Utilizing the Redirected Walking Algorithm to Avoid User-Obstacle Collisions

A THESIS
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Anicia Dcosta

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

Dr. Pete Willemsen

July 2016

Acknowledgements

I would first like to thank my thesis advisor Dr. Pete Willemsen at University of Minnesota, Duluth. He has always inspired me and provided me with guidance needed for my thesis. Door to Prof. Willemsen office was always open whenever I hit a roadblock or had a question about my research or writing. He consistently allowed this thesis to be my own work, but steered me in the right direction whenever he thought I needed it.

I would like to acknowledge Lisa Fitzpatrick, Logan Sales and the other members at MMAD Lab, University of Minnesota Duluth, who were kind enough to provide me the infrastructure to conduct my experiments.

I would also like to acknowledge Jonathan Beaulieu and Alek Straumann at SIVE Lab, University of Minnesota Duluth for their constant support and encouragement.

Finally, I must express my very profound gratitude to my parents and to my spouse for providing me with unfailing support and continuous encouragement throughout my two years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Dedication

I would like to dedicate this thesis to my husband Melwyn Dsouza who has provided me with such a tremendous support throughout my studies. He has always been an inspiration to perceive my dreams in this field. I would also like to dedicate my thesis to my parents Eugene Dcosta and Leo Dcosta for their unconditional love. I would also like to dedicate this thesis to my advisor Dr. Pete Willemsen for being a role model and source of continuous inspiration.

Abstract

Virtual Reality (VR) is an imitation of the real world. VR provides people an opportunity to experience surroundings that simulate real experiences, train people for situations or provide interaction with situations that do not exist. VR allows users to learn about a particular environment which would not be possible due to reasons such as time, distances, expense and safety. Virtual environments (VE) are usually presented through Head Mounted Displays (HMD). There are different locomotive techniques to explore virtual environments. The most common and immersive one is natural walking with a HMD in a room-size or larger tracking space. However, the size of the VE system that can be explored by walking is limited by the size of tracking space area.

Redirected Walking is a promising, low-cost algorithm for enabling large virtual exploration via natural locomotion in smaller tracked spaces. This thesis utilizes a Redirected Walking algorithm that is used to steer user's exploration in an infinite virtual scene while augmenting the algorithm to avoid potential physical obstacles that may exist in the tracked space itself. Adding obstacle avoidance to the Redirected Walking algorithm makes the tracking space more robust because there can be restrictions such as physical tables, doors or other obstacles that could hinder the user's movements in VR. Moreover,this addition has potential use with passive haptics in Redirected Walking situations.

The Magic Barrier Tape Technique is another alternative for exploring a large virtual environments on foot when the size of the physical surroundings is small by taking advantage of people's natural ability to spatially update. Magic Barrier Tape is an interactive locomotion technique that helps the user to navigate in an infinite virtual space while restricted to a limited tracking space area. In the Magic Barrier Tape algorithm the user's pointing direction acts like a joystick to control their movement when near the tracking space boundaries. Magic Barrier Tape has the potential to become an effective resetting

technique for the Redirected Walking algorithm because it may be more natural and there is a continuous visual flow for the users in the system. This thesis implements Redirected Walking while also dealing with potential collisions between users and physical objects in the laboratory.

Contents

List of Figures

1 Introduction

Virtual Reality is a powerful technology that provides an interactive environment that can be utilized for solving different problems. Speaking technically, Virtual Reality can be defined as a 3D computer generated environment where users can interact and explore the environment by walking. In the real world, our minds learn a lot about the physical environment with the help of different senses in our body. The complete understanding of the environment is the combination of the information received from each individual sense along with brain activities. Considering this fundamental fact if we were able to modify the information for users senses then the brain mechanism maps it accordingly. The mapped information would be presented as one version of reality to the user, when actually it is not. This manipulation of perception is the key feature used to give a user the feel of a real world in VR when actually the world is not real. Optical flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer (an eye or a camera) and the scene. Modifying optical flow manipulates the scene in VR environment which helps in giving the user the feel of the real world. The various fields where the VR applications can be used are architecture, medicine, sport, arts etc.

Another problem is exploring a large Virtual Environment (VE) in a limited tracking space. The Redirected Walking algorithm is a promising solution for this because it attempts to maximize the mapping of the physical space to virtual motions. The physical space in the real world act as inputs to the Redirected Walking algorithm. Additional physical objects like tables and chairs pose challenges while apply redirection as the user cannot see these objects in the virtual world. However, mapping of physical objects in virtual world while

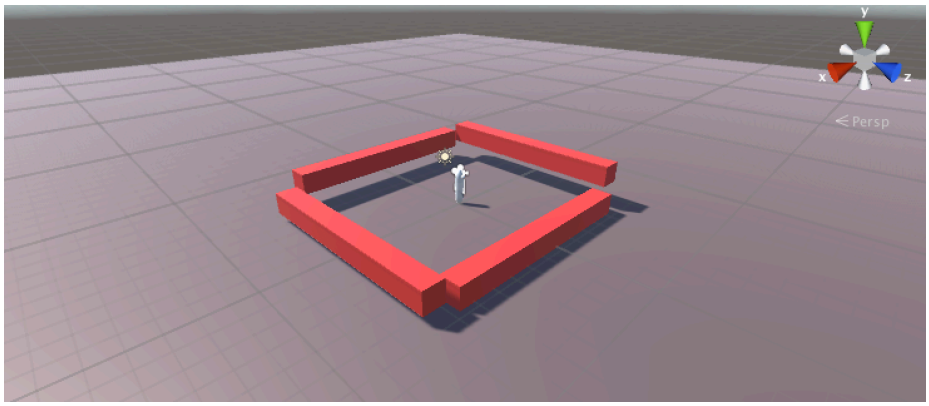

Figure 1.1: A Simple Virtual Environment in Unity

traversing through VR using Redirected Walking is a challenge. This can be achieved by enhancing the visual dominance of the VE. It does, however, hinder the use of passive haptics because it rotates the virtual world relative to the real world.

The idea of using Redirected Walking algorithms to avoid physical obstacles within the tracked space (not necessarily in the virtual space) is the hypothesis of this thesis. As a proof for this hypothesis, the current implementation provides code that has been incorporated in a public available Redirected Walking Toolkit provided by Mahdi A et al. [1] that is capable of avoiding obstacle using the Redirected Walking algorithm's Steer-to-Center condition. Besides this, we also have tested the Magic Barrier Tape technique which has potential to become a resetter within the Redirected Walking algorithms. This work also forms a ground work for testing passive haptics while using Redirected Walking algorithms. The current work done does not provide implementation for multiple physical objects. This can be achieved by implementing a bounding volume hierarchy. Two experiments were conducted during this work. The Redirected Waling algorithm and Obstacle avoidance algorithm were tested in a simulation. A simple pilot study was conducted to test the Magic Barrier Tape.

2 Background

2.1 Locomotion in Virtual Environments

Locomotion is defined as the ability to move from one place to another. It is the key feature for effective, immersive interaction in a VR system. Previously virtual environments have been limited to visual displays combined with mechanical interaction devices or proxy devices for locomotion, e.g., joystick or wand. Recently, walking in VE's by different methods of creating self-motion has become important to improve the naturalness of VR based interaction.In particular, traveling by means of real walking has received much attention due to some cognitive experiments showing that walking has significant advantages over other forms of traveling with respect to user's sense of feeling present in the Virtual Environments. However, while humans navigate with ease by walking in the real world, realistic simulation of natural locomotion is difficult to achieve in Virtual Environments. When walking in the real world, vestibular and visual information create consistent multi-sensory cues which indicate one's own motion, i.e., acceleration, speed and direction of travel. Real walking can be implemented in Virtual Environments by mapping a user's tracked head movements to changes of the camera in the virtual world, e. g., by means of a one-to one mapping. Thus, a one-meter movement in the real world is mapped to a one-meter movement of the virtual camera in the corresponding direction in the VE. While this implementation provides near-natural sensory feedback similar to the real world, it has the drawback that the user's movements are restricted by the limited range of the tracking sensors and limitations of the workspace in the real world. The size of the virtual

world often differs from the size of the tracked workspace so that a straightforward implementation of Omni-directional, unlimited walking is not possible. Various prototypes of locomotion devices have been developed to prevent a displacement during walking in the real world. These devices include Omni-directional treadmills [12], motion foot pads, robot tiles, and motion carpets. Although these locomotion devices represent enormous technological achievements, they are very costly and will not be generally accessible in the foreseeable future

2.2 Previous Work

2.2.1 Redirected Walking

In many virtual environments, it is required to support real world interactions that allow users to move naturally, similar to the way people move in the real world. However, supporting walking is often not feasible because the dimensions of the physical tracked space will ultimately limit the size of the virtual world that may be navigated through natural body movement. To address this problem, researchers have developed redirected walking, a technique that manipulates the mapping between physical and virtual motions to steer the user away from the boundaries of the physical space. Redirected walking is simply applying different strategies for manipulating the user's trajectory to effectively keep them within the bounds of the tracked area. The Redirected Walking technique is discussed in several articles [1, 10, 14, 7, 2].

Redirected walking algorithms are of two types

- Reactive: Reactive algorithms are essentially greedy algorithms that make decisions based on the current state of the user at each point and try to make the optimal choice based on a particular heuristic (e.g. Steer-to-Center)

- Predictive: These algorithms predict the user's path in the virtual environment and use it to plan a redirection movement.

Redirected walking is achieved by manipulating the relation between real and virtual motions by applying gains, or changes to various motions that are possible when we locomote. The different types of self-motion gains are as follows:

- Translation gain: The gain involved in manipulating user's positional motion in 2D world. When this gain is modified, the user's translational motion is changed to either increase or decrease the mapping between physical movements in the tracked space and the related movements in virtual space. For instance, if translational gain is increased the user's physical movement will be scaled up and will give them the impression that they are moving faster in the virtual space. Likewise, if translational gain is decreased, the user will move more slowly in the virtual space as compared with their physical translational motions.

- Rotation gain: Applying manipulation along the user's center of rotation in the virtual world is rotation gain. These gain is applied to the user's head rotation, which changes their perceived rotation in the virtual world. Again, rotational gain changes the mapping between user head rotations and their counterparts in the virtual world. Increasing rotational gain causes the user to rotate faster in the virtual world while decreasing rotational gain causes the user to turn more slowly.

- Curvature gain: These gains involve utilizing slow changes to the user's view rotation while they are translating. In this sense, curvature gain may manipulate both rotation and translation gains. The rotational gains are applied to the virtual environment about the user, and they are applied during translation. Curvature gain is useful for getting the user to walk in a curved path in the physical world while walking along a

straight path in the virtual world. In this way, curvature gain helps to maximize the use of smaller physical tracked spaces.

Redirected Walking is as a promising low-cost solution for enabling large virtual environment exploration via natural locomotion. Two algorithms are discussed and compared in [3] and [5]. In work by Hodgson et al. [5] two algorithms are discussed, Steer-to-Orbit and Steer-to-Center, and compared for their efficiency. The Steer-to-Center algorithm always steers the user towards the center (target), whereas the Steer-To-Orbit always creates new steering targets, all located on an orbit. In the previous research Steer-To-Center algorithm outperformed all other algorithms. The previous trial was a user collecting different objects in a forest. User could walk in any direction. This paper concentrated more on longer and straight roads. The VE of a big grocery store was implemented with 24 parallel aisles, each of 30 m in length. Forty-seven undergraduate students participated in the trial. User had to pick five things from the grocery store and come back to checkout lane. This was conducted twice using both the algorithms. Once the participants were done with the trial, they were asked a couple of questions regarding their experience. There were four metrics for the measurement of efficiency of the algorithms.In this experiment, the Steer-to-Orbit efficiency was much better than Steer-to-Center.The Steer-to-Orbit algorithm performs well in VE having straight lines and orthogonal turns.The Steer-to-Center algorithm performs well in unconstrained VE's. [5] claims that using Steer-to-Orbit algorithm for constrained VE's is more effective. The VE's structure places an important role in the performance of the algorithm. One important observation was that VE caused sickness to users.

2.2.2 Resetter

Resetting includes manipulating the user's position and optical flow in the real world to move them away from a physical obstruction while still allowing them to experience a

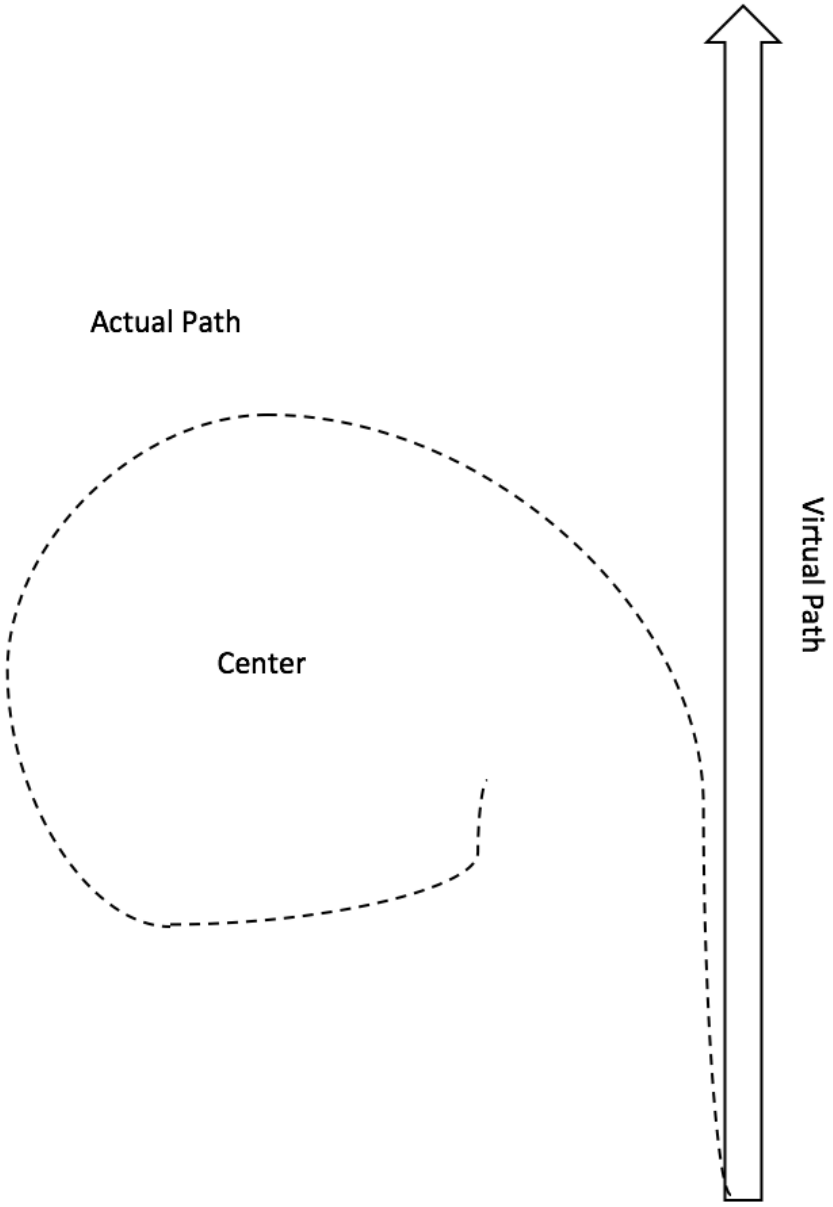

Figure 2.1: Steer-to-Center

continuous sense in the virtual world space. Resetting is used when the user's have reached the limits of tracking space. The resetting algorithm completely relies on the user's ability to accesses the visual information during spatial updating. This technique is also used to overcome the problem of exploring larger virtual environment with limited tracking space.

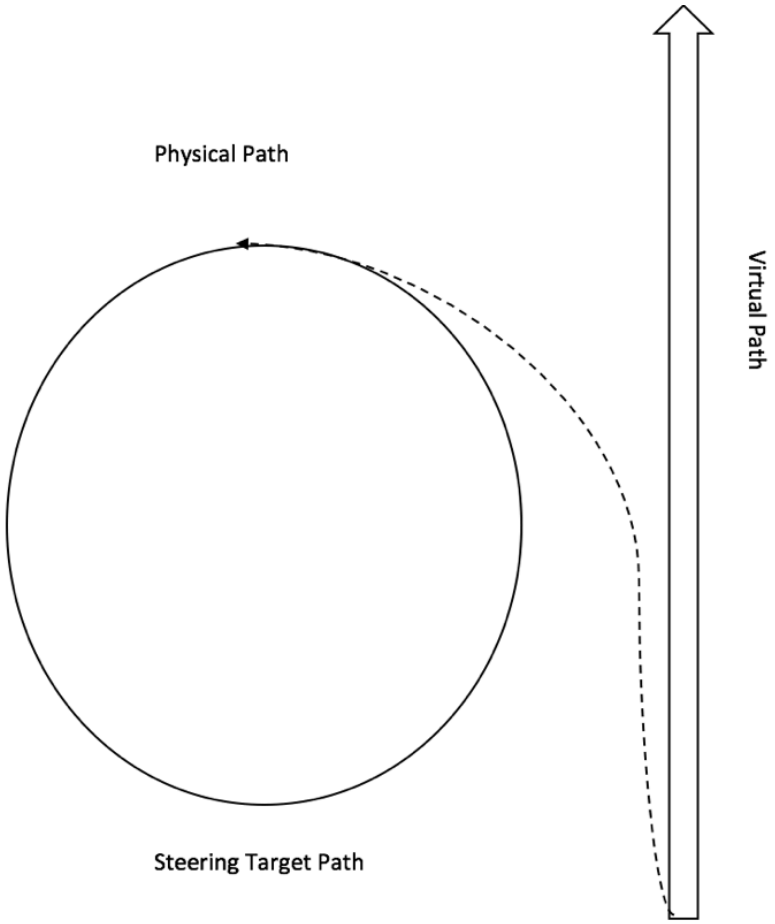

Figure 2.2: Steer-to-Orbit

Previously exploring large virtual space was usually done using a joystick where the user could move in the virtual space just by moving the joystick, but users are more immersed in the virtual world when they explore it by natural walking.

The research paper by Williams et al. [13] describes, the physical position of a user in 3D space using a right-handed coordinate system obtained from the tracking system. The position in the center of the room on the floor is (0, 0, 0). The x, y, and z directions while standing in the center of the room facing to the front correspond to front-to-back movement, user height, and right-to-left movement, respectively. Movement is limited in the x and z directions due to the finite range of the tracking system. Since the y-direction indicates

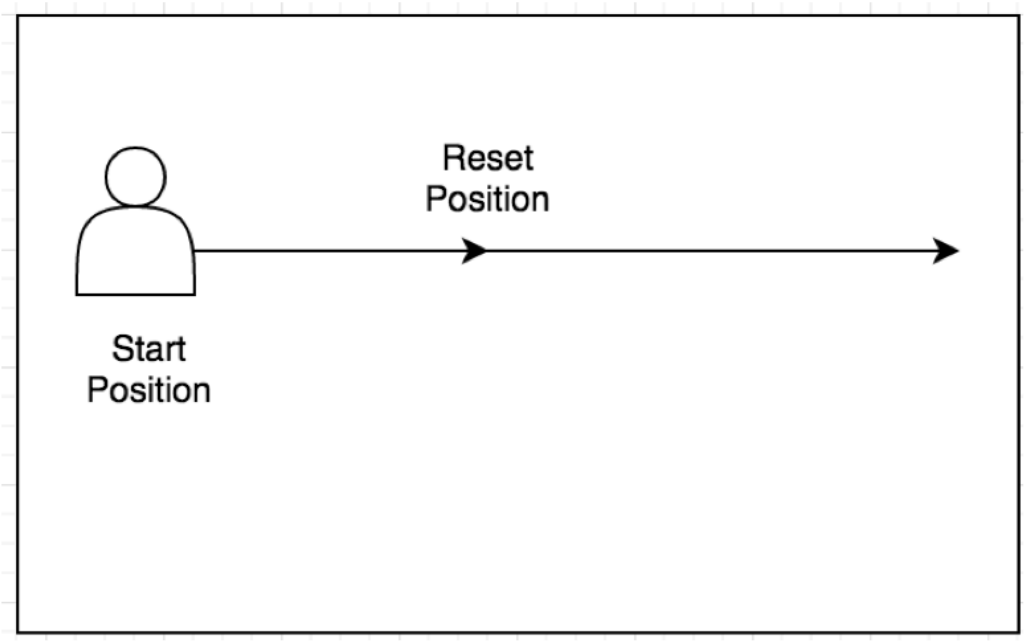

Figure 2.3: Resetter

movement perpendicular to the ground pane, this value typically represents the user's eye height, and does not limit the exploration of the virtual environment. Orientation is obtained from the rotational sensor located on HMD which updates rotation about the x-axis (pitch), y-axis (yaw), and z-axis (roll).

Three types of resetters have been discussed in this paper, Freeze-Backup, Freeze-Turn and 2:1-Turn. These resetters are discussed in detail below

2.2.3 Freeze–Backup

The user obtains more space for virtual exploration by taking some steps backwards while frozen in a fixed position in the virtual environment. In Freeze-backup, the system indicates to the user that they have reached the boundaries of the tracking system either by showing some warning sign or showing a virtual wall in the virtual world space. The

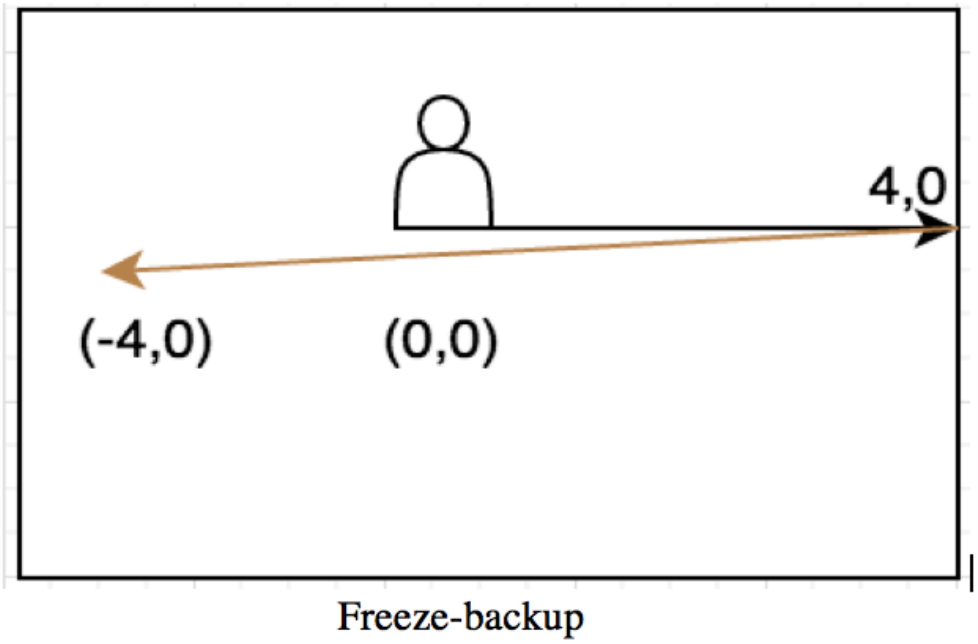

Figure 2.4: Freeze Backup

tracking system is no longer used to update the position of the subject in the virtual environment, so that the user's position in virtual space is no longer updated with movement in physical space. The user is then instructed by the experimenter to take steps backwards in physical space while user's position in virtual space remains fixed. When enough steps are taken, the system indicates the user to stop, the displays are unfrozen, and the user is allowed to continue along the same path that they were walking before the reset. During the backward walking, rotation tracking is active so that the user can look around. The figure below illustrates this method.

2.2.4 Freeze –Turn

In this method, when the tracking system finds that the user has reached a boundary, the system indicates to the user to reset by turning around. The display of the HMD is frozen, freezing the user's position and rotation angle in virtual space, and the user turns

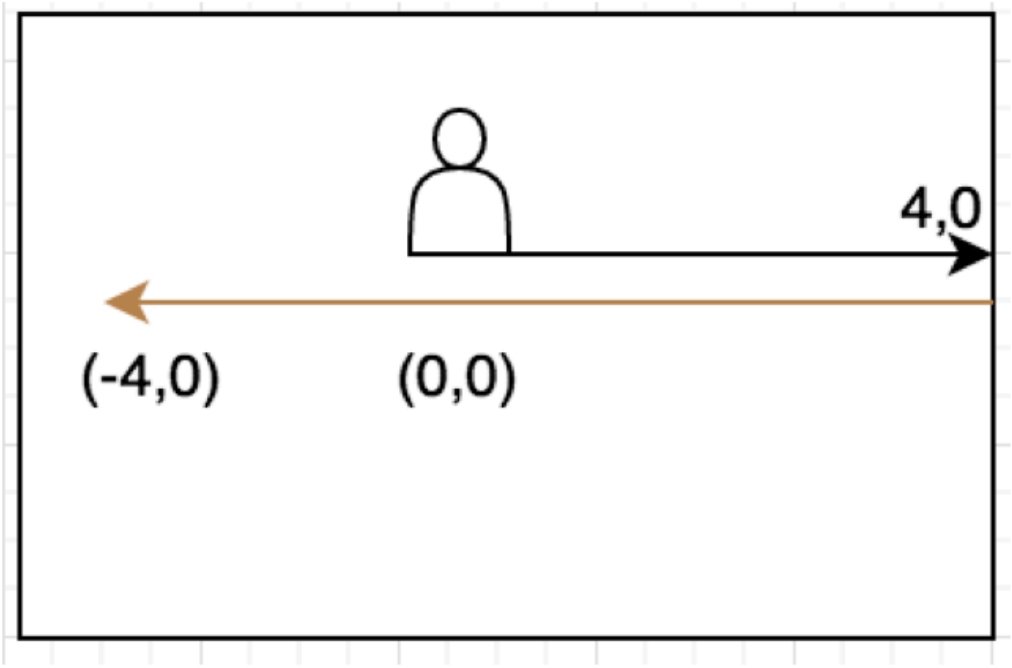

Figure 2.5: Freeze-Turn and 2:1-Turn

180 degrees. In this method both the position and rotation angle is not tracked when the user is outside the tracking system. The display is unfrozen, tracking is updated, and the subject is able to continue traveling along his route.

2.2.5 2:1 –Turn

In this method, when the user reaches the boundaries of the tracking system, the VR system indicates to turn and keep turning until completing a visual full turn in the virtual environment. The rotational gain of the yaw angle during this turn is scaled by two, such that if the user rotates 180 degrees in the physical environment, but actually rotates 360 degrees in the virtual environment.

This research paper by Gabriel Cirio et al. [6] discusses an interactive analogy with

some degree of realism (Magic Barrier Tape) for making locomotion in Virtual Reality (VR) more comfortable. Magic Barrier Tape is a virtual boundary in VR, which actually maps to the real boundary in the real world. The most important constraint while building a VR is limited workspace. With the introduction of Magic Barrier Tape this constraint can be relaxed to some extent. Magic barrier Tape uses hybrid position/rate control mechanism to enable real walking. The most important benefit with this mechanism is that, the user ‘s immersion into VR will not be disturbed.

Two experiments were conducted in Magic tape Barrier. In the first experiment the participant moved from initial position to a central position. The completion time was calculated. They used HMD as display device and a backup with laptop. ART infrared tracking system was used to track participants head position. Participant’s eyes were covered with an opaque cloth so that the surrounding real world could not be seen. The Magic Barrier Tape technique completion time was shorter than the other two techniques. In second experiment, the participant had to follow a route in two virtual walls.The completion time was calculated. Here too the Magic Barrier Tape technique completion time was shorter than the other two techniques. Questionnaires were also provided to participants after the experiments. Magic Barrier Tape technique performed well in both experiments.

The completion time was twice faster than the other techniques. Participants walked less in Magic Barrier Tape technique because there was no resetting of position. This technique was less precise while following a route than other techniques in the experiment second. This technique gave a feel of elastic boundaries.

The research paper by Ruimin ZHang et al. [9] discusses how Minification affects verbal and blind walking judgments in head mounted displays (HMD). Minification causes people to make more accurate distance judgments than they do with normally rendered graphics. Minification also helps to eliminate distance compression without avatars. The researchers conducted two experiments and introduced a new method to calibrate the HMD. A

blind walking experiment was conducted in a dense high fidelity classroom model. The system was calibrated at the start of the experiment. The VR components used here were NVOS nVisor HMD, WorldViz PPT four camera for position tracking and InertiaCube2 sensor to track the orientation of the HMD. In Experiment 1, all the participants were screened for visual acuity and participants with at least 20/30 visual acuity were allowed to participate in the experiment. Twenty-five participants were selected for this experiment. The participants were brought into the laboratory with HMD placed. They would be viewing the virtual world with targets on the ground. They had to view the targets for five minutes and were not allowed to calculate tiles of floor or use any strategy to calculate the distance. Then each participant had to close the eyes and walk to the center of the target with the HMD placed on the head. The same procedure was continued for about fifteen trials with targets at different distances. Audio cues were provided to the participants. Minification affected the distance judgment of the participants. Participants walked of the displayed target distance in minified conditions. In Experiment 2, participants had to verbally report the distance. Twenty-seven participants participated in two conditions: calibrated and minified. The participants had to view the targets as long as they wanted and had to verbally tell the distance to the target. Targets were placed at a similar distance as in first experiment. Minification did not have much effect on verbal reports of distance. Experiment 3 was conducted to verify the system calibration. This procedure was developed to quantify the distance or angle between virtual landmarks to the corresponding real world landmarks. The participants were able to see both the virtual world and real world (transparent configuration). Participants had to indicate verbally or with a laser, the point in real world that would correspond to the center of the target in virtual world. This process was repeated for calibrated and minified conditions. Participants found the virtual horizon to be around 0.81 degrees and 1.33 degrees in calibrated and minified conditions respectively. For distances 2m and 3m, participants judged the distance to be greater than the actual distance whereas

for distances 4m and 5m, the participants judged the distance lesser than the actual distance. The angle of declination affects egocentric distance judgments even if the user can't see the horizon directly. Minification influences the angle of declination depending on how the angle is measured. In above experiments (1 and 2) it was proved that Minification affects distance judgments in sparse classroom. This was proved earlier for hallway room. One important conclusion is that Minification does not influence all distance judgment uniformly like in case of verbal reporting.

The research paper by Nguyen et al. [8] discusses about the possible metrics that affects user judgment of distances in virtual world compared to that in real world. The mass, moment of inertia and restricted Field of View (FOV) are the parameters that are discussed in this paper with respect to HMD. One of the ways to test the correctness of the Virtual Environment (VE) is the blind walking task. Earlier it is proved that humans can judge up to a distance of 25m correctly in real world. HMD system mechanics has an effect on distance judgment in VE. HMD exhibits pressure over the head due to its mass, because of which the user feels a downward pressure. The HMD mass at static conditions generates forces and torques on users. If the center of mass of the HMD is shifted, it exerts extra forces and torques on the neck of the user. This could affect user's distance judgment. The setup was done in 18m by 11m meeting room. NVIS nVisor SX HMD (FOV-47degree horizontal, 38 degree vertical) was used. InterSense IS-900 tracker system was used to track the positions. A mock HMD was used in this task which has similar mass and FOV as real HMD. Inertial headband with the same mass as real HMD but unrestricted FOV was used in this task. This task was tested for three conditions with the real HMD, mock HMD and inertial headband. 116 participants participated in this task. Users were tested for directed and triangulated walking with targets set on ground in VE. Targets were placed at different distances in VE. Four viewing conditions (Real walking, HMD, Mock HMD, inertial headband) were tested. Users were also asked to put in a collar so that they could not see the ground near their feet.

Each participant walked in only one condition. Distances were underestimated when viewing in HMD compared to any other conditions. There was insignificant difference between inertial band walking and real walking. This paper suggests that mechanical aspects alone cannot be the reason for the underestimation of distances in VE. One important fact was that underestimation of distance also happens when the user is allowed with head rotation. The authors concluded that the underestimation of the distances is because of combination of mechanical aspects of HMD and the restricted FOV.

The research paper by Bruder et al. [4] discusses the effects of self-motion illusions created in VR by manipulating the optical flow fields . Self-motion illusions created affect user's self-motion judgment. Optical flow is described as the apparent motion of objects with respect to the user's motion in a scene. Two types of optical flow (expansional, directional) have been implemented to induce illusions in VR. Four types of visual motion illusions have been discussed in this paper:

- Layered Motion: This is the simplest way of creating optic flow cues. VR system displays moving bars, sinus gratings or particle flow fields with strong luminance differences in the background.

- Contour Filtering: In this technique, two images are sampled together using Gaussian and Hilbert transform (time dependent) to form the view.

- Change Blindness: This technique is used to change the scene in VR without causing any disturbance to the user. Based on the calculated blink rates, translation and rotational gains are applied in this technique.

- Constraint Inversion: In this technique, two images and their reversed images are displayed in a loop.

Two blending techniques are discussed in research paper by Steinicke et al. [11]. In

one technique the optical flow was applied on the periphery (peripheral blending) and in the other it was applied to the ground (ground plane blending). Four experiments were conducted. It was conducted in 10m by 7m dark laboratory. Participants had to walk a distance of 2m at some speed to calculate gain threshold. For each trial, participants had to focus eyes on a small crosshair at some height and walk till it turned red. Peripheral and ground plane blending were applied and hence the participant had to perform in two blocks. The four experiments analyzed subject's judgment of self-motion and showed that illusion parameter affected the results. Faster or slower contour motion in the view did affect global self-motion perception. Increasing optical cues, helped the user to perceive virtual motion as in real world. Illusion in the ground was much less distracting than in the entire periphery. By these experiments researchers could prove that illusion affect user's distance judgment. Future work would be to study how space perception is affected by manipulation of virtual translations and rotations.

3 Implementation

The goal of this thesis is to implement Steer-To-Center Redirected Walking algorithm and use this algorithm to avoid potential physical obstacles in the tracked space that the user may not know about while interacting in the virtual world. We have also explored the Magic Barrier Tape technique for future investigation as a resetter within the Redirected Walking algorithm. Prior to the release of USC's Redirected Walking Toolkit, the Steer To Center algorithm was implemented within the Unity Game Engine using C#. Upon release of the Redirected Walking Toolkit [1] in March 2016, the code related to this thesis was migrated to the publically available source of the toolkit. We modified the Steer To Center algorithm within the Redirected Walking Toolkit to be aware of obstacles and redirect the user around these physical constraints. In the following descriptions, the steering parameters used by the redirected walking toolkit were kept constant across all algorithms. Thus, the only differences among different steering choices, such as Steer-to-Center and Steer-to Orbit, lie in where the user is being steered.

3.1 Steer-to-Center Implementation

The Steer To Center algorithm considers the current physical location, linear velocity, and the angular velocity of the user. Based on this input, the algorithm determines whether or not the current steering target should be updated and if necessary it generates or selects a new target. Once the current steering target has been evaluated and possibly updated, the maximum rotation that can be applied is determined based on the state of the user.

In this thesis, the code generates a temporary steering target whenever the bearing-to-center is greater than 160 degrees. Temporary targets are set at a distance of 4m from the user, 90 degrees from the user's current heading, in whichever direction will return the user to the tracking area center soonest. The temporary target is abandoned as soon as the user's bearing-to-center is again less than 160 degrees, and steering instructions again guide the user exclusively towards the center. The complete code is present in the appendix B (section: Steer-to-Center) and represents the code from USC's Redirected Walking Toolkit. Lines 64-90 from Appendix B (section: Steer-to-Center) are listed here to help explain the how the temporary steering targets are generated. Along with this the final rotation is calculated in considering the temporary target position and dampening thresholds. Curvature gain is injected if the rotationFromCurvatureGain is greater than rotationFromRotationGain. The code to calculate these gains is presented in Appendix B (section: SteerToRedirect Lines 46-136).

```
1  if ((bearingToCenter >= S2C_BEARING_ANGLE_THRESHOLD_IN_DEGREE && !
       dontUseTempTargetInS2C))
2  {
3      //Generate temporary target
4      if (noTmpTarget && !avoidObstacle) {
5          tmpTarget = new GameObject("S2C Temp Target");
6          tmpTarget.transform.position = redirectionManager.currPos +
               S2C_TEMP_TARGET_DISTANCE * (Quaternion.Euler(0,
               directionToCenter * 90, 0) * redirectionManager.currDir);
7          Debug.Log("whatever"+tmpTarget.transform.position);
8          tmpTarget.transform.parent = transform;
9          noTmpTarget = false;
10         avoidObstacle = false;
11     }
12     currentTarget = tmpTarget.transform;
13 }
14 else
15 {
16     currentTarget = redirectionManager.trackedSpace;
17     if (!noTmpTarget) {
18         GameObject.Destroy(tmpTarget);
19         noTmpTarget = true;
20     }
21 }
```

3.2 Obstacle Avoidance Implementation

In the obstacle avoidance implementation the code that generates the temporary steering target is different from the Steer-to-Center algorithm. The steering is applied only when the user is very close to the object. Currently we are only checking the distance between the user and the obstacle. For further enhancement, we would be considering the angles between the user's direction and obstacle and using a dot product comparison to activate the steering if the user's heading is within an angular threshold of the obstacle.

Lines 92-105 from Appendix B (section: Steer-to-Center) represent contributions from this thesis and are listed here to help explain how to detect if the user is approaching the obstacle. Line 95 determines the distance between the user's position and obstacle. Line 96 compares the distance with the same threshold that we are temporarily assuming to be a certain number. This threshold can vary. If the distance is less than the threshold we set avoidObstacle to true and return from the method.

```
1 bool nearingObstacle()
2 {
3   float dist = Vector3.Distance(redirectionManager.obstacle.transform.
        position, redirectionManager.currPos);
4   if (dist < 5) {
5       avoidObstacle = true;
6       return true;
7   }
8   return false;
9 }
```

Temporary targets are set at a distance considering both the user and the obstacle, it is set 4 meters away from the user and obstacle, 90 degrees for the current heading, in whichever direction will return the user back to current tracking space area sooner. Along with this, we also deactivated the normal Steer-to-Center redirection when the user is moving, either Obstacle Avoidance implementation or Steer-to-Center is activated. Reference the full code listing in an Appendix B (section: Steer-to-Center). Lines 36-62 from Appendix B (sec-

tion: Steer-to-Center) are my additions and are listed here to help explain how to generates temporary targets when the user is close to the obstacle.

```
1  if (nearingObstacle()) {
2      if (Input.GetKeyDown(KeyCode.S)) {
3          temp_obsc = temp_obsc + 2.0f;
4      }
5      //Generate temporary target
6      if (avoidObstacle) {
7          tmpTarget = new GameObject("S2C Obsctacble Target");
8          tmpTarget.transform.position = redirectionManager.currPos + (
               redirectionManager.currPos-redirectionManager.Obstacle.
               transform.position) + temp_obsc * (Quaternion.Euler(0,
               directionToCenter_t * 90, 0) * temp_t);
9          noTmpTarget = false;
10
11         currentTarget = tmpTarget.transform;
12         avoidObstacle = false;
13     }
14 }
```

3.3 Redirected Tool Kit

In this thesis, we have tested and incorporated the code in a publicly available toolkit (i.e. Redirected Walking Toolkit), which was released in March 2016 at IEEE VR conference [1]. The section below gives a brief explanation of the Redirected Tool Kit.

The Redirected Tool Kit is implemented to provide a general platform to use different redirected algorithms to traverse the user in a limited VR environment. VR is evolving highly at the consumer commodity level. The people who are interested in the VR community are mostly gamers, museums, elementary school students, medical etc. Introducing Locomotion as walking in VR makes it easier and more real. Redirected Tool Kit is an application developed in Unity which can be used by any developer and modify it according to their use. The purpose of the Redirected Tool Kit (RDT) follow:

- To provide VR researchers a basic interface for deploying redirected walking that

doesn't require learning about low level usage.

- The toolkit can be used by the researchers to test new techniques in redirected walking and test it against the previous implemented algorithms.

The Redirected Toolkit implements standard existing redirection algorithms. It is an open source code that is available to the developers. It easy to extend this implementation for different users. This tool provides redirection, reorientation, simulation and analysis tools. The Redirection Algorithms implemented in this toolkit are Steer-to-Center, Steer-to-Orbit and the Zig-Zag technique. The Zig-Zag technique was first introduced by Razzaque in 2001. Each time a participant/user user turned in the virtual world, a negative rotational gain was applied in the system. For example, if the participant/user turns 180 degrees in the real world, it would actually less than 90 degrees in the virtual world, this caused the user to walk back and forth in the real world which formed a zig-zag pattern in the virtual environment. Redirection in the VR system should not be noticed by the user, but previous studies have shown that the manipulation done applying gains is sometimes noticeable by the user's. To overcome this problem, perceptual thresholds have been applied to the gains in this toolkit.

Reorientation is used so that the users do not bump into a physical wall while immersed in the virtual world. This tool kit consists of two components that accomplign the reorientation. The Safety trigger component is used to give warning to the user while approaching the physical wall or while approaching the edge of the tracking system. This is triggered when the user is 0.5 km away from the boundary of the tracking space. The distance between the boundary and safety trigger gives the users time to react for the changes that will happen once the resetters are triggered. Resetters are used to reorient the user once they are very close to the boundary or edge of the tracking system. The current version of the toolkit has implemented 2-1 Turn Resetter which is explained well in the background chapter. The

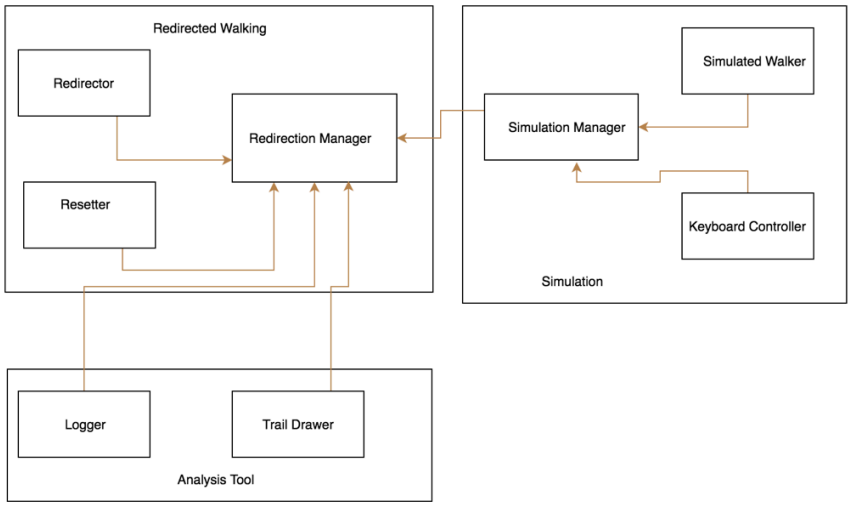

Figure 3.1: Redirected Tool Kit Components

algorithm for 2-1 Turn is explained in the Appendix.

This toolkit supports a component simulation, which traces the positions of the live users in the virtual world. This component consists of two parts, Simulated Walker, which imitates a live user walking in the real world and the Virtual Path Generator component which generates user's path in the virtual world. The toolkit consists of analyzing tools for redirected walking, comparing real and virtual trajectories. This information can be seen in visual forms.

The main components of the Redirected Tool Kit are described below in Figure 3.1

The package is provided in the Unity3D game engine. It consists of a main component, "Redirected User." Redirected User depicts the user's position and orientation. The tracking space has to scaled up to the actual tracking area. The Redirected Tool Kit can be run into three modes Autopilot, Keyboard and Tracker system. The redirection script and resetter script is managed by Redirection Manager. Redirection Manager takes care of the user's position, orientation and it serves as a hub that connects different scripts. It updates the other scripts with the user's position, user's orientation and even tells if the user has reached the boundary.

Figure 3.2: Arrangement of toolkit game objects in scene graph

Figure 3.3 shows how to customize redirection and simulation options by adjusting simple parameters in the Redirected ToolKit.Reference the full code listing in an Appendix B.

The simulation is controlled by Simulation Manager. The Simulation Manager controls the three modes of the toolkit. This component also supports functions for running tests on various conditions. Reference the full code listing in an Appendix B. Analysis tools are provided by Logger and Trail Drawer scripts. These scripts log all the messages while running the tool. Reference the full code listing in an Appendix B .

3.4 Magic Barrier Tape Implementation

Magic Barrier Tape is a virtual boundary in VR, which actually maps to the real boundary in the real world. The most important constraint while building a VR is limited workspace. With the introduction of Magic Barrier Tape this constraint can be lessened to some extent. Magic Barrier Tape uses hybrid position/rate control mechanism to enable real walking. The most important benefit with this mechanism is that the user ‘s immersion into VR will not be disturbed. This concept can be found in common desktop applications and games where the mouse switches to rate control when it reaches the edge of the screen. Inside the workspace, we use position control the user can freely walk, and objects inside the virtual workspace can be reached and manipulated through real walking and real life movements. When reaching the boundaries of the workspace, we switch to rate control the user can move farther in the scene by “pushing” on the virtual barrier tape, hence translating the virtual workspace in the scene. In this thesis, we have tested Magic Barrier Tape.

Reference the full code listing in an Appendix B (section: Magic Barrier Tape).Lines 36-62 from Appendix B (section: Magic Barrier Tape) are listed here to help explain how the Magic Barrier Tape works. When the user is within the bounds(line) , the user is free walking using position control and if the user is getting close to the boundaries either in x or z directions (lines) we start using the rate control. The boundaries can be set by setting the variables.

```
1  void Update ()
2  {
3      curPosition = Tracker.transform.position;
4      velocity = (curPosition - prevPosition) / Time.deltaTime;
5
6      if (curPosition.x <= Min_X_bound && curPosition.x >= ((-1) *
           Min_X_bound) && curPosition.z <= Min_Z_bound && curPosition.z >=
           ((-1)* Min_Z_bound)) {
7          velocity.y = 0;
8          OculusController.transform.position += velocity * Time.deltaTime;
9      } else if (Mathf.Abs (curPosition.x) > Min_X_bound && Mathf.Abs (
```

```
        curPosition.x) < Max_X_bound) {
10      velocity = (curPosition) / (curPosition.magnitude) * 1.4f * Mathf.
            Pow ((Mathf.Abs (curPosition.x) - Min_X_bound), 3);
11      velocity.y = 0;
12      OculusController.transform.position += velocity * Time.deltaTime;
13      boundary_1.transform.position += velocity * Time.deltaTime;
14      boundary_2.transform.position += velocity * Time.deltaTime;
15      boundary_3.transform.position += velocity * Time.deltaTime;
16      boundary_4.transform.position += velocity * Time.deltaTime;
17  } else if (Mathf.Abs (curPosition.z) > Min_Z_bound && Mathf.Abs (
        curPosition.z) < Max_Z_bound) {
18      velocity = (curPosition) / (curPosition.magnitude) * 1.4f * Mathf.
            Pow ((Mathf.Abs (curPosition.z) - Min_Z_bound), 3);
19      velocity.y = 0;
20      OculusController.transform.position += velocity * Time.deltaTime;
21
22      boundary_1.transform.position += velocity * Time.deltaTime;
23      boundary_2.transform.position += velocity * Time.deltaTime;
24      boundary_3.transform.position += velocity * Time.deltaTime;
25      boundary_4.transform.position += velocity * Time.deltaTime;
26  }
27
28  prevPosition = curPosition;
29 }
```

In summary, the Magic Barrier Tape algorithm manifests when the user is near the tracked space boundary. In this way, the user becomes a human joystick. One consideration that this thesis developed while working on the Redirected Walking Algorithms and the Magic Barrier Tape is that the Magic Barrier Tape may be useful as a potential resetting mechanism in Redirected Walking. While not tested in this thesis, the conjecture would be that continued visual flow experienced during the Magic Barrier Tape might be less jarring than the various 2-1 setters.

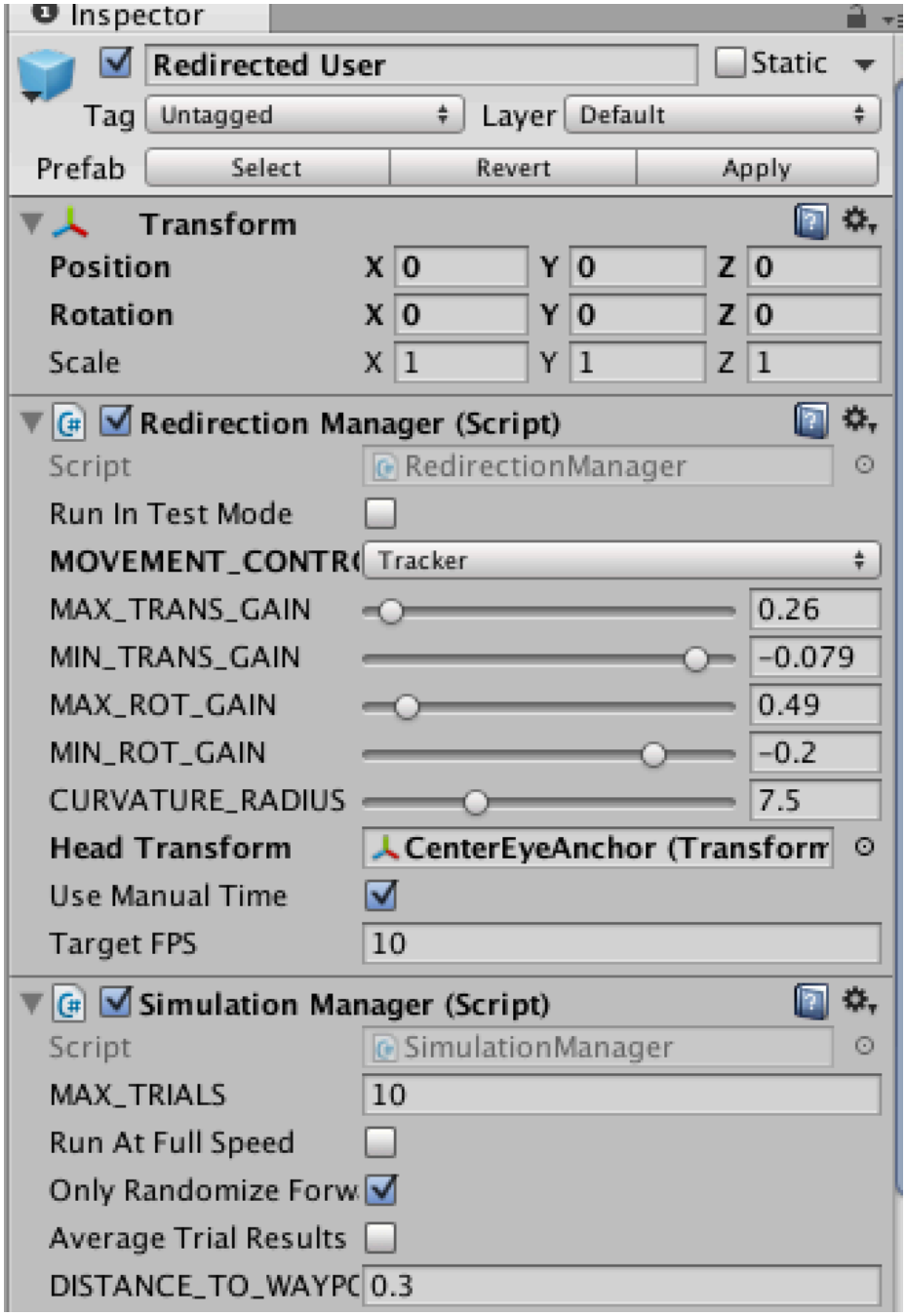

Figure 3.3: Example of customizable redirection and simulation options by simple adjustment parameters in the Unity3D editor

4 Results

This chapter describes the results of the implementation of Redirected Walking algorithm (*e.g.* Steer-to-Center), obstacle avoidance using Redirected Walking algorithm and the Magic Barrier Tape technique. The implementation was incorporated with Redirected Walking Toolkit. Tests were conducted on the Redirected Walking Toolkit simulation. This simulation is discussed in detail in the next chapter. A pilot study was performed in the Motion and Media Across Disciplines (MMAD) LAB and the Simulation and Interaction in Virtual Environments Lab (SIVE Lab) to test the implementation of obstacle avoidance using Redirected Walking algorithm and the Magic Barrier Tape technique.

Experiments were conducted in the laboratory by placing physical obstacles in the path of the users to see how well the algorithm would steer the user around the obstacle, without bumping into the physical object. The Magic Barrier Tape technique was also tested.

4.1 Redirected Tool Kit - Simulation and Results

Experiments simulate user's movement approaching a target (way position). The user's movement is monitored by the character controller in the scene. This simulation can run many trials. The 'Max Trail' variable fixes the number of times the character controller is moving in the VR system . This simulation has an option to run on different Redirected Walking algorithms, different Resetters, different environments, etc. The character controller movement can also be controlled by a variable "fullspeedon". Additionally code was implemented to create an obstacle target prefab (blue sphere) that was placed in the

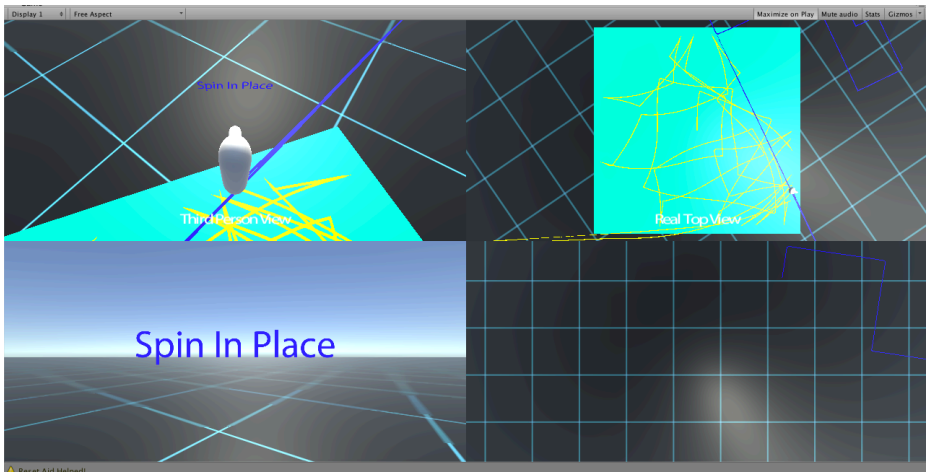

Figure 4.1: Steer-to-Center

center of the tracking space area. This is added to test the obstacle avoidance implementation using the Redirected Walking algorithm.

In the figures, the blue line shows the user's virtual path and the yellow line shows the user's real walking path The below figures depict four different views in the scene: third person view, real top view, first person view and scene view.

Figure 4.1 shows the user's movement while applying Steer-to-Center algorithm. The character controller is simply moving towards a wayposition.

We conducted the experiments in the simulation for different random seeds to place the target in the tracking space area. We conducted five trials for Steer-to-Center algorithm with and without the Obstacle avoidance with the fixed random seed.Below figures represents the real top view for each trial.

Comparing left and right side of figure 4.2 we can conclude that the character controller is deviating while nearing the obstacle and applying Steer-to-Center the Redirected Walking algorithm to reach to the wayposition.

Comparing left and right side of figure 4.3 we can conclude that the character controller is deviating while nearing the obstacle and applying Steer-to-Center the Redirected Walking

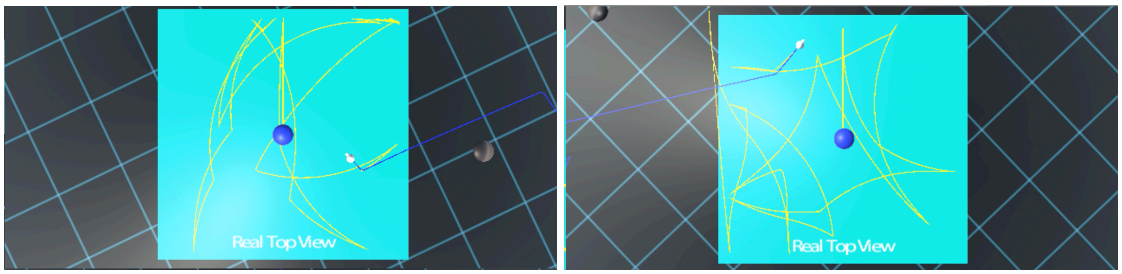

Figure 4.2: Left: Top view of the Simulator without avoidance algorithm with random seed 500. Right: Top view of the Simulator without avoidance algorithm with random seed 500.

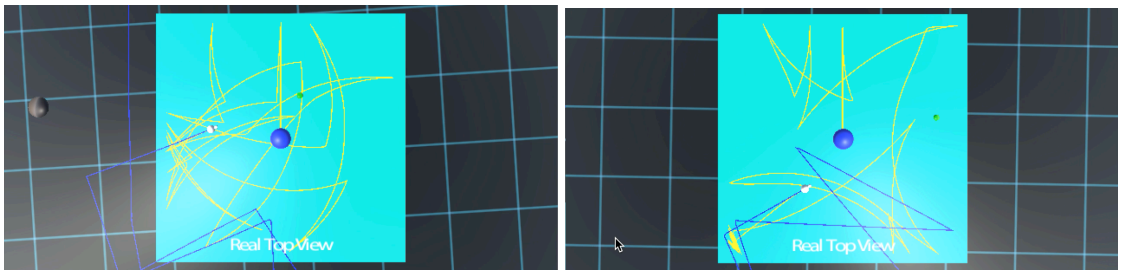

Figure 4.3: Left: Top view of the Simulator without avoidance algorithm with random seed 1500. Right: Top view of the Simulator without avoidance algorithm with random seed 1500.

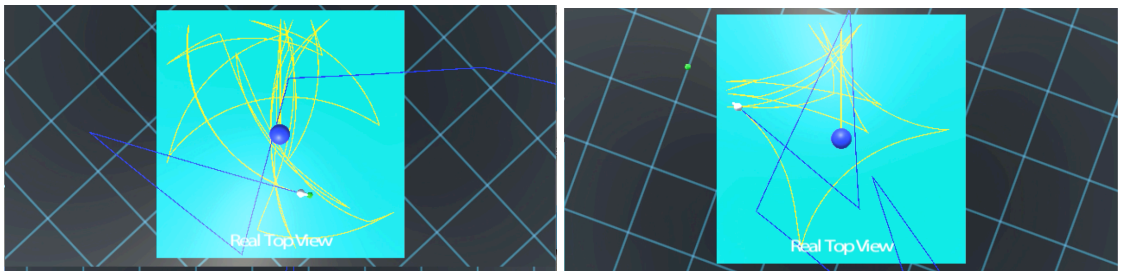

Figure 4.4: Left: Top view of the Simulator without avoidance algorithm with random seed 2500. Right: Top view of the Simulator without avoidance algorithm with random seed 2500.

algorithm to reach to the wayposition.

Comparing left and right side of figure 4.4 we can conclude that the character controller is deviating while nearing the obstacle and applying Steer-to-Center the Redirected Walking algorithm to reach to the wayposition.

Comparing left and right side of figure 4.5 we can conclude that the character controller

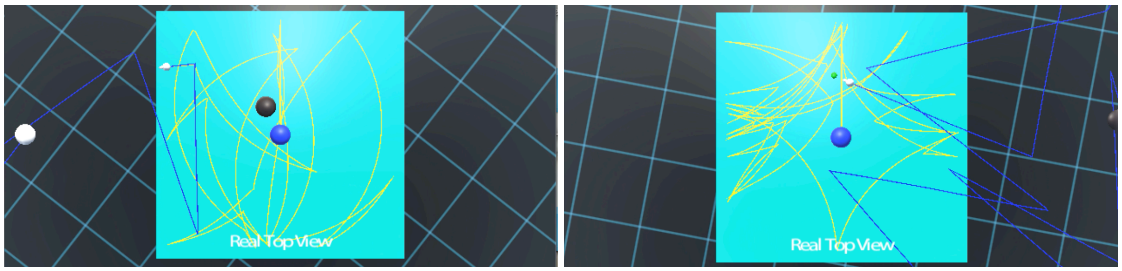

Figure 4.5: Left:Top view of the Simulator without avoidance algorithm with random seed 3500. Right: Top view of the Simulator without avoidance algorithm with random seed 3500.

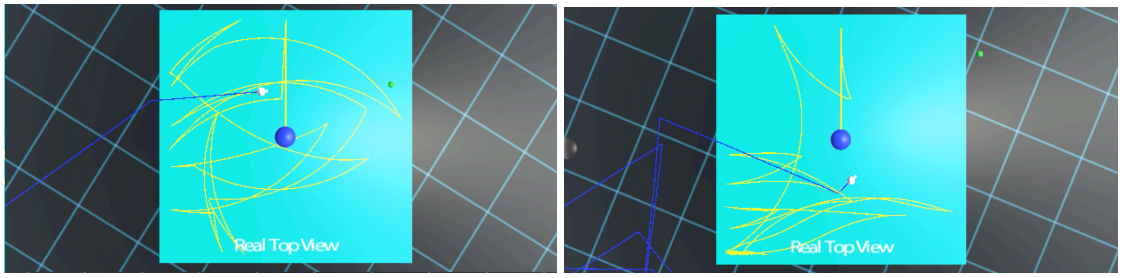

Figure 4.6: Left: Top view of the Simulator without avoidance algorithm with random seed 4000. Right: Top view of the Simulator without avoidance algorithm with random seed 4000.

is deviating while nearing the obstacle and applying Steer-to-Center the Redirected Walking algorithm to reach to the wayposition.

Comparing left and right side of figure 4.6 we can conclude that the character controller is deviating while nearing the obstacle and applying Steer-to-Center the Redirected Walking algorithm to reach to the wayposition. Finally, for most of the cases, the obstacle that was placed in the center of the tracking space areas was avoided by the algorithm. For further enhancement, we need to place the obstacle at different positions and test this algorithm.

Figure 4.7 shows the user's movement while applying Steer-to-Orbit algorithm. The character controller is simply moving towards a wayposition.

Figure 4.8 shows the user's movement while applying the Zig-Zag algorithm. The character controller is simply moving towards a wayposition.

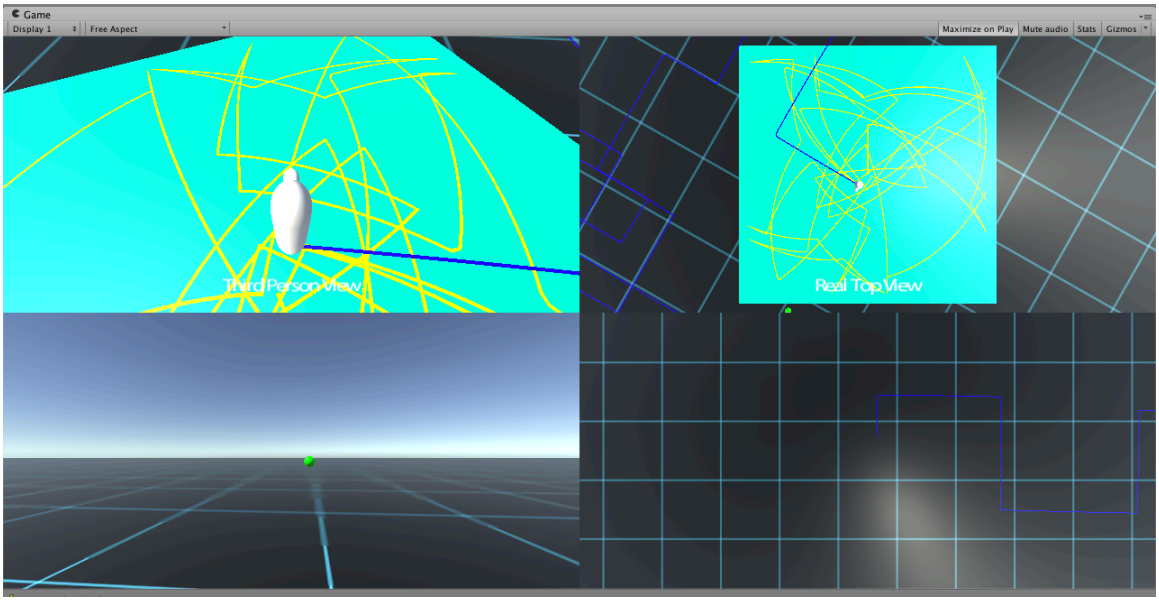

Figure 4.7: Steer-to-Orbit

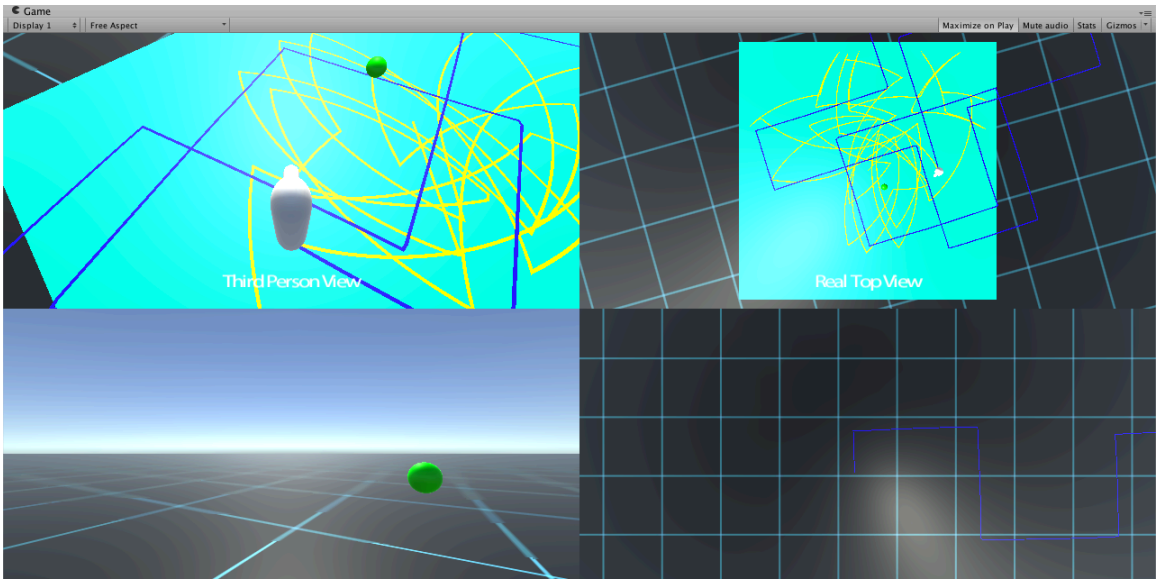

Figure 4.8: Zig-Zag

4.2 Pilot Study and Results

A simple experiment was conducted where the users had to walk from a starting point to an end point. The path consists of a physical obstacle to check the redirection along the

obstacle. When the users reached the end of the tracking system, resetter is activated and the users had to spin in place and come back to the start pointing. We recorded the virtual and real paths for each trial.

4.2.1 Hardware

We used a MAC OSX machine, Vicon Tracker system, and Oculus Rift as a head mounted display in the MMAD Lab. The MMAD Lab consists of 12 camera T-series Vicon motion capture systems with Bonita video camera. Its capures space is approximately 600 square feet. Nexus, Blade, Tracker and Polygon software are also available in the lab. Within the SIVE Lab, we used a WorldViz PPT-H Tracking System. The graphs below represent the user's real and virtual path recorded in the MMAD LAB. Figures 4.9 and 4.10 shows the user's movement while applying Steer-to-Center algorithm without any obstacle. The little jitters in the graph represent the redirection algorithm activating while the user was walking.

Figures 4.11 and 4.12 shows the user's movement while applying Steer-to-Orbit and the Zig-Zag algorithms without any obstacle. The graph represents a Zig-Zag curvature of the user's path in virtual environment

Figure 4.13 shows the user's movement while applying Steer-to-Center algorithm with an obstacle. The blue dotted line in the graph represents the virtual path and green line represents the real path. The little jitters in the graph represents the redirection algorithm activating while the user was walking. The "x" represents the physical obstacle placed in the lab. We can clearly see the jittering in the graph is near the obstacle where in the obstacle avoidance algorithm was activated and the user was redirected from the obstacle.

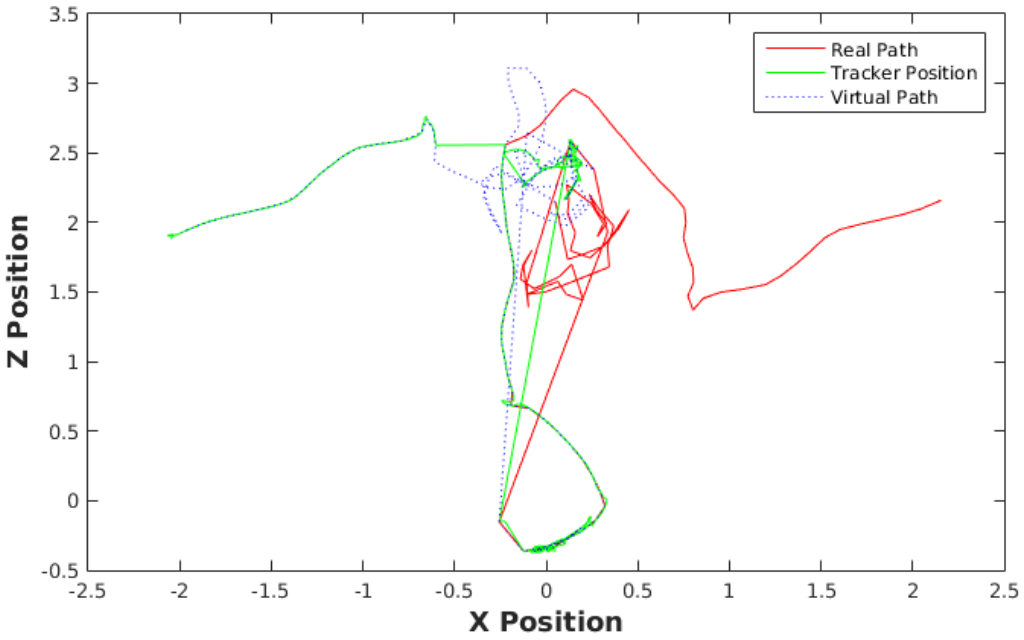

Figure 4.9: Graph depicting a user's path with Steer-to-Center

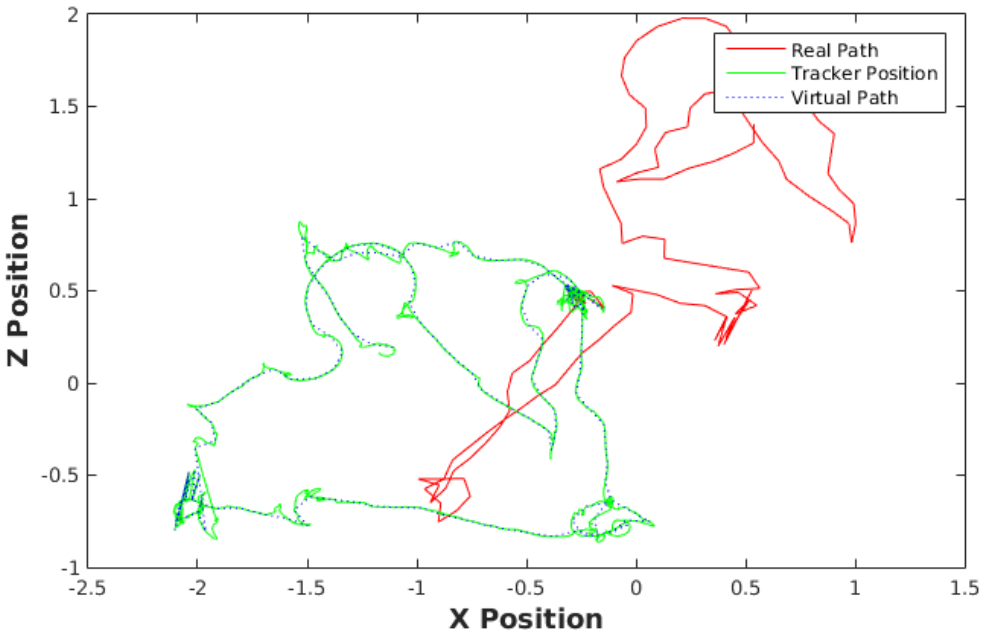

Figure 4.10: Graph depicting a user's path with Steer-to-Center

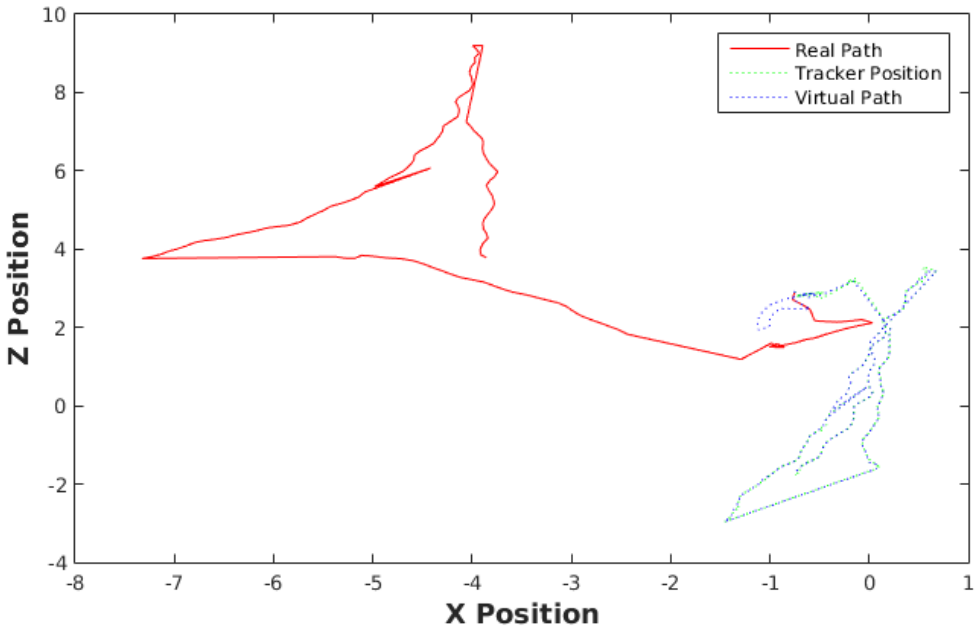

Figure 4.11: Graph depicting a user's path with Steer-to-Orbit

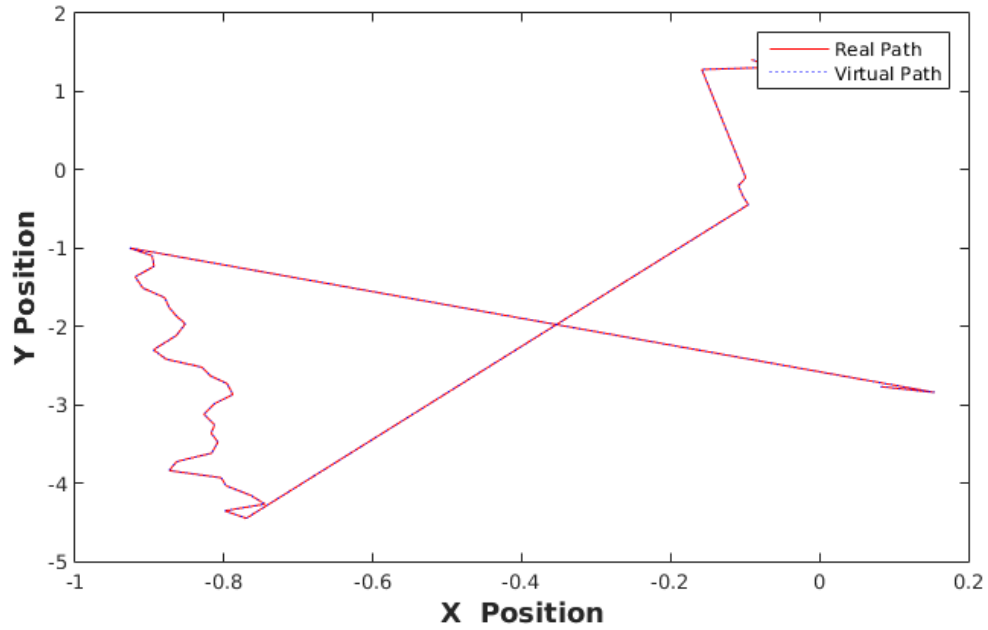

Figure 4.12: Graph depicting a user's path with Zig-Zag

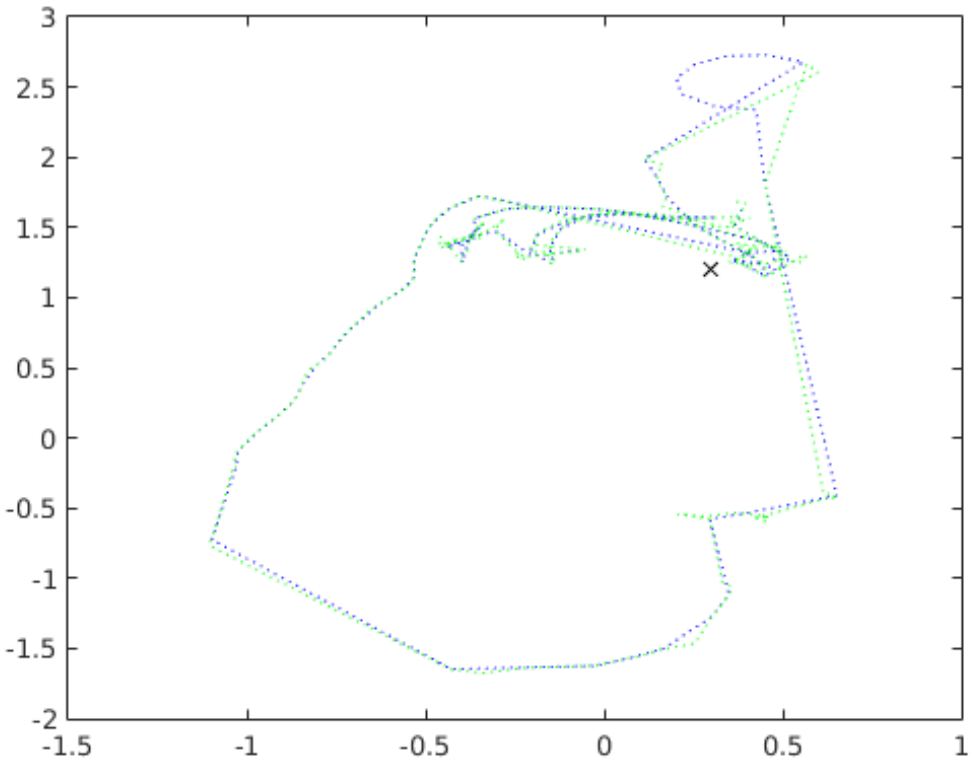

Figure 4.13: Graph depicting a user's path in both real and virtual world avoiding obstacle with the obstacle position

4.3 Magic Barrier Tape Results

Magic Barrier Tape was tested in a different VR System as a single functionality available for exploring larger environments in that VR System. A graph represents the user's position on the Z axis in both Virtual and Real Walking. From the graph its clear that when the user reaches to the boundary, his real position remains same whereas he is moving forward in the virtual world. The velocity in which the user moves once he reaches the boundary is calculated by tracking his hand. Below is the formula for velocity calculation

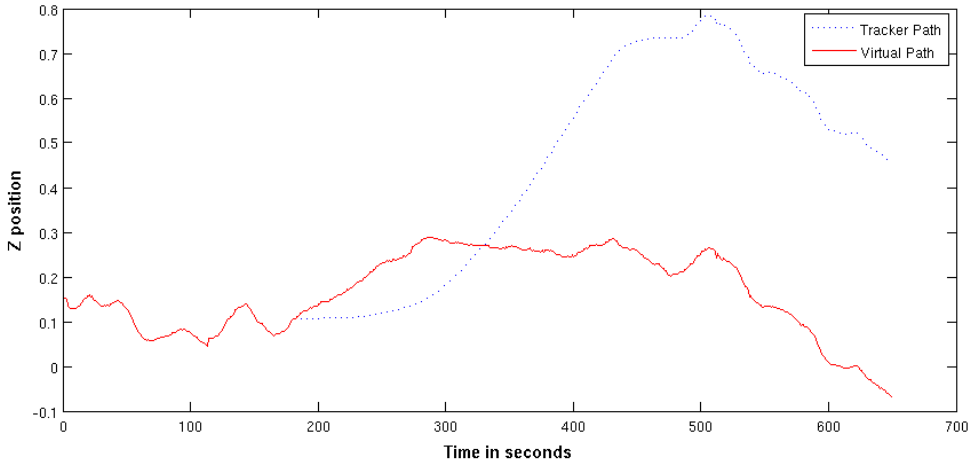

Figure 4.14: Graph depicting a user's path in Z direction for both real and virtual world using Magic Barrier Tape

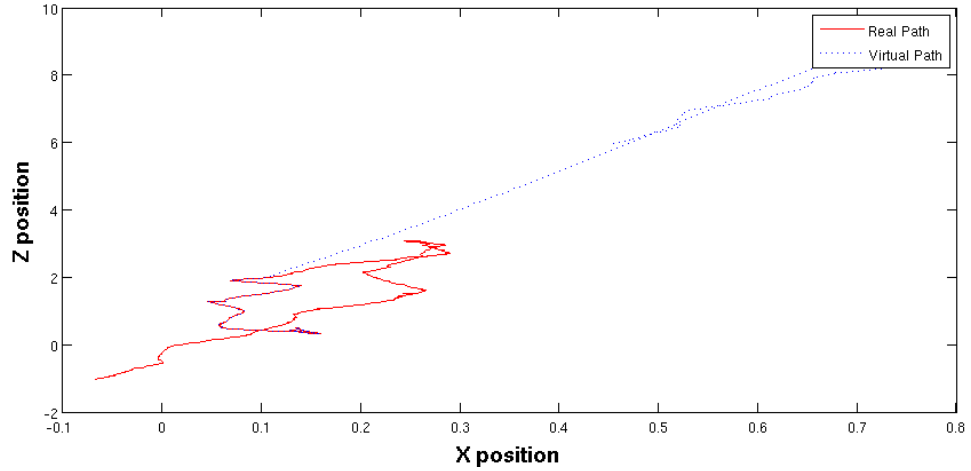

Figure 4.15: Graph depicting a user's full path in both real and virtual world using Magic Barrier Tape

in Magic Barrier Tape technique.

$$
\begin{aligned}
diff =& Mathf.Pow((Mathf.Abs(curPos.x) - TrackSpace.MaxXposition), 3) \\
vel =& \frac{curPos}{curPos.magnitude} * 1.4 * diff
\end{aligned}
\tag{4.1}
$$

5 Conclusions

This thesis presents a novel obstacle avoidance algorithm, which steers the user around obstacles that may cause collision while the user is moving in the VR system. This algorithm leads to robust tracking space area. Without any obstacle avoidance algorithm users are more likely to run into an object, which would break their visual flows in the Virtual environments. With this implementation users can walk naturally with continuous visual flows in the system. This thesis also includes the Magic Barrier Tape technique which is used to explore a larger environment on foot. Magic Barrier Tape is a potential resetter when using Redirected Walking algorithms. It provides users with continuous visual flows. Thus this thesis is a unique combination of Redirected Walking with the Magic Barrier Tape while also dealing with potential collisions between users and physical objects in the laboratory

5.1 Future Work

- This thesis implemented a framework to avoid a fixed physical obstacle in VR system. This can be further enhanced for multiple physical objects. We could form a hierarchical structure of all the obstacles. This structure could be used as reference to steer the user away by considering the whole structure as one big physical obstacle.
- Extending Magic Barrier Tape technique as a resetter for the Redirected Walking ToolKit.

- Further, consider a steering wheel device which is used to drive a vehicle in the virtual world. Initial mapping of physical steering wheel to the virtual steering wheel will be easy considering we know both positions (one to one mapping). If we traverse through the virtual environment using Magic Barrier Tape or Redirected Walking, collocating the virtual steering wheel to physical steering wheel is a challenge. Collocating of the objects can be achieved by manipulating the virtual environment (minification, transitional gains and change blindness).The hypothesis of this work is to collocate the virtual and real objects in virtual world once the user has traversed through the VE using any locomotion technique. Collocating the physical and virtual object would be an interesting idea to take forward.

- Implementation of different Resetters such as Freeze-Backup and Freeze-Turn would be another interesting enhancement to the results of this thesis.

A Appendix A

This describes the algorithms for the Freeze- Back, 2:1 Turn and Freeze-Turn algorithm. The physical path of the user is positioned in x, y, z coordinates. The center of the simulation is considered to be origin (0,0,0). The user can move only in the x and z directions (front and back, left and right). The orientation is captured from the Head Mounted Display.

A.1 Freeze-Backup Algorithm

The first step is to initialize the reset offset to zero.

$$resetOffset_x = 0 \tag{A.1}$$

$$resetOffset_z = 0 \tag{A.2}$$

The user position in the virtual environment is calculated using the adding offset position to the user actual physical position (currPosx, currPosz). The virtual position is represented as vePosx and orientation is represented as veOrix. The current orientation is represented as currOri. The calculation is done using the below formulas.

$$vePos_x = currPos_x + resetOffset_x \tag{A.3}$$

$$vePos_y = currPos_y \tag{A.4}$$

$$vePos_z = currPos_z + resetOffset_z \tag{A.5}$$

$$verOri(x, y, z) = currOri \tag{A.6}$$

The user can freely walk in the space, Once the user has reached the end of the tracking space a message or an image appears to the user requesting to stop walking. The user cannot move, the user's virtual position is fixed, the current location is recorded and the reset offset is calculated accordingly.

$$resetOffset_x = currPos_x + resetOffset_x \tag{A.7}$$

$$resetOffset_z = currPos_z + resetOffset_z \tag{A.8}$$

The user is asked by the experimenter to take steps backwards but the user's position in the virtual world is still not updated. The orientation is not fixed hence the user can look around in space. The new position in the virtual world is calculated as below.

$$vePos_x = resetOffset_x \tag{A.9}$$

$$vePos_y = currPos_y \tag{A.10}$$

$$vePos_z = resetOffset_z \tag{A.11}$$

$$verOri(x, y, z) = currOri \tag{A.12}$$

The user is asked to stop after he comes within the boundaries of the tracking space. The reset is completed and the user can move freely again in the space. The reset is calculated as below.

$$resetOffset_x = resetOffset_x - currPos_x \tag{A.13}$$

$$resetOffset_y = resetOffset_z - currPos_z \tag{A.14}$$

A.2 Freeze –Turn and 2:1 Turn

These two algorithms are almost same with some minor changes which will be discussed in detail in the later section of this appendix.

When the user reaches the boundary of the tracking system, the user turns around with the virtual environment frozen until the user rotates approximately 180 degrees, and then the screen is unfrozen and the user continues along their path. In this algorithm, the y-axis is manipulated. This manipulation is controlled by Thetay, and is initialized with the reset offset in the x (resetOffsetx) and z (resetOffsetz) directions. The two variables rotAxisx and rotAxisz specify the origin of the transformation. Resetting the variables is done by the

following formulas.

$$\theta_y = 0 \tag{A.15}$$

$$rotAxis_x = 0 \tag{A.16}$$

$$rotAxis_z = 0 \tag{A.17}$$

We define a rotation matrix that helps in setting the user's position when the user is not at the boundary of the tracking system. The rotation matrix is represented by R.

$$R = \begin{bmatrix} \cos(\theta_y) & \sin(\theta_y) \\ -\sin(\theta_y) & \cos(\theta_y) \end{bmatrix} \tag{A.18}$$

The rotation and orientation of the user within the boundary is calculated by the following formulas.

$$\begin{bmatrix} vePos_x \\ vePos_z \end{bmatrix} = \begin{bmatrix} currPos_x - rotAxis_x \\ currPos_z - rotAxis_z \end{bmatrix} R + \begin{bmatrix} resetOffset_x \\ resetOffset_z \end{bmatrix} \tag{A.19}$$

$$veOri_x = currOri_x \tag{A.20}$$

$$veOri_y = currOri_y + \theta_y \tag{A.21}$$

$$veOri_y = currOri_y + \theta_y \tag{A.22}$$

$$veOri_z = currOri_z \tag{A.23}$$

When the user is out of the boundary a similar message or an image is shown to the user indicating that the user is out of bounds. The experimenter then asks the user to stop. The user position is frozen in the current position. Calculations are made as below. startAngley represents the manipulation which will be applied in the y-direction. Orientation are calculated as follows.

$$startAngle_y = currOri_y \tag{A.24}$$

$$\begin{bmatrix} resetOffset_x \\ resetOffset_z \end{bmatrix} = \begin{bmatrix} currPos_x - rotAxis_x \\ currPos_z - rotAxis_z \end{bmatrix} R + \begin{bmatrix} resetOffset_x \\ resetOffset_z \end{bmatrix} \tag{A.25}$$

$$rotAxis_x = currPos_x \tag{A.26}$$

$$rotAxis_z = currPos_z \tag{A.27}$$

Reset is applied by updating the Thetay variable that is calculated as below.

$$veOri_x = currOri_x \tag{A.28}$$

$$veOri_y = currOri_y + \theta_y \tag{A.29}$$

$$veOri_z = currOri_z \tag{A.30}$$

$$\theta y = \theta y - (currOri_y - startAngle_y) \tag{A.31}$$

In 2:1 turn, everything remains the same, but the orientation is calculated in a different way. The formula is shown below.

$$verOri_y = (currOri_y - startAngle_y) * 2 + \theta y \tag{A.32}$$

A Appendix B - Code

A.1 Redirection Manager

Redirection Manager code is below:

```
1  using UnityEngine;
2  using System.Collections;
3  using Redirection;
4
5  public abstract class SteerToRedirector : Redirector {
6
7   // Testing Parameters
8  bool useBearingThresholdBasedRotationDampeningTimofey = true;
9
10 bool dontUseDampening = false;
11
12  // User Experience Improvement Parameters
13 private const float MOVEMENT_THRESHOLD = 0.2f;
14
15 private const float ROTATION_THRESHOLD = 1.5f;
16 // degrees per second
17 private const float CURVATURE_GAIN_CAP_DEGREES_PER_SECOND = 15;
18
19 // degrees per second
20 private const float ROTATION_GAIN_CAP_DEGREES_PER_SECOND = 30;
21
22 // Distance threshold to apply dampening (meters)
23 private const float DISTANCE_THRESHOLD_FOR_DAMPENING = 1.25f;
24
25 // Bearing threshold to apply dampening (degrees)
26 private const float BEARING_THRESHOLD_FOR_DAMPENING = 45f;
27
28 // Smoothing factor for redirection rotations
29 private const float SMOOTHING_FACTOR = 0.125f;
30
31  // Reference Parameters
32 protected Transform currentTarget;
33
34 //Where the participant  is currently directed?
```

```
35  protected GameObject tmpTarget;
36
37  // State Parameters
38  protected bool noTmpTarget = true;
39
40  //parameters to avoid obstacle
41
42  protected bool avoidObstacle = false;
43
44  // Auxiliary Parameters
45  //Proposed curvature gain based on user speed
46  private float rotationFromCurvatureGain;
47
48  //Proposed rotation gain based on head's yaw
49  private float rotationFromRotationGain;
50
51  //Proposed rotation gain based on head's yaw
52  private float lastRotationApplied = 0f;
53
54  //method where all the different redirection algorithm
55  //are implemented
56  public override void ApplyRedirection(){
57
58
59      }
60  }
61
62  }
```

A.2 Simulation Manager

Simulation Manager code is below:

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using Redirection;
5
6  public class SimulationManager : MonoBehaviour {
7
8  [HideInInspector]
9  public RedirectionManager redirectionManager;
10
11 //[SerializeField]
12 //bool showUserStartAndEndInLastSnapshot;
13
14 [HideInInspector]
15 public static string commandLineRunCode = "";
16
```

```
17  // Experiment Variables
18  System.Type redirector = null;
19
20  System.Type resetter = null;
21
22  List<Vector3> gainScaleFactors = new List<Vector3>();
23
24  [SerializeField]
25  float MAX_TRIALS = 10f;
26
27  [SerializeField]
28  bool runAtFullSpeed = false;
29
30  [SerializeField]
31  public bool onlyRandomizeForward = true;
32
33  [SerializeField]
34  bool averageTrialResults = false;
35
36  [SerializeField]
37  // Maximum distance requirement to trigger waypoint
38  public float DISTANCE_TO_WAYPOINT_THRESHOLD = 0.3f;
39
40  float zigLength = 5.5f;
41
42  float zagAngle = 140;
43
44  int zigzagWaypointCount = 6;
45
46  float trialsForCurrentExperiment = 5;
47
48  }
```

A.3 Trail Drawer

Trail Drawer code is below:

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using System.Linq;
5  using Redirection;
6
7  public class TrailDrawer : MonoBehaviour {
8   //[SerializeField]
9   LayerMask trailLayer;
10
11  [SerializeField]
12  bool drawRealTrail = true, drawVirtualTrail = true;
```

```
13
14 [SerializeField, Range(0.01f, 1)]
15 float MIN_DIST = 0.1f;
16 [SerializeField, Range(0.01f, 0.5f)]
17 float PATH_WIDTH = 0.05f;
18 const float PATH_HEIGHT = 0.0001f;
19 [SerializeField]
20 Color realTrailColor = new Color(1, 1, 0, 0.5f);
21 virtualPathColor = new Color(0, 0, 1, 0.5f);
22 List<Vector3> realTrailVertices = new List<Vector3>();
23 List<Vector3> virtualTrailVertices = new List<Vector3>();
24
25 [HideInInspector]
26 public RedirectionManager redirectionManager;
27
28 bool isLogging;
29
30 void Awake()
31  {
32     trailParent = new GameObject("Trails").transform;
33     trailParent.parent = this.transform;
34     trailParent.position = Vector3.zero;
35     trailParent.rotation = Quaternion.identity;
36     trailLayer = LayerMask.NameToLayer("Redirection");
37  }
38
39  void OnEnable()
40 {
41       BeginTrailDrawing();
42  }
43
44 void OnDisable()
45 {
46         StopTrailDrawing();
47         if (drawRealTrail)
48             ClearTrail(REAL_TRAIL_NAME);
49         if (drawVirtualTrail)
50             ClearTrail(VIRTUAL_TRAIL_NAME);
51 }
52   public void StopTrailDrawing()
53     {
54         isLogging = false;
55     }
56
57     public void ClearTrail(string trailName)
58     {
59         Transform trail;
60       if ((trail = trailParent.FindChild(trailName)) != null)
61             Destroy(trail.gameObject);
62     }
63 }
```

A.4 SteerToRedirect

```
1  using UnityEngine;
2  using System.Collections;
3  using Redirection;
4  //added by dcost004
5  using System;
6  using System.IO;
7
8  public abstract class SteerToRedirector : Redirector {
9
10     // Testing Parameters
11     bool useBearingThresholdBasedRotationDampeningTimofey = true;
12     bool dontUseDampening = false;
13
14     // User Experience Improvement Parameters
15     private const float MOVEMENT_THRESHOLD = 0.2f; // meters per second
16     private const float ROTATION_THRESHOLD = 1.5f; // degrees per second
17     private const float CURVATURE_GAIN_CAP_DEGREES_PER_SECOND = 15;  //
           degrees per second
18     private const float ROTATION_GAIN_CAP_DEGREES_PER_SECOND = 30;  //
           degrees per second
19     private const float DISTANCE_THRESHOLD_FOR_DAMPENING = 1.25f; //
           Distance threshold to apply dampening (meters)
20     private const float BEARING_THRESHOLD_FOR_DAMPENING = 45f;
21    // TIMOFEY: 45.0f; // Bearing threshold to apply dampening (degrees)
22     private const float SMOOTHING_FACTOR = 0.125f; // Smoothing factor
           for redirection rotations
23
24     // Reference Parameters
25     protected Transform currentTarget; //Where the participant  is
           currently directed?
26     protected GameObject tmpTarget;
27
28     // State Parameters
29     protected bool noTmpTarget = true;
30
31     //parameters to avoid obstacle
32
33     protected bool avoidObstacle = false;
34
35    //added by dcost004
36    string  fileName = "finalpoints.txt";
37
38     // Auxiliary Parameters
39     private float rotationFromCurvatureGain; //Proposed curvature gain
           based on user speed
```

```
40      private float rotationFromRotationGain; //Proposed rotation gain
            based on head's yaw
41      private float lastRotationApplied = 0f;
42
43
44      public abstract void PickRedirectionTarget();
45
46      public override void ApplyRedirection()
47      {
48          //Debug.Log("hello in Applyredirection");
49          PickRedirectionTarget();
50
51          // Get Required Data
52          Vector3 deltaPos = redirectionManager.deltaPos;
53          float deltaDir = redirectionManager.deltaDir;
54
55          rotationFromCurvatureGain = 0;
56
57          if (deltaPos.magnitude / redirectionManager.GetDeltaTime() >
                MOVEMENT_THRESHOLD) //User is moving
58          {
59              Debug.Log("User is Moving");
60              rotationFromCurvatureGain = Mathf.Rad2Deg * (deltaPos.
                    magnitude / redirectionManager.CURVATURE_RADIUS);
61              rotationFromCurvatureGain = Mathf.Min(
                    rotationFromCurvatureGain,
                    CURVATURE_GAIN_CAP_DEGREES_PER_SECOND *
                    redirectionManager.GetDeltaTime());
62          }
63
64          //Compute desired facing vector for redirection
65          Vector3 desiredFacingDirection = Utilities.FlattenedPos3D(
                currentTarget.position) - redirectionManager.currPos;
66          int desiredSteeringDirection = (-1) * (int)Mathf.Sign(Utilities.
                GetSignedAngle(redirectionManager.currDir,
                desiredFacingDirection));
67        // We have to steer to the opposite direction so when the user
              counters this steering, she steers in right direction
68
69          //Compute proposed rotation gain
70          rotationFromRotationGain = 0;
71
72          if (Mathf.Abs(deltaDir) / redirectionManager.GetDeltaTime() >=
                ROTATION_THRESHOLD)  //if User is rotating
73          {
74
75              //Determine if we need to rotate with or against the user
76              if (deltaDir * desiredSteeringDirection < 0)
77              {
78                  //Rotating against the user
```

```
79                  rotationFromRotationGain = Mathf.Min(Mathf.Abs(deltaDir
                        * redirectionManager.MIN_ROT_GAIN),
                        ROTATION_GAIN_CAP_DEGREES_PER_SECOND *
                        redirectionManager.GetDeltaTime());
80              }
81              else
82              {
83                  //Rotating with the user
84                  rotationFromRotationGain = Mathf.Min(Mathf.Abs(deltaDir
                        * redirectionManager.MAX_ROT_GAIN),
                        ROTATION_GAIN_CAP_DEGREES_PER_SECOND *
                        redirectionManager.GetDeltaTime());
85              }
86          }
87
88          float rotationProposed = desiredSteeringDirection * Mathf.Max(
                rotationFromRotationGain, rotationFromCurvatureGain);
89          bool curvatureGainUsed = rotationFromCurvatureGain >
                rotationFromRotationGain;
90
91
92          // Prevent having gains if user is stationary
93          if (Mathf.Approximately(rotationProposed, 0))
94              return;
95
96          if (!dontUseDampening)
97          {
98              //DAMPENING METHODS
99              // MAHDI: Sinusiodally scaling the rotation when the bearing
                    is near zero
100             float bearingToTarget = Vector3.Angle(redirectionManager.
                    currDir, desiredFacingDirection);
101             if (useBearingThresholdBasedRotationDampeningTimofey)
102             {
103                 // TIMOFEY
104                 if (bearingToTarget <= BEARING_THRESHOLD_FOR_DAMPENING)
105                     rotationProposed *= Mathf.Sin(Mathf.Deg2Rad * 90 *
                            bearingToTarget /
                            BEARING_THRESHOLD_FOR_DAMPENING);
106             }
107             else
108             {
109                 // The algorithm first is explained to be similar to
                        above but at the end it is explained like this.
110             // Also the BEARING_THRESHOLD_FOR_DAMPENING value was never
                    mentioned which make me want to use the following even
                    more.
111                 rotationProposed *= Mathf.Sin(Mathf.Deg2Rad *
                        bearingToTarget);
112             }
113
```

```
114
115             // Linearly scaling the rotation when the distance is near
                    zero
116             if (desiredFacingDirection.magnitude <=
                    DISTANCE_THRESHOLD_FOR_DAMPENING)
117             {
118                 rotationProposed *= desiredFacingDirection.magnitude /
                        DISTANCE_THRESHOLD_FOR_DAMPENING;
119             }
120
121         }
122
123         // Implement additional rotation with smoothing
124         float finalRotation = (1.0f - SMOOTHING_FACTOR) *
                lastRotationApplied + SMOOTHING_FACTOR * rotationProposed;
125         lastRotationApplied = finalRotation;
126         if (!curvatureGainUsed)
127         {
128             InjectRotation(finalRotation);
129         }
130         else
131         {
132
133             InjectCurvature(finalRotation);
134         }
135     }
136 }
```

A.5 Steer-to-Center

```
1  using UnityEngine;
2  using System.Collections;
3  using Redirection;
4
5  public class S2CRedirector : SteerToRedirector {
6
7
8      // Testing Parameters
9      bool dontUseTempTargetInS2C = false;
10
11
12     private const float S2C_BEARING_ANGLE_THRESHOLD_IN_DEGREE = 160;
13     private const float S2C_TEMP_TARGET_DISTANCE = 4;
14    float temp_obsc = S2C_TEMP_TARGET_DISTANCE;
15
16     public override void PickRedirectionTarget()
17     {
18         //Debug.Log("Hello in PickRedirectionTarget");
```

```
19
20          System.IO.File.WriteAllText ("trackingpoints.txt", "\n "+
                redirectionManager.currPos.x + "  " + redirectionManager.
                currPos.y
21             + "  " + redirectionManager.currPos.z + ";");
22
23          Vector3 trackingAreaPosition = Utilities.FlattenedPos3D(
                redirectionManager.trackedSpace.position);
24          Vector3 userToCenter = trackingAreaPosition - redirectionManager.
                currPos;
25
26          redirectionManager.Obstacle.transform.position =
                trackingAreaPosition ;
27          Vector3 userToObsCenter = trackingAreaPosition -
                redirectionManager.Obstacle.transform.position;
28          Vector3 temp_t = redirectionManager.Obstacle.transform.position -
                redirectionManager.currPos;
29          temp_t.Normalize ();
30          //Compute steering target for S2C
31          float bearingToCenter = Vector3.Angle(userToCenter,
                redirectionManager.currDir);
32          float directionToCenter = Utilities.GetSignedAngle(
                redirectionManager.currDir, userToCenter);
33          float directionToCenter_t = Utilities.GetSignedAngle(temp_t,
                userToObsCenter);
34          redirectionManager.Obstacle.transform.rotation = Quaternion.
                Inverse (redirectionManager.trackedSpace.rotation);
35
36          if (nearingObstacle ()) {
37              //added by dcost004
38              if (Input.GetKeyDown (KeyCode.S)) {
39                  //Debug.Log ("S key is working");
40                  temp_obsc = temp_obsc + 2.0f;
41              }
42              //Generate temporary target
43              if (avoidObstacle)
44              {
45
46                  tmpTarget = new GameObject("S2C Obsctacble Target" );
47                  tmpTarget.transform.position = redirectionManager.currPos +
                        (redirectionManager.currPos-redirectionManager.Obstacle.
                        transform.position) + temp_obsc * (Quaternion.Euler(0,
                        directionToCenter_t * 90, 0) * temp_t);
48                  //tmpTarget.transform.parent = transform;
49                  noTmpTarget = false;
50                  Debug.Log ("okay whatever" + tmpTarget.transform.position);
51                  //System.IO.File.AppendAllText ("obstacle.txt", "\n The
                        Obsctacle Position is" + redirectionManager.Obstacle.
                        transform.position.x + " " + redirectionManager.Obstacle
                        .transform.position.y + " " + redirectionManager.
                        Obstacle.transform.position.z );
```

```
52
53              //System.IO.File.AppendAllText ("obstacle.txt", "\n The
                    Temporary target Position is" + tmpTarget.transform.
                    position.x + " " + tmpTarget.transform.position.y + " "
                    + tmpTarget.transform.position.z);
54
55              currentTarget = tmpTarget.transform;
56              //GameObject sphere = GameObject.CreatePrimitive(
                    PrimitiveType.Sphere);
57              //Destroy(sphere.GetComponent<SphereCollider>());
58              //redirectionManager.S2C_Obstacle.transform.position=
                    tmpTarget.transform.position;
59              avoidObstacle = false;
60
61          }
62      }
63
64      else if ((bearingToCenter >= S2C_BEARING_ANGLE_THRESHOLD_IN_DEGREE
            && !dontUseTempTargetInS2C))
65      {
66
67          //Generate temporary target
68          if (noTmpTarget && !avoidObstacle)
69          {
70              Debug.Log("in if clause");
71              tmpTarget = new GameObject("S2C Temp Target");
72              tmpTarget.transform.position = redirectionManager.
                    currPos + S2C_TEMP_TARGET_DISTANCE * (Quaternion.
                    Euler(0, directionToCenter * 90, 0) *
                    redirectionManager.currDir);
73              Debug.Log("whatever"+tmpTarget.transform.position);
74              tmpTarget.transform.parent = transform;
75              noTmpTarget = false;
76              avoidObstacle = false;
77          }
78          currentTarget = tmpTarget.transform;
79      }
80      else
81      {
82          //Debug.Log("Hello in else clause");
83          currentTarget = redirectionManager.trackedSpace;
84          if (!noTmpTarget)
85          {
86              GameObject.Destroy(tmpTarget);
87              noTmpTarget = true;
88          }
89      }
90  }
91
92  //added by dcost004
93  bool nearingObstacle()
```

```
94      {
95          float dist = Vector3.Distance(redirectionManager.obstacle.
                transform.position, redirectionManager.currDir);
96        if (dist < 5)
97          {
98              avoidObstacle = true;
99              return true;
100         }
101
102         return false;
103     }
104
105 }
```

Bibliography

[1] M. A. ``The Redirected Walking Toolkit: A Unified Development Platform for Exploring Large Virtual Environments". In: (2016), pp. 41–48 (cit. on pp. 2, 4, 17, 20).

[2] G. Bruder, V. Interrante, L. Phillips, and F. Steinicke. ``Redirecting walking and driving for natural navigation in immersive virtual environments". In: *Visualization and Computer Graphics, IEEE Transactions on* 18.4 (2012), pp. 538–545 (cit. on p. 4).

[3] G. Bruder, P. Lubas, and F. Steinicke. ``Cognitive resource demands of redirected walking". In: *Visualization and Computer Graphics, IEEE Transactions on* 21.4 (2015), pp. 539–544 (cit. on p. 6).

[4] G. Bruder, F. Steinicke, P. Wieland, and M. Lappe. ``Tuning self-motion perception in virtual reality with visual illusions". In: *Visualization and Computer Graphics, IEEE Transactions on* 18.5 (2012), pp. 1068–1078 (cit. on p. 15).

[5] T. T. E Hodgson E Bachmann. ``Performace of Redirected Walking Algorithms in a Constrained Virtual World". In: *IEEE Conference* 20.1 (Apr. 2014), pp. 579–587 (cit. on p. 6).

[6] A. L. Gabriel Cirio Maud Marchal. ``The Magic Barrier Tape:a Novel Metaphor for Infinite Navigation in Virtual Worlds witha a Restricted Walking Workspace". In: *ACM Symposium on Virtual Reality Software and Technology* 20.2 (Nov. 2009), pp. 155–162 (cit. on p. 11).

[7] C. T. Neth, J. L. Souman, D. Engel, U. Kloos, H. H. Bülthoff, and B. J. Mohler. ``Velocity-dependent dynamic curvature gain for redirected walking". In: *Visualization and Computer Graphics, IEEE Transactions on* 18.7 (2012), pp. 1041–1052 (cit. on p. 4).

[8] T. D. Nguyen, C. J. Ziemer, T. Grechkin, B. Chihak, J. M. Plumert, J. F. Cremer, and J. K. Kearney. ``Effects of scale change on distance perception in virtual environments". In: *ACM Transactions on Applied Perception (TAP)* 8.4 (2011), p. 26 (cit. on p. 14).

[9] J. W. Ruimin ZHang Anthony Nordman. ``Minification affects verbal- and action-based distance judgments differently in head-mounted displays". In: *ACM Symposium on Applied Perception* 9.3 (July 2012) (cit. on p. 12).

[10] F. Steinicke and G. Bruder. ``Using Perceptual Illusions for Redirected Walking". In: *IEEE computer graphics and applications* 1 (2013), pp. 6–11 (cit. on p. 4).

[11] F. Steinicke, G. Bruder, and S. Kuhl. ``Realistic perspective projections for virtual objects and environments". In: *ACM Transactions on Graphics (TOG)* 30.7 (2011), p. 112 (cit. on p. 15).

[12] Z. Wang, K. Bauernfeind, and T. Sugar. ``Omni-directional treadmill system". In: (2003), pp. 367–373 (cit. on p. 4).

[13] B. Williams, G. Narasimham, B. Rump, T. P. McNamara, T. H. Carr, J. Rieser, and B. Bodenheimer. ``Exploring large virtual environments with an HMD when physical space is limited". In: (2007), pp. 41–48 (cit. on p. 8).

[14] R. Zhang and S. A. Kuhl. ``Flexible and general redirected walking for head-mounted displays". In: (2013), pp. 127–128 (cit. on p. 4).