is saved as the target and when the hand moves, so does the target. From thereon out, the velocity and angular velocity needed to reach the target position and rotation in the current frame is calculated and applied to the object. Setting the velocities of the grabbable instead of letting a joint apply forces removes the springiness seen with the Fixed Joint Grabbable, when suddenly stopping after having moved fast. The issue with the object being able to be separated from its target position and orientation

still persists, though. As an addition to the Velocities Grabbable, compared to the Parenting Grabbable, we wanted to implement the effect of dropping the object, when hitting it hard into other objects. This was given to us by using joints for the Fixed Joint Grabbable. To implement it here, we decided use a max separation in distance and rotation from the target as the threshold for when to drop the grabbed object. When either the distance between the grabbbable and its target is too large or its rotation is to far off, the object will be dropped from the hand, simulating the effect of breaking the joint for the Fixed Joint Grabbable. The implementation of the Velocities Grabbable makes it look and feel very much like the Fixed Joint Grabbable, just without the springiness problem described.

As a final note about the grabbing of objects, it's worth noting that the Physics Hand also allows for lifting objects without grabbing them using the trigger in a stable manner. As mentioned above, lifting has benefits like giving a sense of weight to objects and disallowing objects that are too heavy to be carried. Even if an object is too heavy to carry, it might still be light enough for it to be pushed or rolled. This gives a completely different feel to the act of carrying objects than grabbing with the trigger, although using the trigger method gives a very high amount of control over the grabbed object, as lifting requires the player to look at the object to be able to see how their hands are interacting with the object, while grabbing it with the trigger locks the object to the hand and the player doesn't need to think about dropping it except for when the grabbed object hits another object hard enough and it's dropped. Being very different in feel, lifting and grabbing with an artificial grabbing system might fit better for different types of situations. Natural lifting requires the player to be more concious about the act of carrying, whereas artificial grabbing is more pragmatic and can be used when the player's attention is meant to be elsewhere, perhaps in a task or events in the environment.

4.6 The Baseline: Our Job Simulation Hand

To be able to evaluate our prototypes, we decided to implement a version of hands imitating the behavior seen in Job Simulator, mentioned previously. The defining features of the Job Simulator Hand as shown in the examples in Section 1.1.2 are the fact that it isn't constrained by immovable objects in the virtual world and will freely penetrate objects and that when it is used to grab objects, the hand becomes hidden. We took a black box approach for the implementation of the hand, since we didn't have access to the source code nor were we aware of any detailed descriptions of the actual implementation in Job Simulator. We tested our implementation with different use cases and modelled a control scheme that behaved as closely as possible.

In our implementation of the Job Simulator Hand, it is set up using a kinematic rigidbody, which has its flaws and benefits as described above. The hand is moved

by using Unity's $Rigidbody.MovePosition$ and $Rigidbody.MoveRotation$, which we also used before for some of the prototypes. It has no position filtering to stop it from penetrating objects, so it will always keep the position and orientation of the controller it's following. As in Job Simulator, our implementation also indicates when the hand can grab an object by bending all the fingers slightly. As with the Physics Hand, the hand can grab a grabbable object when it's insid the hand's grab area. Unlike the Physics Hand though, we just use a predefined hover closed value as the closed value of the hand, when hovering over a grabbable. So when the grab area is empty, we use the player's input for the closed value of the hand and set the animation to blend accordingly. When there is a grabbable object in the grab area, we overwrite the input with the defined hover closed value. When the new closed value has been defined, we lerp towards it to smoothen out the motion of the fingers. One thing that none of our prototypes do, but the implementation of the Job Simulator Hand does is to hide itself when grabbing objects. When an object has been grabbed by closing the hand while hovering over a grabbable, we disable all renderers[24] for the hand, effectively making it invisible. When the object is released or dropped the renderers are enabled again. While grabbing, the hands in Job Simulator will in all likelihood also disable their colliders to omit collisions between objects and an invisible entity. We didn't run into the issue during testing and only noticed it in retrospect, resulting in our implementation not including this.

Position and rotation:

1. Move the hand's rigidbody to the target controller using $MovePosition$.
2. Orient the hand's rigidbody compared to the target controller using $MoveRotation$.

Finger positions:

1. Check if there is a grabbable object in the grab area of the hand below the palm.
2. If there is, set the new target closed value to a defined hover closed value.
3. If there is not, set the new target closed value to the trigger input value.
4. Lerp towards the new target closed value and update the hand animation accordingly.

Figure 4.19: Step-by-step process for placing the Job Simulator Hand and its fingers.

4.7 Overview

In this chapter we've examined the categorization of approaches, which describes what position filtering, rotation filtering and finger position filtering are, explaining on a high level how they can be implemented and how they've been used in our prototypes.

[24] In Unity a renderer is a component, which renders a mesh, making it visible in the world.

We approached the Virtual Constraints Problem by using these filters to make the hands behave in a more natural way than what is seen in games like Job Simulator. We've used position filtering as the main mechanism behind stopping penetration of objects and we've used rotation and finger position filtering as a means to minimize the amount of position adjustment needed, as well as to increase the realistic feel of the hands. Furthermore, the finger filtering has also been used as a diegetic way of displaying to the player which objects are grabbable and which aren't. The grabbable system we've implemented has the goal of making it easier to setup which objects can be grabbed by the hands and which can't as well as define different behaviors for different grabbables. The grabbables come in three variants: the Parenting Grabbable, the Fixed Joint Grabbable and the Velocities Grabbable. These three approaches all have their own set of pros and cons. The Parenting Grabbable follows the hand as precisely as possible, but doesn't always work well with the physics system without extra implementations. The Fixed Joint Grabbable interacts as expected with objects around it, while moved by the hand, but has some issues related to the mass of objects which lets it drag the hand around and makes the grabbable feel springy when moving it around fast. The approach taken with the The Velocities Grabbable uses velocities instead of a joint to make it follow the hand, which alleviates the problems mentioned for the Fixed Joint Grabbable.

To help the players create a better mental modal of the system(Norman, 2013), we implemented the visualization of the player's "real hand", which displays where the hand would have been without any filtering. This helps the user understand how the system works as well as show what their actual input is. By showing the visualization we therefore inform the player that the hands haven't stopped moving due to bugs, but because it's the sytem's intention. The visualization system also allows for tweaking of different settings like how it fades in in relation to the distance between the virtual hand and the controller. To indicate when players were touching objects with the virtual hand and to indicate other types of interactions, haptic feedback was put in place. The haptic feedback works by vibrating with certain intensities and intervals over a duration. This vibration can be used to signal to the player that they were stroking or hitting objects or to indicate when events like dropping objects occur.

The hand prototypes along with these additional systems try to solve certain problems. These problems include enforcing physical constraints on the hand, which means that it shouldn't be able to penetrate objects, for instance. Furthermore, using visualization and finger filtering, the prototypes try to reduce confusion and increase understanding of the systems, by letting the players know when they cross boundaries and are near interactive objects. There are problems, however, that none of the prototypes solve. One problem that all of the prototypes have in common is introduced when the controller is placed inside an object, leaving the virtual hand on the surface, and moving the controller sideways. The hand slides over the surface, but when the

hand reaches an edge and the controller is now outside the object, but far away from the hand, the hand jumps or lerps to the position of the controller. This movement, where the hand tries to correct for the distance between itself and the controller feels unnatural, but also emanates from a situation that can't exist in the real world. To the best of our knowledge, this problem isn't solvable with filters if we don't want the hand to penetrate objects in the virutal world. Another issue, which is related to the use of kinematic rigidbodies, is the fact that they interact in a stable fashion when forcing other non-kinematic objects into each other or into immovable objects. These issues have already been described above, but our solution so far has been to use the Physics Hand, which has fewer of these issues due to that it can be affected by its surroundings and allows the physics system to handle such conflict.

For these reasons and because the Physics Hand and the Velocities Grabbable in their current state of development are more complete, our preferred setup amongst our prototypes and systems is the combination of the Physics Hand and the Velocities Grabbable using real hand visualization and haptic feedback and is therefore also the setup we chose to use during our user evaluations.

5 User Evaluation

5.1 Experiment Setup

After developing the prototypes, we implemented a version of the hands from Job Simulator which we used as a baseline for comparison during our user evaluation. For the evaluation we decided to compare only one of the prototypes with the Job Simulator Hand, which was the Physics Hand.

During the evaluations, subjects tested our implementation of the Job Simulator Hand and our Physics Hand in two different environments and gave individual feedback. Each person would first play through what we call the Dev World environment, starting with the Job Simulator Hand to establish the baseline and then with the Physics Hand, after which they would do the same with a more game-like environment, which we call Touchy Island. The Dev World is a small-scale environment with a few basic objects that the users can interact with in different ways. Touchy Island is like a playground, where the player can interact with different creatures and objects on a floating island. The details of both these environments are noted below in Sections 5.1.1 and 5.1.2. While experiencing the two hands in the Dev World, the users were prompted to follow a set of instructions, seen in Figure 5.1, which would guide them through the different interactions possible with the hands to make sure they noticed their features. Besides these interactions, the instructions also mentioned the controls like using the trigger to close the hand. In order for the players to be able to move around the environments, which are larger than the play space of the HTC Vive, we made use of a Unity plugin called Vive-Teleporter (Biagioli, 2016), which was slightly modified to solve problems with grabbing. Teleportation is activated using the trackpad on the controller.

When the testers had gone through the instructions in the Dev World for both hands, we continued to Touchy Island. In Touchy Island we would not guide them with specific instructions, but allow them to freely move around and experience the interactions possible. Sometimes we would hint at certain interactions or features of Touchy Island, if the testers hadn't discovered them within a certain timeframe, but the second environment was meant for playing with the hands. In Touchy Island, the testers would also first evaluate our implementation of the Job Simulator Hand before moving to the Physics Hand, each of which took 5 to 10 minutes. During the evaluations we would record both the screen and the testers themselves through the webcam of the computer running the tests. This material was saved to be able to look back at, if any issues should come up later during the analysis of the evaluations.

- Press the triggers to close the hands.
- Touch immovable object.
- Move controller deep into immovable object to see real hand visualization.
- Hover over a grabbable object.
- Grab a grabbable object.
- Move around the grabbed object.
- Hit an object with a grabbed object.
- Throw a grabbed object.
- Push a movable object.
- Break the joint set up between two of the grabbables by grabbing one of the objects with each hand and pulling in opposite directions.
- Lift a grabbable object without grabbing it with the triggers.
- Teleport using the trackpad on the controller.

Figure 5.1: List of actions the testers were guided through, while in the Dev World environment.

After the evaluations were over, we asked the testers to fill in a questionnaire, where most questions were formulated as A/B tests. They would select which of the hands they preferred in certain contexts or situations. An example from the questionnaire would be: "With which version did you prefer pushing things?". The full questionnaire can be found in Appendix C. During the evaluation, the Job Simulator Hand was named Standard Hand and the Physics Hand was named Adaptive Hand. This was to make them easier to remember and refer to for the testers as well as trying to avoid influencing feedback. At the end of the questionnaire we had two questions, which were about selecting from a list of words the ones they thought fit best with each of the hand versions. The reasoning behind these questions was to get the testers into a descriptive mindset, making them think about how to describe the hands. We are aware of the risks of influencing them in a certain direction, because of our choice of words, but we would argue that if a tester doesn't find any of these words fitting, it could make them realize this and point it out in the "other" field or during the following interview.

The questionnaire was meant to make the testers start thinking about the hands and how they compared in different contexts, which would lead into an interview about the hands. When the questionnaire had been filled, we would inquire about both hands and ask why the tester preferred one hand over the other in certain contexts. The questions were guided by the answers we'd received through the questionnaire to so we could ask about the details for why a tester had certain preferences. We wrote notes about their comments and answers, but did not transcribe them verbatim. The

questionnaires and the notes from the interviews form the base of our analysis. The video material ended up not being used.

Phase	Duration	Tester	Organizer
Introduction	5 mins		Introduce tester to test scenario and purpose.
Play test	5 mins + 5-10 mins	Test Standard Hand and Adaptive Hand in Dev World and Touchy Island environments.	Observe tester, guide through actions in the Dev World.
Questionnaire	3 mins	Tester fills out questionnaire	
Verbal feedback	20 mins	Answer questions about test session and discuss hands.	Ask questions about and discuss the hands.

Table 5.1: Test procedure for the user evaluations.

5.1.1 Dev World

The Dev World was created and expanded upon while the prototypes were developed. It was originally set up to allow us to test our prototypes as they were being implemented. It originally consisted of only a floor, a large immovable cube, some small cubes on top that could be pushed around and an immovable capsule. The immovable cube was used mainly to test how the position filtering worked for the different hands allowing us to test on large flat surfaces, but also around sharp edges and corners, which often introduce difficulties. Later in the process a smaller cube was added only partly sticking out of the bigger cube to allow testing against concave surface. The immovable capsule had the same purpose as the cube, but for rounded surfaces. The smaller cubes could be pushed and later, when grabbing was introduced they could also be grabbed and thrown. These features helped test how the fingers' placement looked, both before and while grabbing the objects.

Later in the process we added a few extra things to test newer functionality implemented in the prototypes and grabbing systems. We added a long thin cylinder, which was used to test finger positioning on smaller objects and was also used to test behavior like hitting objects with a grabbed object instead of with the hand directly. A set of cubes connected in pairs by a joint were also added. The joints have break forces enabled allowing them to break when forces above a certain threshold are detected. If

the cubes are grabbed, one in each hand, and they are pulled apart, the joint will break after offering some resistance. This was created as a play-thing that the users could interact with, but also to test the feel of pulling on objects that are stuck. The last thing added to the Dev World was a large cube which isn't immovable, but has a very high mass. This cube was added to be able to test the effect of pushing heavy objects with the hands, especially the Physics Hand. All our prototypes, with the exception of the Physics Hand, has the same feeling while pushing objects of different weights, whereas the Physics Hand made the users' able to get a sense of weight.

In the Dev World, all immovable objects are dark gray of color and all objects that can be pushed, grabbed or otherwise interacted with are brightly colored to show the users which are which.

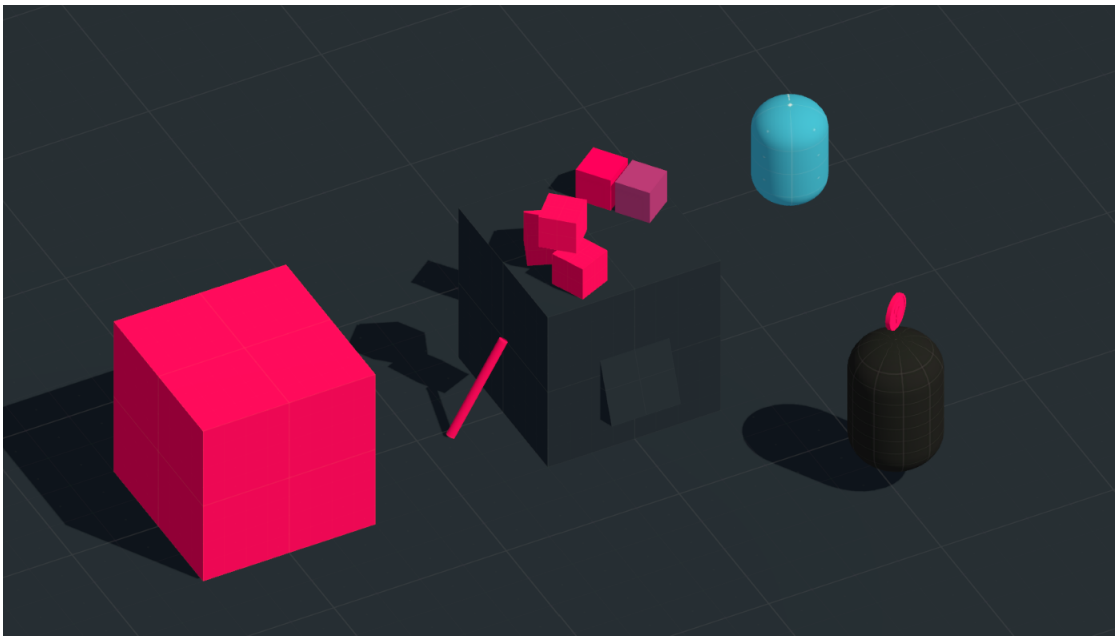


Figure 5.2: An image of the Dev World.

5.1.2 Touchy Island

Whereas the Dev World was created and developed along with the prototypes, Touchy Island was created specifically for use during the user evaluations. Touchy Island is like a small playground or game-like experience, where the player can interact with a couple of different types of creatures and objects on a small floating island. The main idea is that the creatures all have moods and their moods are affected by their surroundings. Each of these creatures also have their own small quirks or features, which differentiate

them from the other types and was meant to encourage the players to interact with them in certain ways without the need for complicated instructions[1]. The four creature types present on the island are the Rock creatures, the Cherry creatures, the Mole creatures and the Cloud creatures.

The Rock creatures start out as small and almost cube shaped. Their shape allows them to be stacked on top of each other, if that's what the player would like to do. The player can then try to move around stacks of Rock creatures while playing a balancing game of sorts. The Rock creatures move around by rolling onto one of the sides it's not currently standing on, like a dice rolling. This also means that they sometimes lie on their sides, their backs or even their faces. The Rock creatures are the only ones that age and grow. Over time the Rock creatures grow old and when this happens, they will be enveloped in a cloud of smoke and they will emerge larger and with crystal hair and a mustache. Rock creatures are also the only creatures that die of old age.

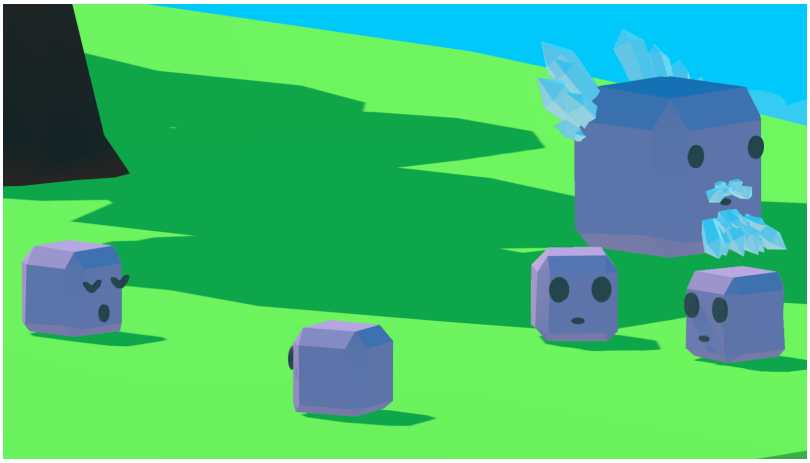


Figure 5.3: An image of the Rock creatures.

The Cherry creatures start their lives growing from the cherry trees. When they are mature, they drop to the ground and start roaming their environment. They move by jumping around, which also makes them hard to catch, when they are not resting for a while. The size of the Cherry creatures makes them more manageable in terms of their size compared to the other larger creatures. Originally we wanted the Cherry creatures to come in pairs connected by their stem and a joint so that they could be pulled apart, but we opted to let the bamboo, mentioned below, serve that purpose instead.

The Mole creatures are capsule shaped creatures that stick their heads out of holes. When touched or hit, they bounce up and down in a spring-like fashion, usually

[1]As part of the planning for the development of Touchy Island we prepared a design document. It contains descriptions of several of the elements that went into the final version of the sandbox, although it wasn't kept completely up to date throughout the process. The design document can be found in Appendix B.

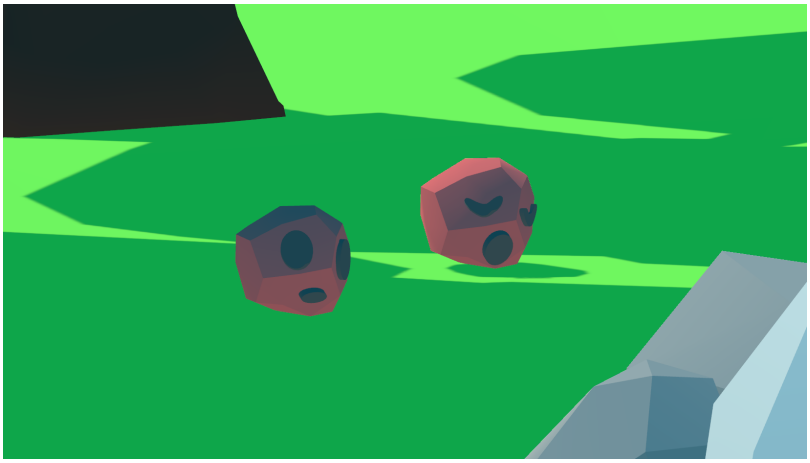


Figure 5.4: An image of the Cherry creatures.

drawing players to hit them even more. When hit hard they even jump out of their holes completely. When grouped together, they could be compared to a game of whack-a-mole. They only move in a straight line up and down, but can also spin around. The Mole creatures are the only ones that don't have an interaction, when the player tries to grab them using the trigger button.

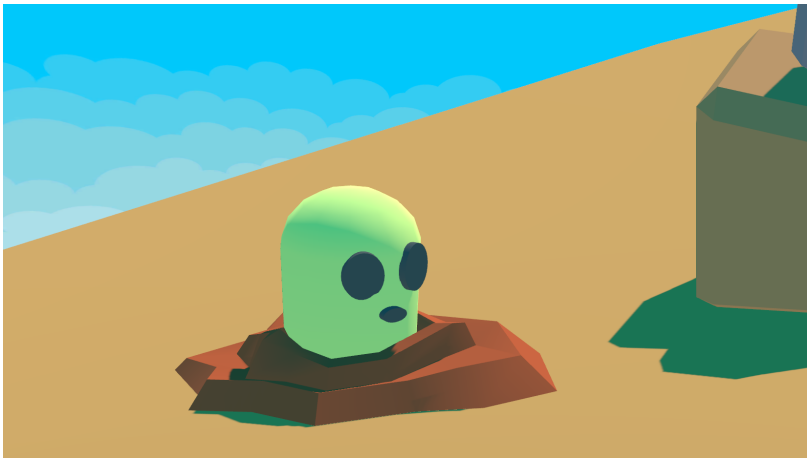


Figure 5.5: An image of the Mole creatures.

The Cloud creatures float around on each of their spot around the island. They can be pushed around if done gently, but will always try to find a comfortable hovering height over the ground. The Cloud creatures are extremely delicate and can't handle hard hits or grabs. If the player chooses to grab or hit a cloud creature they are killed and vanish in a cloud of smoke. So in order for the player to interact with the cloud

creature they need to employ a gentle touch. This encourages the players to handle the Cloud creatures differently from Rock and Cherry creatures that can be grabbed, although that difference can both result in the gentle care of the Cloud creatures or their demise as players try grabbing or hitting them all.

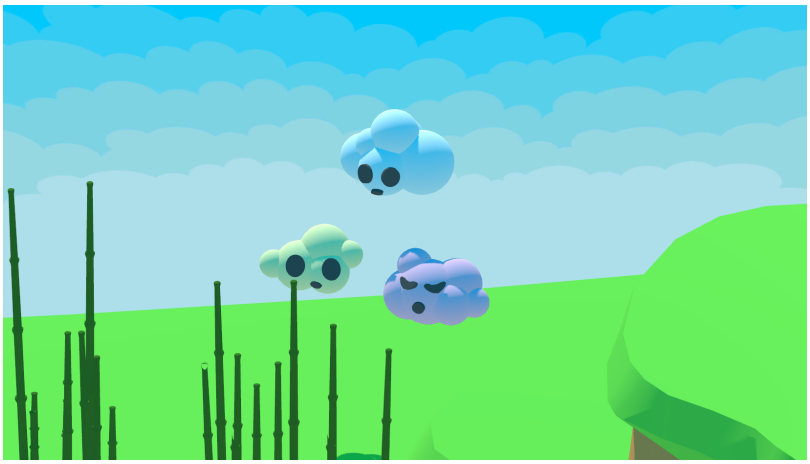


Figure 5.6: An image of the Cloud creatures.

The creatures all have moods which are affected by their surroundings. Creatures can be in one of four different moods: Neutral, Happy, Sad and Angry. Besides allowing moods to increase or decrease over time, the main factor which can change a creature's mood is the player's actions. If a player is gentle and strokes or pets a creature they will become increasingly happy, up till a point where they might even start singing which can make other nearby creatures happier. On the other hand, if the player is throwing or hitting a creature it will become sadder and angrier and surrounding creatures will become sad. The easiest way to make creatures sad is to throw one of their friends over the edge of the island or to poof a cloud by hitting or grabbing it, resulting in their death. A creature's current mood is displayed by their facial expression and by sound effects. Creatures all have a face which can look neutral, happy, sad or angry. Along with that, they also have temporary facial expressions which display the effect of an action. When petted for a while a creature's eyes will turn into hearts, for instance. This way the creatures give the player feedback on the effect of their actions.

As mentioned above there are also two interactive objects besides the creatures. They both exist to allow the player to experience different interactions. The interactive objects are bamboos and movable rocks. On the island there is a small bamboo forest. If the player so desires, they can break off the bamboo from its root and use it as a stick to throw, push or hit things with. The bamboos are connected to their roots using joints which have a certain break force. This allows the player to be able to grab the bamboo and then pull it off its root, but they can also be hit to break them.

The movable rocks are essentially just rocks that are scattered around the island but aren't stuck in the ground. We disabled the grabbing feature from the rocks and made them rather heavy so that players could experience the different feel of weight when pushing and lifting the rocks with the different hands. Finally, the island also has static objects like small crystals, immovable rocks, the cherry trees and some plateaus. When the player interacts with these they'll really see the position filtering in action for the Physics Hand along with the real hand visualization and notice the fact that our implementation of the Job Simulator Hand moves straight through such objects.

Figure 5.7: An image of Touchy Island.

These different creatures and objects all have their distinct purpose in the world because of their unique features. These are not implemented to have diversity, but to draw in the player and make them want to interact with the environment. The reason for the creatures to have moods that are affected by the player's actions is to make the players want to interact with them and thereby use the hands. This also leads back into the decision to make a more game-like experience rather than making another environment like the Dev World or no new environment at all. By first guiding testers through through a set of interactions in the Dev World and learn the features of the hand, releasing them into this playful sandbox would be a different experience which is

closer to the context in which the hands would be used commercially. Since the Dev World is a guided, more abstract environment without distractions and Touchy Island is a more free-form contextualized experience they complement each other and combined give us better results than if we had tested in only one of these environments.

5.2 Results

The information gathered in the user tests through the questionnaire and the interviews is presented in this section. The full questionnaire and the interview notes can be found in Appendices D and C respectively. The questionnaire answers are divided in two graphs: Figure 5.8 summarizes the results of the A/B test-type questions, while Figure 5.9 contains the results of the questions where the testers where asked to select descriptions from a list to describe each hand version (they were also given the option to write their own description). Furthermore, Figure 5.10 summarizes the information from the description selection questions by combining opposing terms into one term (the difference of the number of testers that selected the opposing terms) for each hand version.

Figure 5.11 presents the the result of performing a qualitative data analysis of the interview notes. The notes were coded and categorized following an inductive process, meaning that they were created based on the themes of the interview notes (Burnard, Gill, Stewart, Treasure & Chadwick, 2008), and the number of occurrences of the labels in the notes was counted as a measure of the relevance of each label.

We conducted the test procedure with 10 graduate-level university students (7 men and 3 women). It should be noted that all of the test subjects are part of the games program at the IT University of Copenhagen, so these results might not be fully representative of the general VR game player/user population.

The meanings of labels in Figure 5.11 are described in what follows. In the *Fingers* category, *Stroking Good* means that the tester expressed that the finger adjustment improved the stroking experience, *Hover Good* means that they liked the finger adjustment or hover around movable objects in a general sense. *Impossible Grip* means that they expressed a dislike for the cases when the hands grabbed objects in a way that would be impossible in real life because it would not be possible to get a grip e.g. on a completely flat surface. *Loose Grip* is similar to the previous one, and refers to disliking the cases when they grabbed an object but the hand was in a pose such that didn't make contact with the object. *Hidden Grab* refers to users expressing a dislike for the Standard Hands disappearing when an object is grabbed. *Flat Slap* refers to users pointing out that because of the finger adjustment or hover around movable objects, hitting the objects with a flat hand wasn't possible. *Unnoticed* refers to the cases when the users didn't notice the finger adjustment in the Adaptive Hand.

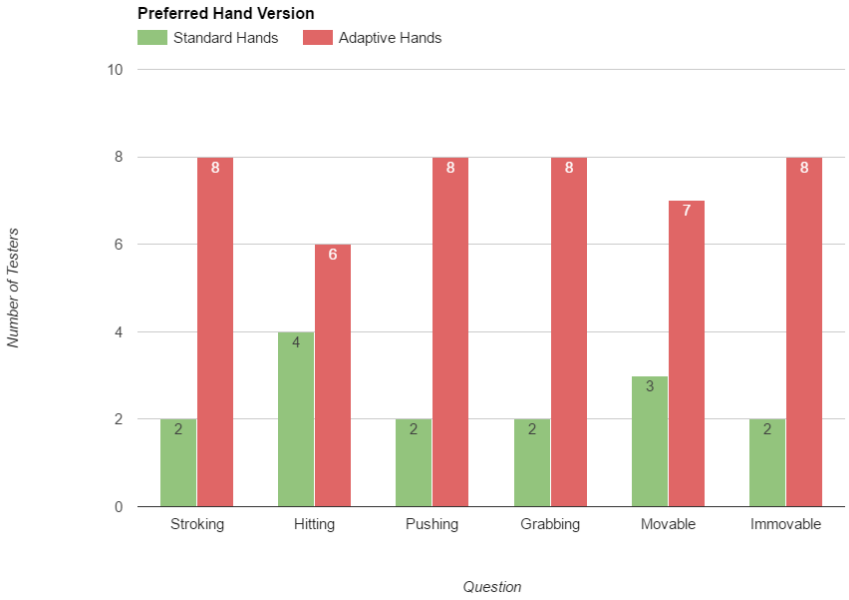


Figure 5.8: Graph showing the number of testers that preferred each version of the hands in each of the aspects described by the A/B test-type questions: "With which version did you prefer petting/stroking the creatures?" (Stroking), "With which version did you prefer hitting things?" (Hitting), "With which version did you prefer pushing things?" (Pushing), "With which version did you prefer grabbing things?", (Grabbing), "Which version did you prefer in terms of how they interacted with objects that you could move?" (Movable), "Which version did you prefer in terms of how they interacted with objects that you could NOT move?" (Immovable).

Inconsistent means that the tester mentioned the fact that the fingers didn't adjust around immovable objects and were inconsistent in that sense.

In the *Rumble* category, the label *Unpleasant* means that testers expressed that the haptic feedback or rumble given by the controllers was unpleasant to them. *Satisfying* means that the users mentioned that the rumble was satisfying at least in some of the cases when they felt it. *Informative* means that the testers found that the rumble informed them about the virtual environment, virtual events or the environment constraints or "rules". *Innacurate* means that the testers described the rumble as "innacurate", "imprecise" or "unpolished". *Intimate* means that the user considered that the rumble made the touch interaction feel more intimate. *Confusing* means that the user expressed feeling confused about why the rumble was being produced.

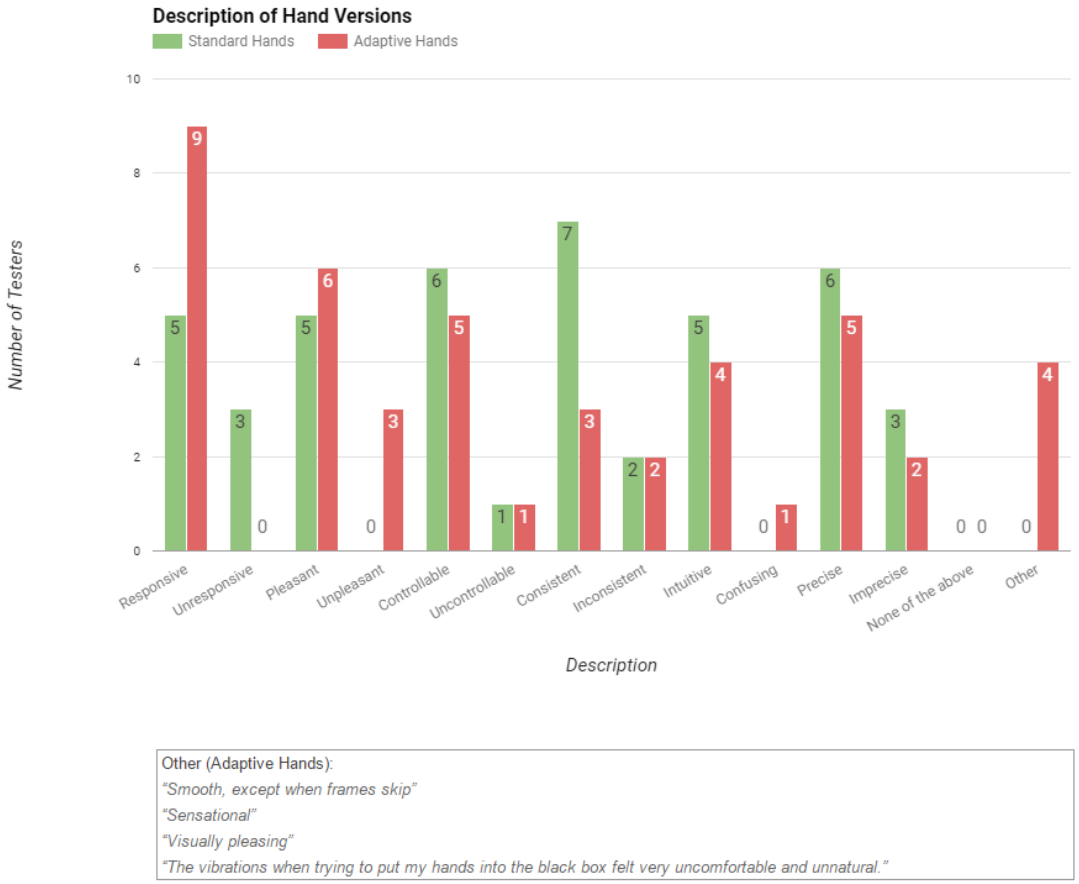


Figure 5.9: Graph of the number of testers that chose each of the words in a list to describe the two versions of the hand that were shown. A few of the testers added descriptions of their own, which are quoted under the graph.

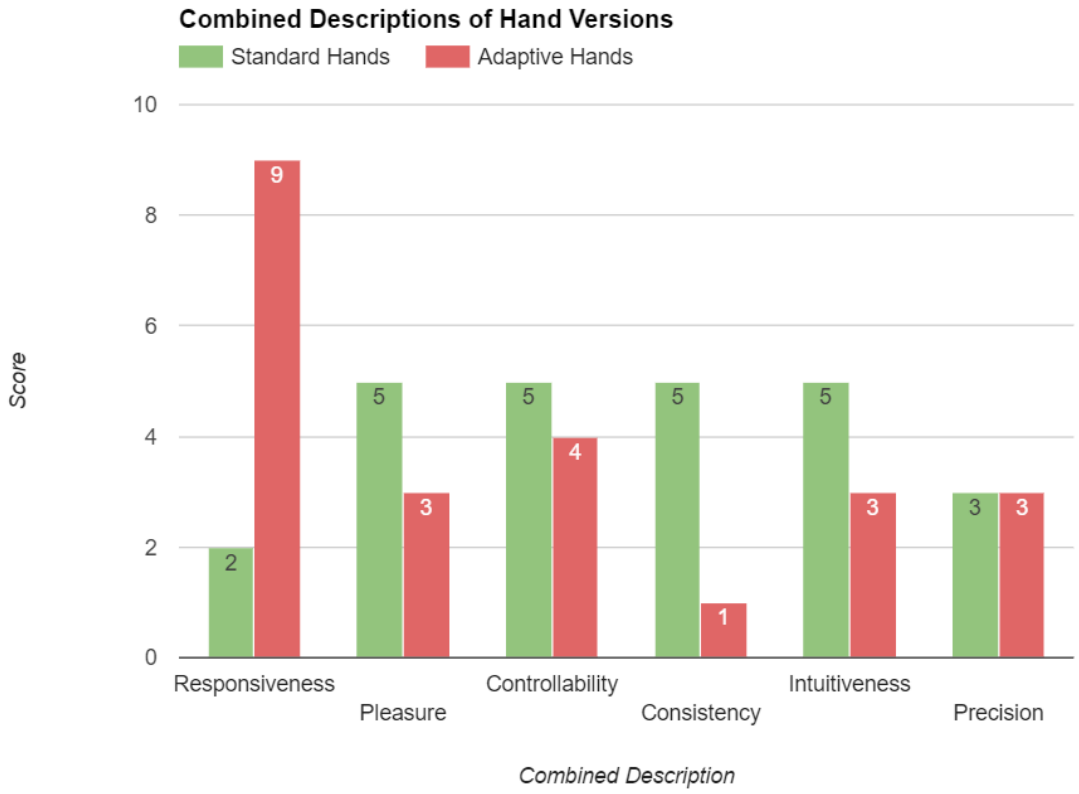


Figure 5.10: Graph that synthesizes the results from the word selection questions (shown in Figure 5.9) by combining opposite terms into one term. For each hand, the new terms are calculated as follows: Responsiveness = Responsive - Unresponsive; Pleasure = Pleasant - Unpleasant; Controllability = Controllable - Uncontrollable; Consistency = Consistent - Inconsistent; Intuitiveness = Intuitive - Confusing; Precision = Precise - Imprecise.

In the *Physics* category, *Natural Grab* means that users liked and found it easier with the Adaptive Hands to lift objects and manipulate them purely using the physics system (e.g. lifting an object by placing their hands on opposite sides of the object and moving their hands towards each other) instead of making use of the "artificial" trigger grab mechanic. *Penetration* means that the testers expressed liking the fact that their hands could not move into immovable objects. *Weight* means that the users mentioned that objects seemed to have weight when interacting with them with the Adaptive Hands. *Hard Lifting* refers to users expressing that the Natural Grab was better, but not easy enough with the Adaptive Hands. *Unnecessary* means that the user expressed

not feeling inclined to break the physics rules of the environment, because there were contextual signifiers (Norman, 2010) that made each virtual object's affordances clear.

In the *Communication* category, *Wireframe Late* means that the users found that the wireframe version of the Adaptive Hand shown when the hand was separated from the controller appeared too late (for example because the controller is inside of a virtual object). *Wireframe Good* means that they found that the wireframe visualization clarified that the system was working as intended and that there was no "bug" when the virtual hands separated from the controller. *No Grab Feedback* means that a user expressed missing some kind of feedback when in the moment when the grab action is completed and the grabbed object is attached to the grabbing hand. *Sound Feedback* means that the user expressed liking the sound effects produced by certain interactions. *Feedback Good* refers to a general positive appreciation of the feedback with the Adaptive Hands.

Fingers	20
Stroking Good	5
Hover Good	5
Impossible Grip	3
Loose Grip	2
Hidden Grab	2
Flat Slap	1
Unnoticed	1
Inconsistent	1

General	8
Body Illusion	3
No Preference	3
Adaptive Real	2

Rumble	16
Unpleasant	5
Satisfying	3
Informative	3
Innacurate	3
Intimate	1
Confusing	1

Stability	7
Shaking	6
Discontinuity	1

Hand Adjustment	6
Bothered	3
Not Bothered	3

Physics	11
Natural Grab	5
Penetration	3
Weight	1
Hard Lifting	1
Unnecessary	1

Communication	7
Wireframe Late	2
Wireframe Good	2
No Grab Feedback	1
Sound Feedback	1
Feedback Good	1

Figure 5.11: Each table shows a category and its labels with the number of mentions found in the interview notes.

Under *Stability*, *Shaking* means that the users noticed and expressed dislike for the cases when the Adaptive Hand shook or moved in a "jumpy" fashion when interacting with objects. *Discontinuity* refers to users noticing small discontinuities in the finger movement in some situations.

In the *Hand Adjustment* category, *Bothered* means that the users expressly disliked when the Adaptive Hands separated from the controllers in terms of position or rotation (not including finger adjustments). *Not Bothered* means that the users expressly

manifested not being bothered or in some cases not noticing when the hands separated from the controller in the previous sense.

Finally, under *General*, *Body Illusion* refers to users expressing that they sometimes found themselves behaving as if they had body parts that the virtual body didn't have. *No Preference* means that the users expressly recognized that there was "nothing wrong" with the Standard Hand or that they had no strong feelings of preference towards either version. *Adaptive Real* refers to users that mentioned that the Adaptive Hands felt "more real" to them.

5.2.1 Discussion

The results from the A/B questions on their own seem to be pretty conclusive in favor of the Adaptive Hands, but at the same time they provide very little detail. However, looking at the results from the description questions and specially the interviews, reveals that the issue is far more intricate than the A/B results indicate.

Reviewing Figure 5.8, only two of the questions were slightly contentious. The lower preference in favor of the Adaptive Hands in *Movable* might be due to the fact that the Standard Hands already work reasonably with objects that can be moved in most cases because these objects can be forced away from the hands, unlike in the case of immovable objects, where the Standard Hands just go through the objects.

The same argument might partly explain why *Hitting* was even more divided. Hitting immovable objects with the Standard Hands is not really possible, so one would expect *Hitting* to be more clearly in favor of the Adaptive Hands. However, we observed that players didn't interact with immovable objects with their hands often and some of the users even reported in the interview that they did not feel inclined to interact with the elements of the virtual environment that seemed to not have a specific interaction (*Unnecessary*), which in the case of Touchy Island happen to be the immovable objects. An explanation then could be that the testers simply didn't experience the difference between the Standard Hands and the Adaptive Hands in terms of hitting immovable objects. Reflecting on the design of the the test environments, particularly Touchy Island, giving the immovable objects interactive properties might have naturally encouraged testers to explore this aspect or the hands more.

The comparison becomes less straightforward looking at the rest of the data: Figure 5.10, interestingly, places the Standard Hands ahead of the Adaptive Hands in most aspects. The only aspect in which the Adaptive Hands seem to stand out is *Resposiveness*. Understood as describing whether the hands "react" to the environment, this seems like a reasonable description for the Adaptive Hands since they both don't penetrate objects and their fingers react in a more dynamic way to movable objects near them. This meaning of responsiveness in the context of hands for VR is, however, not necessarily a desirable quality, since it comes at the cost of behavior that is only indirectly caused by the user's movements, which could translate in losses in Sense

of Agency. On the other hand, it could enhance the tactility of the interactions with virtual objects and could be used to diegetically communicate that certain actions are available with objects. The Standard Hands also use automatic finger movements to indicate that actions are available with the object the hand is hovering over, which should be at least as objectionable in terms of Sense of Embodiment, which makes us think that *Responsiveness* was used by the users in a positive sense.

What might seem like a simple tie in *Precision* and a close score in *Controllability*, we would argue could be in fact a very positive sign for the Adaptive Hands and could clarify the doubts with respect to the value of *Responsiveness*. Given that the Adaptive Hands deviate from the user input, the fact that they are considered equally precise and almost equally controllable by the testers can mean that they do not perceive a loss of control when using the Adaptive Hands, that is, there doesn't seem to be a loss of Sense of Agency after all.

The biggest difference in favor of the Standard Hands is in *Consistency*. We attribute this to two factors: firstly, the finger adjustment in the Adaptive Hands only happens around movable objects. This is also the case with the Standard Hands, but if the automatic finger movement in the Standard Hands is only interpreted as a signifier of the possibility of grabbing an object whereas in the Adaptive Hand finger adjustment is thought of as a natural behavior around and physical object, more related with feeling the objects with the fingers. A more straightforward suspect for this big difference in *Consistency* is the stability issues present in the Adaptive Hands, which were noted by several of the users in the interviews.

By first discussing the interview categories *Rumble* and *Communication*, explaining the results for the descriptions *Pleasure* and *Intuitiveness* becomes easier. The most frequent opinion about the rumble in the Adaptive Hands is that it was "unpleasant", "unnatural" and "uncomfortable" (reported also by one of the testers in the questionnaire). This aspect of the hands turns out to be a rather contentious one, as it seems that some users did find that the rumble was satisfying in certain interactions like hitting and pushing objects or even "intimate" when stroking or caressing the creatures in Touchy Island. It was also described as "confusing", "inaccurate", "imprecise" and "unpolished", which might explain the division of opinions. We consider this to have become a confounding factor for the evaluation of the adjusted hand control model, since only the Adaptive Hands provided haptic feedback. Given these explanations, it seems likely that the differences in *Pleasure* are linked to the problems with the rumble. This seems all the more likely when reviewing the original Figure 5.9 and finding that the Adjusted hands actually were described as "pleasant" by more users than the Standard Hands, but that fact was neutralized by other users describing them as "unpleasant". Our conclusion is then that other aspects of the Adaptive Hands were in fact very pleasant to the testers and had there not been any haptic feedback or if it had been working differently, the results could be far better with the Adaptive Hands.

The difference in *Intuitiveness* might in fact have also been tainted by the negative experience that the rumble caused in the users. Testers described the Adaptive Hands as "confusing" in the questionnaire (remember that the *Intuitiveness* score is negatively affected by users describing the hands as "confusing"), but also explained in the interviews that the rumble was confusing. Testers also seemed to understand how the Adaptive Hand worked better when they saw wireframe visualization, but some of the users indicated that the visualization appeared too late (*Communication*). Based on this, we speculate that if the tests were conducted again without any rumble feedback for either hand, the *Intuitiveness* score might have been similar for both versions.

Focusing on the interviews now, we can see that the fingers got a lot of attention from the users. Many of them expressed liking the stroking and the finger adjustment while hovering over movable objects with the Adaptive Hands. Other users disliked that the Adaptive Hands could grab objects in ways that would make it impossible to get a grip of a similar object in reality (e.g. grabbing by touching only one flat surface, see Figure 5.12). Similarly, others wished that the grip of the Adaptive Hands was tighter and always made contact with the grabbed objects. Some testers also expressly disliked that the Standard Hands became invisible while holding objects. These observations point to a general appreciation of the sense of touch conveyed by the Adaptive Hand and a desire to round it off by adjusting the fingers and perhaps even the hand position and rotation so that the grip is more believable.

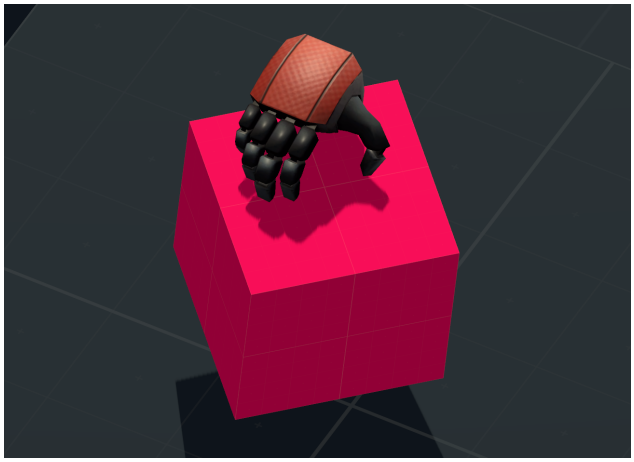


Figure 5.12: Image showing a position where the hand seems unlikely to be able to grab the object, but has done so nonetheless.

In the broad *Physics* category, many testers appreciated that the Adaptive Hands made manipulating objects and holding them in their virtual hands without using the "artificial grabbing mechanic" controlled by the trigger buttons much easier. Less users than we anticipated explicitly valued that the Adaptive Hands did not penetrate other

virtual immovable objects. This might be again be caused by the users having spent less time interacting with immovable objects than with movable objects, which the Standard Hands also don't penetrate in most use cases. As discussed in Chapter 3, PI, Psi and SoE are all subjective experiences and the degree to which each VR user will experience them depends entirely on how they interact with the environment. If they don't experience the cases where the system doesn't work, the illusions will not be broken. In regards to the physics, which we related to Psi, this means that the illusion worked similarly for both hand versions because the users weren't probing the limitations of the Standard Hand. Job Simulator often relies on this knowledge to avoid having to find technical solutions for certain problems, directing the attention of users to the areas of the game where their system works best.

A few of the subjects described unconsciously behaving as if they had virtual arms or a virtual torso only to realize that they didn't in fact have them. These situations strongly point to users experiencing a strong sense of PI and we think these situations happen often when the users are holding objects with their virtual hands naturally, without using the grabbing mechanic. An example of the situations we are describing could be he following: the user tries to hold an object with their virtual hands and intuitively tries to hold it against their chest for support, only to see the object fall as they realize that they have no virtual chest. If this kind of experience is in fact connected to the *Natural Grab*, it would seem that this is an area with a lot of potential for improving PI and perhaps SoE (if this happens when the user actually has the expected virtual body parts).

One of the main concerns when considering implementing an adjusted hand control model to deal with the Virtual Constraints Problem is whether the separation between the user's real hands and their virtual hands would negatively impact the experience of the users (presumably because of loss of the SoE). In this regard, the results from the tests are inconclusive: the same amount of testers reported that the separation bothered them as the ones that explicitly mentioned that the separation didn't bother them, either by explaining that they didn't notice it or even by indicating that they felt like they would rather have a more believable grip on the objects they grabbed even if it meant that the virtual hands would separate from their own real hands. The subjects that expressed a disliking when the virtual hands separated from their real hands were not bothered by the adjustment of the fingers, which they found "natural", in this regard it seems like there was consensus among the testers. The hand position and rotation of the virtual hands is only different from the position and rotation of the real hands when the real hands are inside of virtual objects, but this is also the situation where most users found the rumble to be unpleasant, so this could have influenced their judgment (some users described the rumble as "small shocks" and others identified the rumble in these cases as a signal from the game telling them that they weren't supposed to be doing what they were doing). Furthermore, it is also in these cases

when the stability issues sometimes happened, so improvements in this aspect as well could change the experience for the users.

In sum, the results are not as clear-cut as one might have hoped: we see two possible interpretations of them. The simplest interpretation is to take the results of the A/B questions as the testers' overall opinion of the hand versions, which would indicate that they clearly preferred the Adaptive Hands. The rest of the results (description questions and interviews) in this case show us that there are still many issues with the adaptive hands that could be improved. A second reading of the results, however, is to consider that despite the A/B questions providing favorable results for the Adaptive Hands, the descriptions and the interviews are what actually reveals the testers' experience with each hand version, and in case of contradiction, the A/B results should be disregarded.

Very few testers made observations about the Standard Hands or found any fault with them in the interviews. This could be due to the way the tests were setup, which set the focus on the Adaptive Hands. The testers always tried the Standard Hands first, and then the Adaptive Hands, instead of trying them in a randomized order. The names we used to refer to each version of the hands also framed the Adaptive Hand as an upgrade with respect to the Standard Hands. The hands could have been called "hand A" and "hand B", but we decided against it, fearing that it would make it harder to communicate with the testers.

If one were to follow Occam's Razor, one should find the first explanation to be more likely to be true, but we would still consider the second one as a cautionary note.

6 | Conclusion

This work has attempted to glean knowledge about how to control hands in Virtual Reality using positional tracking devices. Specifically, it has focused on the issue of how to deal with the conflicts that arise between the position and pose of the user's real hands and the physical constraints that the user might expect the virtual environment they are interacting with to impose on their virtual hands: we have referred to this issue as the Virtual Constraints Problem. Our proposed approach to deal with this problem is to adjust the virtual hands position, rotation and finger pose, deviating from the tracking data and other user input in a way that respects the intentions of the user in order to minimize the loss of the sense of "presence" as explained in terms of Place Illusion, Plausibility Illusion and Sense of Embodiment, discussed in Chapter 3. We referred to ways of controlling hands that work like this as Adjusted Hand Control Models.

The controlled experiments conducted in order to compare an example Adjusted Hand Control Model with an Unadjusted Hand Control Model based on the game Job Simulator showed positive results and, together with the recount of prototyping experiments presented in Chapter 4, could provide guidance as to how to proceed in the future when developing Adjusted Hand Control Models. The test setup was however not perfect and we believe there were confounding factors that might have affected the results. Conducting new experiments after solving some of the discovered issues would produce more reliable results. Alternatively, experiments could be performed following established evaluation techniques developed by other researchers for the purpose of measuring "presence" (see overview in (Schuemie et al., 2001)).

Furthermore, we find that previous research could support the hypothesis that separating the virtual hands from the real body as Adjusted Hand Control Models require might not have a negative impact on "presence". At the same time, other developers have begun to create VR games like Wilson's Heart and Lone Echo that use Adjusted Hand Control Models or at any rate approaches that also separate parts of the virtual body from the real body under certain circumstances. The results shown by these developers are promising at the very least.

6.1 Future Work

There is still much to be done in the context of controlling hands in VR, and even more in the larger problem of controlling virtual bodies, instead of just hands. In terms of

avoiding the penetration of virtual objects with the hands, the Adaptive Hands relied almost entirely on Unity's physics engine. This is a convenient shortcut for the purpose of experimenting, but observing the results, it seems likely that custom solutions will be the only way to achieve greater control of the behavior of the virtual hands when separated from the real hands' position by virtual obstacles. The Adaptive Hands became noticeably unstable in many cases and the rotation adjustment sometimes happens in sudden and unexpected ways that are disagreeable for the user. Alternatively, if access to the internal functionality of Unity's physics engine were an option, techniques like the ones used in Lone Echo could become easier (Ready At Dawn Studios, 2017).

Based on the test results, we think that improving the form of the grip of the hands when they grab an object so it is more believable is one of the aspects that could produce the most benefit on its own. Adjustment to the finger positions did not seem to bother any of the testers, but only adjusting the fingers would require the user to be more precise with the hand positioning when grabbing in order to ensure that the fingers can reach the object's surface. An alternative to this is to also adjust the hand position and rotation when necessary so that the fingers can be placed properly on the object. This raises the question of whether this acquired positional and rotational offset should be maintained while the object is held or if the virtual hand should "reach out" to the object and then go back to the position and rotation of the real hands. We take how the hands behave in Wilson's Heart to be a positive sign that this kind of adjustment can work in ways that are not off-putting for users.

During the prototyping process, we assumed that using rotation filtering to minimize the amount of position filtering necessary was beneficial. Observing the fact that testers felt very differently about the filtering on hand position, rotation and finger positions has called our attention to the fact that this is an assumption that cannot be taken for granted and should be verified. Experiments specifically designed to discover whether the tolerance for separation from the real hands in each of these variables is different could inform developer's choices when implementing Adjusted Hand Control Models.

The testers also seemed to find that the finger adjustment when hovering over objects with the Adaptive Hands enhanced the illusion of touching the virtual objects and improved the experience of performing actions like stroking. The hover adjustments however do take some control away from the user, possibly getting in the way of actions like slapping an object with a flat hand.

Finally, an aspect that has clearly not been explored sufficiently in this work, but that is very relevant for VR - hands being no exception - is the use of multimodal feedback. Besides the ever-present visual information, haptic and sound feedback can become powerful enhancers of the illusions we are after: remember that Place Illusion is bounded by the Sensorimotor Contingencies of the system, in particular, this means that every sense we add to the experience will increase the potential degree of PI we can achieve in a virtual reality experience. Some of the comments from the testers

support this: despite the unrefined haptic feedback available in the Adaptive Hands, in some interactions it was reported to be satisfying and to even produce a sense of intimacy in specific actions.

References

Bibliography

Argelaguet, F., Hoyet, L., Trico, M. & Lécuyer, A. (2016). The Role of Interaction in Virtual Embodiment: Effects of the Virtual Hand Representation. *Virtual Reality (VR), 2016 IEEE*, 3–10.

Armel, K. C. & Ramachandran, V. S. (2003). Projecting sensations to external objects: evidence from skin conductance response. *Proceedings of the Royal Society B: Biological Sciences*, *270*(1523), 1499–1506. doi:10.1098/rspb.2003.2364

Botvinick, M. & Cohen, J. (1998). Rubber hands 'feel' touch that eyes see. *Nature*, *391*(6669), 756. doi:10.1038/35784

Burnard, P., Gill, P., Stewart, K., Treasure, E. & Chadwick, B. (2008). Analysing and presenting qualitative data. *British Dental Journal*, *204*(8), 429–432. doi:10.1038/sj.bdj.2008.292

Ferri, A. J. (2007). *Willing Suspension of Disbelief: Poetic Faith in Film*. Lexington Books. Retrieved from https://books.google.dk/books?id=yB_ZOzxqMVcC%7B%5C&%7Dlpg=PR7%7B%5C&%7Dots=B6GfHz7Jnw%7B%5C&%7Ddq=poetic%20faith%7B%5C&%7Dlr%7B%5C&%7Dpg=PA2#v=onepage%7B%5C&%7Dq=poetic%20faith%7B%5C&%7Df=false

Fourneret, P. & Jeannerod, M. (1998). Limited conscious monitoring of motor performance in normal subjects. *Neuropsychologia*. doi:10.1016/S0028-3932(98)00006-2

Gillies, M. (2016). What is Movement Interaction in Virtual Reality for? In *Proceedings of the 3rd international symposium on movement and computing* (p. 31). doi:10.1145/2948910.2948951

Gillies, M. & Kleinsmith, A. (2014). Non-representational Interaction Design. *Contemporary Sensorimotor Theory*, 201–208. doi:10.1007/978-3-319-05107-9_14

Kilteni, K., Groten, R. & Slater, M. (2012). The Sense of Embodiment in Virtual Reality. *Presence: Teleoperators and Virtual Environments*, (21(4)), 373–387.

Koskinen, I., Zimmerman, J., Binder, T., Redström, J. & Wensveen, S. (2011). *Design Research Through Practice From the Lab, Field and Showroom*. Elsevier.

Lim, Y.-K., Stolterman, E. & Tenenberg, J. (2008). The anatomy of prototypes. *ACM Transactions on Computer-Human Interaction*, *15*(2), 1–27. doi:10.1145/1375761.1375762

Norman, D. (2010). Natural user interfaces are not natural. *Interactions*, *17*(3), 6. doi:10.1145/1744161.1744163

Norman, D. (2013). *The Design Of Everyday Things* (Revised &). Basic Books.

Nowak, K. L. & Biocca, F. (2003). The Effect of the Agency and Anthropomorphism on Users' Sense of Telepresence, Copresence, and Social Presence in Virtual Environments. *Presence*, *12*(5), 481–494.

Sanchez-Vives, M. V., Spanlang, B., Frisoli, A., Bergamasco, M. & Slater, M. (2010). Virtual hand illusion induced by visuomotor correlations. *PLoS ONE*. doi:10.1371/journal.pone.0010381

Schubert, T., Friedmann, F. & Regenbrecht, H. (1999). Embodied Presence in Virtual Environments The role of psychology in virtual environments. *Visual representations and interpretations*, 269–278.

Schuemie, M. J., Van Der Straaten, P., Krijn, M. & Van Der Mast, C. A. P. G. (2001). Research on Presence in Virtual Reality: A Survey. *CYBERPSYCHOLOGY & BEHAVIOR*, *4*(2).

Sicart, M. (2008). Defining Game Mechanics. *Games Studies*. Retrieved from http://gamestudies.org/0802/articles/sicart

Slater, M. (2009). Place Illusion and Plausibility Can Lead to Realistic Behaviour in Immersive Virtual Environments. *Philosophical Transactions of the Royal Society B: Biological Sciences*, (1535), 3549–3557.

Steuer, J. (1992). Defining Virtual Reality: Dimensions Determining Telepresence. *Journal of Communication*. doi:10.1111/j.1460-2466.1992.tb00812.x

Waern, A. & Back, J. (2015). *Game Research Methods: An Overview* (P. Lankoski & S. Björk, Eds.). ECT Press.

Ludography

Facebook. (2017). Facebook Spaces. Retrieved from https://www.facebook.com/spaces

Oculus VR, L. (2016a). Oculus First Contact. Retrieved from https://www.oculus.com/experiences/rift/1217155751659625/

Owlchemy Labs. (2016). Job Simulator: the 2050 archives. Owlchemy Labs. Retrieved from http://jobsimulatorgame.com/

Ready At Dawn Studios. (n.d.). Lone Echo. Retrieved from https://www.loneecho.com/

Twisted Pixel Games. (2017). Wilson's Heart. Oculus Studios. Retrieved from http://twistedpixelgames.com/wilsonsheart/

Attributions

Manus VR. (2016). Manus VR Gloves. Retrieved May 9, 2017, from https://manus-vr.com/

Oculus VR, L. (2016b). Oculus Rift+Touch. Retrieved from https://www.oculus.com/rift/

Sony Interactive Entertainment. (2010). PlayStation Move Motion Controller. Retrieved from https://www.playstation.com/en-us/explore/accessories/playstation-move/

Sony Interactive Entertainment. (2016). PlayStation VR. Retrieved from https://www.playstation.com/en-gb/explore/playstation-vr/

Tomorrow Today Labs. (2016). NewtonVR. Retrieved May 16, 2017, from http://www.newtonvr.com/

Valve Corporation & HTC Corporation. (2016). HTC Vive. Retrieved from https://www.vive.com/eu/product/

Other References

Armstrong, P. (2017). Just How Big Is The Virtual Reality Market And Where Is It Going Next? *Forbes*. Retrieved from https://www.forbes.com/sites/paularmstrongtech/2017/04/06/just-how-big-is-the-virtual-reality-market-and-where-is-it-going-next/#781aa3b14834

Biagioli, A. (2016). Vive-Teleporter. Retrieved May 30, 2017, from http://flafla2.github.io/2016/05/17/viveteleport.html

Bye, K. (2016). 'Job Simulator' and the Magic of Hand Presence. *RoadToVR*. Retrieved from http://www.roadtovr.com/job-simulator-magic-hand-presence/

Galyonkin, S. (2017). Job Simulator SteamSpy Data. Retrieved May 12, 2017, from http://steamspy.com/app/448280

Heidegger, M. (1968). *What is called thinking?* (1st). Harper & Row Publishers Incorporated.

Hvas Mortensen, D. (2017). Natural User Interfaces – What are they and how do you design user interfaces that feel natural? Retrieved May 21, 2017, from https://www.interaction-design.org/literature/article/natural-user-interfaces-what-are-they-and-how-do-you-design-user-interfaces-that-feel-natural

Kuchera, B. (2016). PlayStation VR supports room-scale experiences, clearing the way for Vive ports. *Polygon*. Retrieved from https://www.polygon.com/2016/3/17/11254170/playstation-vr-vive-rift-tracking-area

Lang, B. (2016). Touch and Vive Roomscale Dimensions Visualized. *RoadToVR*. Retrieved from http://www.roadtovr.com/oculus-touch-and-htc-vive-roomscale-dimensions-compared-versus-vs-visualized/

Metanaut VR. (2015). Cross-Platform Baseline VR Controls. Retrieved May 13, 2017, from https://utype.wordpress.com/2015/11/02/vrbaselinecontrols/

Oculus VR, L. (n.d.). Introduction to Best Practices. Retrieved May 21, 2017, from https://developer.oculus.com/design/latest/concepts/bp_intro/

Owlchemy Labs. (2017). About Owlchemy Labs. Retrieved May 12, 2017, from http://owlchemylabs.com/about/

Ready At Dawn Studios. (2017). GDC 2017 Session Preview - It's All in the Hands: VR Animation and Locomotion Systems in 'Lone Echo'. Retrieved from https://www.youtube.com/watch?v=A5tsdRb3PFw

RootMotion. (2017). Final IK. Retrieved May 16, 2017, from http://root-motion.com/

Schell, J. (2015). Making Great VR: Six Lessons Learned From I Expect You To Die. Retrieved December 11, 2016, from http://www.gamasutra.com/blogs/JesseSchell/20150626/247113/Making_Great_VR_Six_Lessons_Learned_From_I_Expect_You_To_Die.php

Schwartz, A. & Reimer, D. (2017). 'Job Simulator' Postmortem: VR Design, Tech, and Business Lessons Learned. Retrieved from http://www.gdcvault.com/play/1024256/-Job-Simulator-Postmortem-VR

Slater, M. & Bye, K. (2015). #183: Mel Slater on VR Presence, Virtual Body Ownership, & the Time Travel Illusion. Voices of VR Podcast. Retrieved from http://voicesofvr.com/183-mel-slater-on-vr-presence-virtual-body-ownership-the-time-travel-illusion/

Sony Interactive Entertainment. (2017). PlayStation Move: The Ultimate FAQ. Retrieved May 13, 2017, from https://blog.us.playstation.com/2010/09/07/playstation-move-the-ultimate-faq/

SuperData LLC, D. S. R. (2017). Virtual Reality Market and Consumers. Retrieved from https://www.superdataresearch.com/market-data/virtual-reality-industry-report/

Sysdia Solutions, L. (2017). VRTK - Virtual Reality Toolkit. Retrieved May 30, 2017, from https://www.assetstore.unity3d.com/en/#!/content/64131

Tomorrow Today Labs. (2016). NewtonVR. Retrieved May 16, 2017, from http://www.newtonvr.com/

Unity Technologies. (2015). Movement in VR. Retrieved May 21, 2017, from https://unity3d.com/learn/tutorials/topics/virtual-reality/movement-vr?playlist=22946

Unity Technologies. (2016). Unity Awards 2016 Winners. Retrieved May 10, 2017, from https://unity3d.com/awards/2016/winners

Unity Technologies. (2017a). Execution Order of Event Functions. Retrieved May 21, 2017, from https://docs.unity3d.com/Manual/ExecutionOrder.html

Unity Technologies. (2017b). The Unity Scripting Reference. Retrieved May 16, 2017, from https://docs.unity3d.com/ScriptReference/

Unity Technologies. (2017c). Unity Scripting Reference - MovePosition. Retrieved May 18, 2017, from https://docs.unity3d.com/ScriptReference/Rigidbody.MovePosition.html

Unity Technologies. (2017d). Unity User Manual - Rigidbody. Retrieved May 22, 2017, from https://docs.unity3d.com/Manual/class-Rigidbody.html

Valve Corporation. (2017). SteamVR Unity Plugin. Retrieved May 30, 2017, from https://www.assetstore.unity3d.com/en/#!/content/32647

Appendices

A | VR Controller Comparison Diagram

BASELINE CONTROLS FOR VR CONTROLLERS

METANAUT VR
2015.08.30

FRONT VIEW

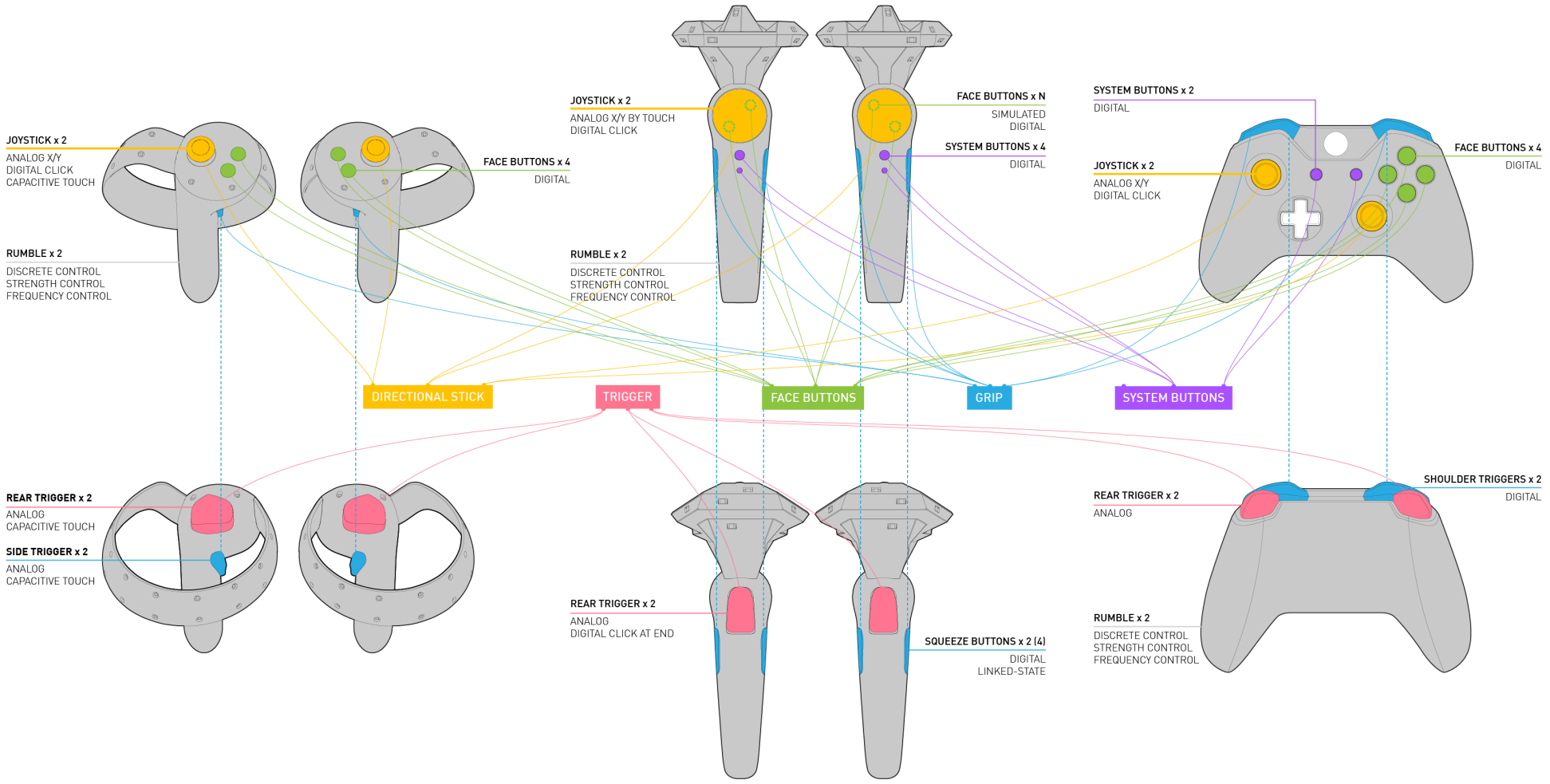
OCULUS
TOUCH

[NON-FINAL]

HTC / VALVE
VIVE

[NON-FINAL]

MICROSOFT
XBOX ONE



LEFT HAND RIGHT HAND LEFT HAND RIGHT HAND BOTH HANDS

BACK VIEW

The controls are colour-coded by their common functionality shared across the three types of controllers.
The number count of each input type depicts the number of unique inputs provided by the buttons.

METANAUT VR
Andrew Yao-An Lee | utypedesign.com | utypedesign@gmail.com
Peter Kao | peterkao.com | peter@peterkao.com
Jean Marais | moshang.net | moshangmusic@yahoo.com
Dilun Ho | dilunho.com | dilun@dilunho.com

B | Touchy Island Design Document

Touchy Island Design Document

Description

Touchy Island is a VR experience, where a player is placed on a floating island full of weird, but cute creatures. The creatures will have different moods depending on their interactions with other creatures and the environment. The player can also affect the moods of the creatures by interacting with them.

Environment

The environment will consist of a floating island. To accommodate for VR gameplay, ground surfaces will be flat but will contain plateaus to break up the flat feeling. The plant life on the island will consist of bamboos and cherry trees.

Items

- Base island ground (should have an underside).
- Plateaus (could be separate objects to be placed, rotated and scaled as pleased).
- Bamboo.
- Cherry tree (maybe more than one trunk? Separate leaves from trunk to reuse?).

Nice to have items

- Grass.

Creatures

The creatures inhabiting the island mostly look like primitive shapes. All “living” creatures will have faces which can have different facial expressions. The facial expressions will be extreme in their portrayal of a feeling, like in Anime and other cartoons.

All creatures hail from certain elements or items, which also informs their look and feel.

Items

- **Rock creature**
  Rock creatures spawn from the rocks (perhaps from plateaus?). They might have crystals on their bodies indicating their age. They weigh more than the other creatures, since they are made of stone.

- **Earth creature**
  Earth creatures mostly live underground but will sometimes stick their heads out into the open. People have been trying to pull them out of the ground to see what the rest of their bodies look like, but so far they have persistently kept themselves from being pulled out of their holes by holding on tight or trying to hide, when people get near.
- **Cherry creature**
  The cherry creatures grow from the cherry trees. They always comes in pairs. When fully grown they can be shaken down from the trees making them plummet to the ground. The two connected cherries (siblings) aren't always happy with each other (which might show in their moods). The player can separate two connected cherries and if they liked each other, they will become sad, if they didn't like each other they will become happy.
- **Cloud creature**
  The cloud creates can spawn anywhere from the water in the air. Cloud creatures are very light. So light in fact that they sometimes feel like they don't weigh anything at all. They can float around, but they can't be grabbed without disappearing with a poof. So to handle these creatures, a gentle nudge is the way to go… Perhaps a nudge with both hands.

Behaviours and interactions

The behaviours and interactions are split into two categories: Behaviours and interactions between creatures and behaviours and interactions between player and creature.

Creature moods

The creatures can have different moods, which depends on how they experience the world around them. Each type of mood has a value that can be increased or lowered.

The moods are as follows:

Name	Description	Other
Happy	When the creature is happy, it starts smiling. Happy creatures might also sing small tunes sometimes. Happiness can be increased by petting or stroking a creature and can be reduced by hitting a creature.	
Angry	When a creature is angry, it will look angry and will not sing at all. Creatures can become angry if you hit it or another creature nearby. Creatures can become less angry by petting or stroking them or by nearby creatures singing. Anger might reduce over time.	

Shocked	When creatures become shocked they will open their eyes and mouths and make a “shocked” sound. Creatures can become shocked if the player throws a creature over the edge of the island. Shock vanished quickly over time. Often shocked creatures have heightened anger, which means that shock can lead to anger.	Decreases over time (*medium/a little fast*)
Scared	When creatures become scared they will look scared and might shake a bit. Creatures can become scared if a player continuously hurts nearby creatures or throws creatures over the edge. The scaredness meter should slowly increase with evil deeds and slowly decrease with good deeds.	

Implementation of moods

Each mood will be a value between 0 and 100 (percentages) (perhaps certain moods can have different bounds, for instance shock might have an upper bound of 110, so it has the opportunity to always be the top one and overrule the others). Influences from the environment and the player will affect these values as well as some of them might change over time. The creature will always be in only one mood, which is the one which currently has the highest value amongst all the moods.
Implementing moods in this way separates the moods from each other and makes it easier to extend the palette of moods without major changes (although balancing of values might be necessary).
For each mood a creature can also define multipliers (one for positive values and one for negative values). A multiplier either increases or decreases how much a creature is influenced by that mood. For instance a creature might not be very susceptive to anger and therefore has a multiplier of 0.8 for anger, when it is increased, and a multiplier of 1.1 for anger, when it is decreased, making it harder to anger the creature, but easier to make less angry.

Influencing mood values

Influencing mood values will happen through influence spheres. An influence sphere is an invisible object, which affects all creatures within its radius. When created it will influence creatures within its radius with its values (creatures might have multipliers for each mood that increases or decreases the value before being applied).

Creature behaviour and interactions

Movement

- **Rock creature**
  The rock creatures roll around the world 90° at a time. Small/young rock creatures might

roll faster, whereas bigger/older ones roll from side to side slowly. When they grow old enough so their crystals are large they might get stuck, since the crystals have been thrust into the ground. When this happens the player can help them by pushing them. Otherwise they have found their final resting place. Perhaps this is how the plateaus were originally created.

- **Earth creature**
  How the creature moves underground is still unknown to this day. They only come above ground to take a peek, so they will appear from a hole and then go back under.
- **Cherry creature**
  The cherry creatures will jump a bit around, but they will roll as physics indicate. This makes them feel quite helpless sometimes, which means they are prone to be scared and sad and they really appreciate it when others are nice to them.
- **Cloud creature**
  The cloud creatures can float around as they please, although the often just let the wind take them where it wants them to go. They might fly near to the player as they are quite curious creatures, but otherwise they will just be floating around taking in the scenery.

Creature events

Different moods can lead creatures to do certain events, for instance singing, which can affect creatures around them.

Mood	**Event**	**Description**
Happy	Sing (low chance)	When a creature is singing, surrounding creatures will increase their happiness (*a little*) and decrease their anger (*a little*) and sadness (*a little*).
Angry	Scream/shout (very low chance)	When a creature screams out of anger, surrounding creatures will increase anger (*a little*), shock (*medium/a lot*) and sadness (*a little*) and decrease happiness (*a little*).

Player interactions

The player will interact with the creatures in the world and depending on the player's actions the creatures will react differently.

Creature interactions

Stroking/petting

How to	The player places their hand gently on the creature and moves it

	around the surface of the creature.
Effect on creatures	Creatures like being petted. Happiness (*medium*) will increase, anger (*medium*), sadness (*a little*) and fear (*very little*) will decrease.

Hitting

How to	The player quickly moves their hand into a creature.
Effect on creatures	Creatures don't like being hit. Anger (*medium*) and fear will increase, happiness (*medium*) will decrease.
Special effect: Cloud creature	The cloud creatures will disappear into a cloud of smoke (killing them). This will increase shock (*a lot*) and fear (*a little/medium*) in the surrounding creatures.

Pushing

How to	The player places their hand(s) on the creature and moves the hands towards the creature ("inside" the creature).
Effect on creatures	Creatures don't mind being gently pushed (if it's not over the edge). No effects will take place while pushing.
Special effect: Rock creature	If an older rock creature is stuck on its side and the player pushes it back on its "feet" the rock creature will gain happiness (*medium/a little*)

Grabbing

How to	The player places a hand on the creature and closes the hand.
Effect on creatures	Creatures don't mind being grabbed. No effects will take place while grabbing.
Special effect: Cloud creature	The cloud creatures will disappear into a cloud of smoke (killing them). This will increase shock (*a lot*), sadness (*a little/medium*) and fear (*a little/medium*) in the surrounding creatures.

Killing

How to	The player places throws a creature over the edge of the island or hits a creature hard with the tip of a bamboo (thrust or thrown).
Effect on creatures	Creatures just hate getting killed… Don't you? This will increase shock (*a lot*), sadness (*a little/medium*) and fear (*a little/medium*) in the surrounding creatures.

C | User Evaluation Questionnaire

VR Hands

Thank you for helping us! Please answer these questions.

*Required

1. **What is your name?** *

_____

2. **With which version did you prefer petting/stroking the creatures?** *
   Mark only one oval.
   - ◯ Standard hands
   - ◯ Adaptive hands

3. **With which version did you prefer hitting things?** *
   Mark only one oval.
   - ◯ Standard hands
   - ◯ Adaptive hands

4. **With which version did you prefer pushing things?** *
   Mark only one oval.
   - ◯ Standard hands
   - ◯ Adaptive hands

5. **With which version did you prefer grabbing things?** *
   Mark only one oval.
   - ◯ Standard hands
   - ◯ Adaptive hands

6. **Which version did you prefer in terms of how they interacted with objects that you could move?** *
   Mark only one oval.
   - ◯ Standard hands
   - ◯ Adaptive hands

7. **Which version did you prefer in terms of how they interacted with objects that you could NOT move?** *
   Mark only one oval.
   - ◯ Standard hands
   - ◯ Adaptive hands

8. **Which of the following would you use to describe the STANDARD hands (select all that apply)** *

Tick all that apply.

- [ ] Intuitive
- [ ] Confusing
- [ ] Responsive
- [ ] Unresponsive
- [ ] Precise
- [ ] Imprecise
- [ ] Pleasant
- [ ] Unpleasant
- [ ] Consistent
- [ ] Inconsistent
- [ ] Controllable
- [ ] Uncontrollable
- [ ] None of the above
- [ ] Other: _____

9. **Which of the following would you use to describe the ADAPTIVE hands (select all that apply)** *

Tick all that apply.

- [ ] Intuitive
- [ ] Confusing
- [ ] Responsive
- [ ] Unresponsive
- [ ] Precise
- [ ] Imprecise
- [ ] Pleasant
- [ ] Unpleasant
- [ ] Consistent
- [ ] Inconsistent
- [ ] Controllable
- [ ] Uncontrollable
- [ ] None of the above
- [ ] Other: _____

D | User Evaluation Interview Notes

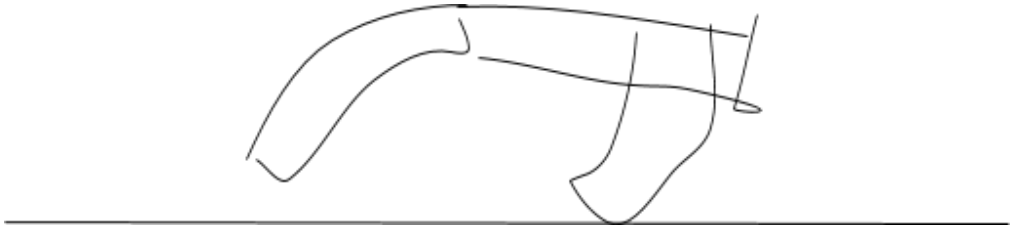
Interview notes

Sophia

- Rumble confusing
- Stroking a bit better with adaptive
- Felt like standard needs to get closer to object to grab it
- Lack of feedback when hand grabs. Thought hand disappeared because of bug.
- Adaptive felt more real
- Standard was pleasant as well because it did what i wanted
- Easier to grab by just pressing it
- More feedback was good with adaptive
- No penetration was good
- Bothered by inaccurate rumble
- Adaptive was more responsive
- Stability is better in standard
- No penetration was informative
- Adaptive no penetration made her feel more inside the world

Raghav

- Adaptive hands allowed for caressing clouds
- Adaptive hands allow for gentle interactions
- Liked holding things without grabbing (specially with the adaptive hands)
- Hitting bamboos together was nice with rumble
- Adaptive haptic feedback allows to feel touch of objects without visual feedback
- Some of the objects had weird weights (or center of mass)
- Fun to play with adaptive hover state of hands
- Adaptive hands felt more real
- Grabbing can feel weird and unrealistic
- Prioritize natural physics grabbing (not using trigger)
- Strange to grab an object when grip is not possible (see image)



- Missed having a body to balance a natural hold against
- Notice state changes in adaptive hand

- Disliked the non firm grabbing
- Lift and grab should be different

Lucas

- Needed to get used to adaptive
- Liked visualization, still 1 to 1 mapping. Clarifies that there is no bug. System still tracking actual hand
- Adaptive hand felt like an upgrade, but there was nothing wrong with the standard one
- Preferred grabbing because of haptic feedback
- Intimate feeling with haptic feedback on stroking
- Sound feedback helped
- Haptic of contact happening
- Didn´t like hand disappearing when you grabbed something
- Right pose at the cost of losing exact tracking would be preferable
- As long as it meets the intention, filtering is good

Pablo

- Instability breaks immersion more than breaking physics laws
- Enjoyed vibration
- Adaptive nice when it works
- Adaptive fitting to control non-anthropomorphic creatures (alien, tentacles…)
- No strong preference in choices
- Resistance of objects was nicer with adaptive. Everything breaks with standard hands.
- Liked free grab (better without grabbing)
- Hand positioning felt unnatural

Alberto

- Too much rumble. Not gradual enough
- Expected to have arms and behaved like he did
- Preferred stroking with standard, rumble might have been the cause
- Liked visualization, informative about actual players position
- Preferred adaptive with static objects
- Wasn’t bothered by input filtering, almost didn’t notice it
- Adaptive pleasant because of adaptiveness but unpleasant because of rumble
- Rumble was unpolished and imprecise but appropriate use

Peter

- Adaptive makes sense for stroking
- Flat hand slap meh meh
- Adaptive didn't feel adaptive around static objects
- Adaptive generally felt like they followed his intentions, but not when the fingers didn't adapt
- Did not feel disconnection from avatar
- Wireframe became visible too late, user got confused when hand didn't follow 1 to 1
- Language that informs of whether an object is intractable was enough to make user not want to break the physics rules.
- Standard hands not responsive to the environment
- Noticed stability issues
- Prefered standard because with movable objects because adaptive were a bit unstable
- Stroking creatures was pleasant

Praveen

- Liked finger adaptation
- Wasn't immediately apparent that the computer was still tracking the controller
- Uncomfortably rumble
- Hover was nice, but standard was nicer inside objects because of rumble discomfort
- Very bothered by disconnect between hand position and rotation and controller
- Finger disconnection didn't bother him because it felt natural that the fingers would adapt. Natural reaction of fingers.

Sarah

- Easier to pick up without grabbing with adaptive, but not easy enough
- Missed having a body to hold things against
- Rumble intensity informed of penetration
- Unstable interactions with objects when lifting

Marco

- Deviation from tracking devices felt unnatural when lifting
- Grabbing a flat surface should be impossible
- Did not notice finger adaption

Camila

- Haptic feedback was unpleasant
- Bothered by deviation from their hand position
- Haptic feedback was satisfying when hitting and pushing
- Lost focus on the hands in Touchy Island once she started exploring the game itself
- Haptic feedback described as “small shocks”