PLANNING REDIRECTION FOR DYNAMIC PASSIVE HAPTICS USING MODEL PREDICTIVE CONTROL

By

ROHIT NUTALAPATI

Bachelor of Technology in Computer Science

GITAM University

Vishakhapatnam, Andhra Pradesh

2015

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2017

PLANNING REDIRECTION FOR DYNAMIC PASSIVE HAPTICS USING MODEL PREDICTIVE CONTROL

Thesis Approved:

ACKNOWLEDGEMENTS

I would like to thank Dr. Mayfield for all his valuable inputs in this research and for guiding me through two years of my studies. I’ve had the opportunity to learn a lot under him both in computer science and being a professional computer scientist. With his help, I was able to experience, learn and implement immersive technologies.

I would also like to express my gratitude to my thesis committee: Dr. David Cline and Dr. Douglas Heisterkamp for their involvement in this work.

I thank my friends Saiprasanna Palle and Ashwini Jinka for their stimulating discussions and constructive criticism.

Lastly, I would like to thank my Family and friends for their unhindered support and belief in me.

Name: ROHIT NUTALAPATI

Date of Degree: DECEMBER, 2017

Title of Study: PLANNING REDIRECTION FOR DYNAMIC PASSIVE HAPTICS USING MODEL PREDICTIVE CONTROL

Major Field: COMPUTER SCIENCE

Abstract: Navigating an immersive, virtual environment (VE) is one of the key challenges when a head-mounted display is used. The most natural way of navigating a virtual environment would be walking. But walking in a large virtual environment is practically impossible unless the user is in an equally large, walkable real environment (RE), i.e. a one-to-one mapping between real and virtual worlds. A promising solution for navigating large virtual spaces in a limited real space, such as a room, is Redirected Walking.

Redirected Walking (RDW) uses so-called Redirection Techniques (RETs) to guide the walker away from obstacles in the real world. These techniques modify the mapping between VE and RE depending upon user movements in real-time.
Hence, a point in a real environment can be mapped to just about any point in a virtual environment at a particular time. This makes the task of redirecting the user toward an object in the real environment that serves as a proxy for an object in the virtual environment a much more complex problem. This problem ultimately boils down to creating a dynamic map between selected real and virtual world points, i.e. an entire one-to-one map between real and virtual environments is not needed. This dynamic map applies redirections not just to avoid obstacles but also to redirect the user such that whenever the user reaches for an object in the virtual world, he/she senses the proxy object in the real world.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

With the recent explosion of virtual reality research and development, and an increase in consumer-level, head-mounted displays (HMD) on the market, immersive virtual environments (VEs) are becoming more viable. These VEs are a way to give the user a feeling of presence in any world imaginable. The ultimate goal of any such immersive system is to simulate a completely believable environment that is reactive to all the human senses. Two key aspects to achieving an immersive user experience are the ability to navigate through and to touch objects in the VE.

NAVIGATION

Early methods of navigating a VE included joysticks, keyboard & mouse, etc. These methods allowed users to navigate endless virtual spaces while staying put in their real environment (RE). However, these had the drawback of creating limited immersiveness in such systems. More recent methods have adopted head-mounted, stereoscopic displays and body tracking so that the user movements are utilized to manipulate what they see accordingly. While body tracking improves the illusion of virtual presence, it makes navigation a challenging task because of two main reasons: The first is keeping constant track of the user's movements through the RE, and the second is when there is a significant difference between the sizes of the VE and the RE.

Whenever the RE is larger than or the same size as the VE, a one-to-one map can be created from the VE to the RE; i.e. every point in the VE corresponds to a unique point in the RE.

However, if the VE is larger than the RE, then a problem arises; it will be unavoidable that some points in the VE map to a single point in the RE; this is an example of the pigeonhole principle. To address this problem, techniques such as teleportation to a different location in the VE can be adopted. While these methods allow the user to navigate a seemingly endless VE, walking is still the most natural way of navigating, and is a perfect haptic sensation for traversing the VE. Research suggests the use of redirected walking (RDW) to allow the user to navigate a VE that is much larger than an RE [1].

Current HMDs (e.g., the HTC Vive) prevent the user from seeing the RE. A person trying to walk a straight line blindfolded typically ends up traversing a curved path, because he/she is unable to judge his/her direction [2]. HMDs present the user with a virtual environment, and – like a blindfold – prevent the user from seeing the RE. RDW takes advantage of this inability of the user to perceive the RE. Since the user is completely relying on the VE projected in the HMD to navigate, it is possible to influence his/her walking direction by gradually manipulating the projected orientation of the VE, in real-time. If these reorientations are too small to be noticed by the human consciousness, they help preserve a sense of immersiveness. But as we shall see later, one drawback is that RDW induces a non-deterministic, dynamic mapping from the VE to the RE.

Nescher, Huang, and Kunz [3] propose a state-of-the-art algorithm to modify RDW as an optimization problem to better preserve immersiveness. Their algorithm introduces a so-called planning framework to take into account all the user's possible future paths and calculate the best redirection that can be applied.

TOUCH

Passive haptics are an easy way to provide a sense of touch for static objects in a VE [4] by using objects in the RE as proxies for the virtual objects; i.e. when the user reaches out to touch an object in the VE, they actually are reaching out to touch an object in the RE. To create a precise sense of touch for an object in the VE, the physical object needs to overlap perfectly with it. Hence, a precise map is needed from the VE to the RE for this object.

COMBINED APPROACH

Navigation through walking used in conjunction with passive haptics is easy to achieve when the RE has a greater or equal area as compared to the VE, as a proxy object can be placed in the RE to represent an object in the VE according to a one-to-one mapping from the VE to the RE. But if the area of the VE is greater than that of the RE, using RDW to allow the user to navigate the VE induces non-determinism in the mapping, which contradicts one of the requirements for precise, passive haptics, i.e. a precise mapping between the real and virtual worlds. This research addresses the non-determinism with a method to preserve maps from objects in the VE to their proxies in the RE; for the purposes of this work, it is assumed that there is one virtual object that needs to be mapped to a proxy object to provide passive haptic feedback.

Using the algorithm of Nescher, Huang, and Kunz, it is possible to keep track of the user's potential future paths. This can be used to redirect and position the user at the right place for meaningful haptic feedback. This method is implemented by keeping track of offsets, as described in Chapter III.

CHAPTER II

LITERATURE SURVEY

Research suggests that real walking is the best way to navigate in a VE because it provides vestibular and proprioceptive feedback to the user [5]. Razzaque initially proposed RDW and described the basic steering techniques, also known as redirection techniques (RETs) [1]. Typically, to allow a person to walk in an immersive, virtual world, his/her movement is tracked and his/her virtual avatar is modified accordingly. The idea of RDW is to add additional movements, also known as gains to the avatar such that the user is unable to perceive them; these additional movements steer him/her away from obstacles. Razzaque implemented this original idea using rotational RET, i.e. the original rotations (turns) made by the user were scaled to keep user within a tracked space. This idea was developed further by Williams [6], who introduced transitional RET, which scales the user's linear movement. These transitional gains allowed the user to travel a much lesser/greater distance as compared to the actual distance traversed in the VE.

In any RET, it is important to predict the user's next possible path accurately so that an appropriate amount of redirection can be applied. Otherwise, a false redirection may occur and lead the user to collide with an obstruction instead of avoiding it. Interrante [7] proposed a hybrid approach to predict robustly the user's future path; this approach takes into account the user's gaze direction and his/her previous path over a particular time frame.

Though the user is redirected constantly away from obstacles, it is possible to reach a point where there is an unavoidable collision in the course. To address these situations, Williams suggested “reset” techniques, which reorient the user towards a particular steer point [8]. These reset techniques prompt the user to stop walking and reorient himself/herself in his/her current location, usually by turning around. These reset techniques were improved further by Peck [9] as distractors. A distractor attracts the user’s attention such that the user turns around, which then are used to add additional rotations (rotational RET).

A taxonomy of available RETs was provided by Steinickle et al. [10] defining rotation, curvature, and translation gains.

A gain is the quotient of the amount of movement made by the user in the RE and the corresponding movement applied in the VE. It is important to understand the types of gains available to get a clear idea about it.

A rotation gain, is the fraction of VE rotation compared to its corresponding RE rotation.

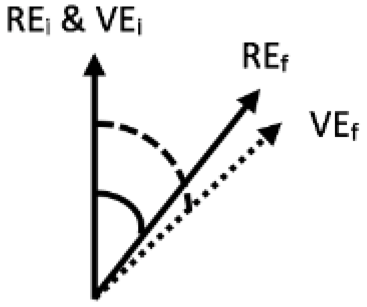


Figure 2.1: Rotational Gain

For example, consider a user whose initial, forward direction in the VE and the RE are given by VE_i and RE_i, respectively, as shown in Figure 2.1. Now, assume the user turns an angle of R_{real} (represented by solid arc) and his/her forward direction is modified to RE_f in the RE. The angle of the corresponding turn made by the user in the VE is $R_{virtual}$ (represented by dashed arc) and his/her forward direction is modified to VE_f. The rotational gain ρ_q is

$$\rho_q = R_{virtual}/R_{real}. \tag{2.1}$$

Similarly, a translation gain is the fraction of the VE translation compared to its corresponding RE translation.

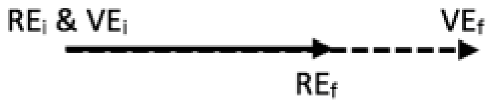


Figure 2.2: Translational Gain

As shown in Figure 2.2, consider a user's translation from RE_i to RE_f, which are T_{real} units apart in the RE and the corresponding translation from VE_i to VE_f, which are $T_{virtual}$ units apart in the VE. The translation gain ρ_t is the quotient of $T_{virtual}$ and T_{real} [10] i.e.,

$$\rho_t = T_{virtual}/T_{real}. \tag{2.2}$$

A curvature gain is a special case, applied when the user is moving straight without any rotations. Steinickle et al. state that "if the injected manipulations are reasonably small, the user will unknowingly compensate for these offsets resulting in walking a curve". A curvature gain implies the bend in the real path. The curve is determined by a circular arc of radius r. The curvature gain ρ_c is defined as [10]

$$\rho_c = 1/r. \tag{2.3}$$

Curvature and rotational RETs are closely related, as both modify the rotation of the user. At each time-step it is possible to apply either one of them. The one to apply is decided simply by selecting the maximum between them [1, 3], as the goal is to induce the maximum amount of redirection.

Hence, additional rotation added per time-step is given by [3]

$$\Delta\hat{\phi} = max \begin{cases} v_R \rho_c \Delta t \\ \omega_R \rho_q \Delta t \end{cases}, \tag{2.4}$$

where

$\Delta\hat{\phi}$ is the angle added per time-step,

v_R is the tangential velocity of the user in the RE,

ω_R is the angular velocity of the user in the RE,

ρ_c and ρ_q are the curvature and rotation gain given in equations 1, 3, and

Δt is the frame rate.

An RET is applied only as long as it is unperceivable by the user; this implies that there is a limit to the degree of redirection that can be applied while preserving immersiveness. These limits of applicable redirection were also estimated by Steinickle et al. [10, 11].

Once the RETs are defined and their limits are established, proper methodologies have to be developed to apply RETs; these methodologies are called *steering* or *redirection algorithms*. These algorithms typically assume that the tracked space is void of obstacles, except for the walls surrounding the tracked space.

The *steer-to-center algorithm* (S2C) [12] is one such methodology that applies RETs so that the user always is redirected towards the center of the tracked space, as the probability of collision over the next movement is least at the center. Three other algorithms, including *steer-to-circle* (which is a general improvement of S2C), are discussed by Hodgson [13]. In *steer-to-circle*, the redirection is made to keep the user in a circular path around the center of the tracked space. Zmuda et al. introduced their FORCE algorithm, which uses probabilistic planning and terminal state evaluation to apply redirection [14].

The MPCRed algorithm proposed by Nescher et al. [3] implements the steering problem as an optimization problem. Here, the user's future path is examined to find the minimal set of applicable RETs along this path that will result in the smallest possible redirection. This smallest possible redirection is evaluated by assigning costs to each of the RETs. The costs are assigned based on the priorities of the RETs; for example, a *reset* RET might be assigned a cost of 500, whereas a rotational RET might be assigned a cost of 5; clearly, this states that a *reset* needs to be avoided much more than a rotation. The map of a VE is represented as a graph, where nodes represent junctions and branches represent corridors in the space. At each point, the user's possible future paths are calculated using this graph and appropriate redirection is applied.

In order to predict the user's possible future position when a rotational RET is applied, Nescher et al. developed an arc-length, parametrized equation with parameter s, which is

$$\gamma_R(s) = \frac{1}{\kappa_R}\begin{bmatrix} sin(\Theta_{R0} + \kappa_R s) - sin(\Theta_{R0}) \\ cos(\Theta_{R0}) - cos(\Theta_{R0} + \kappa_R s) \end{bmatrix} + p_{R0}\,, \tag{2.5}$$

where

$s \in$ [0, length of the curve],

κ_R is a gain parameter, which depends on rotation and curvature gains to be applied,

Θ denotes the orientation of the user, and

p_{R0} is the initial position.

The time dynamic system given by Nescher et al. [3] is defined as

$$x_{k+1} = f_k(x_k, u_k, w_k) \; , \tag{2.6}$$

where x_k is a set of user's current properties including position orientation angular velocity etc. and u_k is the applied RET and w_k is the noise/uncertainty of the system.

As stated earlier, each of the RETs is defined by a cost. These costs help determine the best possible applicable redirections at each stage to get the best possible result. A sum of all the redirections (G) applied along a path is calculated as

$$G = g_N(x_k) + \sum_{k=0}^{N-1} g_k(x_k, u_k, w_k) \; , \tag{2.7}$$

where $g_k(x_k, u_k, w_k)$ is the cost at stage k.

The goal here is to minimize G to get the minimum cost along that path. Hence, the dynamic programming algorithm is given as

$$J_k(\mathrm{x}_k) = \; min_{u \in U_k} \mathbb{E}_{w_k}[\mathrm{g}_k(\mathrm{x}_k, \mathrm{u}_k, \mathrm{w}_k) \; + \; J_{k+1}(\mathrm{x}_{k+1})] \tag{2.8}$$

Where J(x) denotes the optimal cost at stage x. This time dynamic system is used to minimize the cost at over the next k steps.

Kohli et al. state that a compulsive, one-to-one mapping is not a necessity for passive haptics [15]. They simplify the problem of providing haptic feedback during RDW by assuming that there is only one object in the VE. It also is assumed that the RE contains a single object, which serves as a proxy for the object in the VE. They further simplified the problem by assuming that the user's path toward the object includes a turn point; This makes it easy to align the virtual object with its physical counterpart. In a way, their assumptions lead to a highly simplified version of the problem being addressed in this research. Haptic Retargeting, developed by Azmandian et al. at Microsoft Labs [16], provides three methods to align objects in the virtual and the real environments. These approaches are similar to RDW as described by Razzaque, but here the redirection is applied in the user's hand movements, when he/she tries to reach for an object in the VE. These redirections then are used to provide haptic feedback that maps multiple, virtual objects to a single, real object.

A mapping from a larger VE onto a smaller RE can be defined such that it provides a "proper folding of large virtual scenes into smaller real scenes" as shown by Sun et al. [17]. They propose a planar mapping between the VE and the RE floor maps. Their method bends the VE floor map to fit inside the given RE floor map, such that the angular and distal distortions are minimized during mapping of these maps.

CHAPTER III

METHODOLOGY

Redirected walking (RDW) works by using redirection techniques (RETs). The main aim of redirection is to permit a user to move about in a limited real environment (RE) while it appears to the user that they are exploring a larger virtual environment (VE). A redirection technique must induce an adequate amount of angular and distal distortion to realize this illusion and keep them within the bounds of the RE.

BASIC MODEL

Assume a VE that is comprised of a maze containing an object. Also assume that this VE is navigated in a smaller RE by means of RDW with the model predictive control redirection (MPCRed) algorithm, which has been modified in this research. The user is represented by an avatar, and the object located in the VE is at a certain distance from the avatar. The user's main task is to navigate the VE to the object and try to touch it. This requires that the user to be able to walk around freely in the VE even though the RE has a smaller area, and when the user touches the virtual object, he/she actually must touch the proxy object in the RE.

The mismatch between the sizes of the VE and the RE is addressed by RDW with the MPCRed algorithm. But to enable passive haptics, placing the proxy object in the RE is a difficult task, because there is no fixed map that can define the user’s position in the RE compared to his/her position in the VE during run-time when RDW is used. The proposed method uses MPCRed for RDW in an attempt to minimize the total degree of redirections and the deviation between the locations of the virtual object and the proxy object.

PROPOSED METHOD

Consider the object in the VE and its corresponding proxy object in the RE as described above. The user follows a virtual path to this object in the VE, which includes a few turns and a few nearly-straight translations.

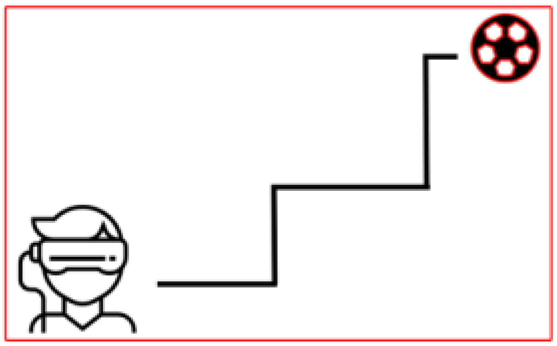


Figure 3.1: Layout of the VE

Figure 3.1 shows the VE: the red outline shows the boundary of the VE, and the football with red outline denotes the virtual object. It also shows a virtual path the user might take to get to the object in the VE; this is denoted by the black stroke.

Figure 3.2 shows the corresponding RE: the green outline represents the boundary of the RE, and the football outlined in green is the proxy object in the RE. Similar to figure 3.1, the real path followed by the user in the RE is denoted by the black stroke. These two figures also show how the user's perceived path in the VE differs from the actual path he/she takes in the RE. The reason the virtual and real paths differ is because of the RETs applied as the user moves.

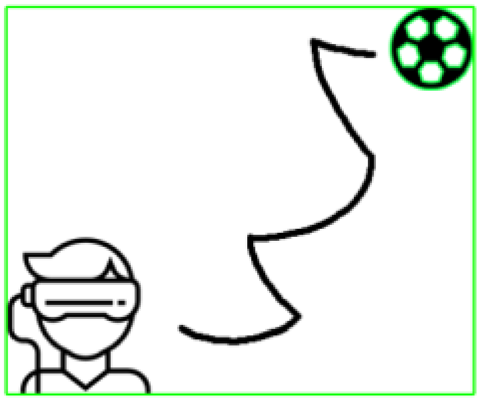


Figure 3.2: Layout of the Corresponding RE

These RETs are the result of the modified MPCRed algorithm, which addresses the mismatch in alignment of virtual object and its proxy, and tries to align their positions as the user get closer to the virtual object. For example, the initial alignment of the VE and the RE from figures 3.1 and 3.2 is shown in figure 3.3.

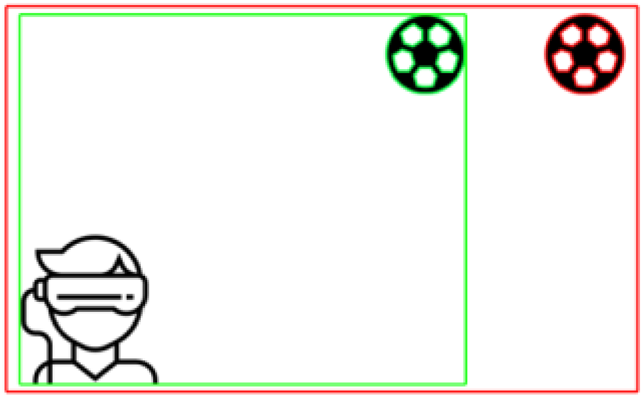


Figure 3.3: Example Initial Alignment of the VE with Respect to the Corresponding RE

Clearly the objects are not aligned and are a certain distance apart from each other. At the end of the traversal, the VE and the RE should be aligned as shown in figure 3.4. If no redirection is applied, the object in the VE and its proxy object in the RE will not be aligned when user reaches the virtual object. And the same holds if MPCRed is applied to minimize redirection. The modified MPCRed algorithm must address this and align the objects using the same redirection techniques.

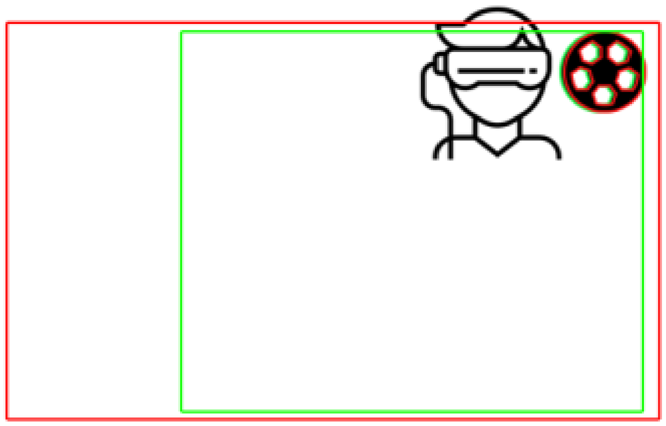


Figure 3.4: Expected final alignment of the VE with respect to the RE

Consider a rectangular-plane as the VE, as shown in Figure 3.1. Assume that this plane has its own coordinate system centered at its mid-point, with the positive x-axis extending to the right, the positive z-axis extending upward, and the positive y-axis extending toward the viewer. This coordinate system is considered to be the *frame of reference (FoR)* of the VE.

Consider another rectangular-plane as the RE, as shown in Figure 3.2. The mid-point of this rectangular-plane is considered to be the *center* of the RE. This plane also has its own coordinate system similar to the one above, and this coordinate system is considered to be the *FoR* of the RE.

When a head-mounted display is used to navigate the VE, the RE overlaps with the VE. In this overlap, the orientation of the RE with respect to that of the VE can be determined by comparing the coordinate system of the RE in the *FoR* of the VE, and vice versa.

RDW modifies the orientation of the VE with respect to the RE. However, the modifications are made only to the x-axis and z-axis translations, and the y-axis rotations. This is because of the fact that both the VE and the RE are assumed to be horizontal, flat, and coplanar surfaces. Hence, the y-coordinate of the VE remains constant, as do the x-axis and z-axis rotations of the VE. Since these values are constant, it is reasonable to exclude them from the calculations given below. So, for example, the center of the VE within the *FoR* of the RE is represented as (x, z) instead of (x, y, z), because y does not change at any stage of navigation.

Assume that the object in the VE initially is centered at (x_v, z_v) and its proxy object is centered at (x_r, z_r), both in the *FoR* of the VE. Similarly, assume the object in the VE has a rotation θ_{yv} and its proxy object has a rotation θ_{yr}, both in the *FoR* of the VE. When RDW is applied, (x_v, z_v) and θ_{yv} are constant since they do not change in the VE; but (x_r, z_r) and θ_{yr} change continuously and dynamically as the user moves. So, the translational offset between the virtual object and its proxy in the *FoR* of the VE is given by

$$O_t = \sqrt{(x_r - x_v)^2 + (z_r - z_v)^2}, \qquad (3.1)$$

and the *rotational offset* is given by

$$O_r = |\theta_{yr} - \theta_{yv}|. \qquad (3.2)$$

For objects that have line symmetry about an axis that passes through their midpoint, such as a torus lying on the x-z-plane with the y-axis passing through the hole in its middle, the rotational offset always can be viewed as zero. This is because there is no difference in the shape of the object no matter how it is rotated about that axis. This research focuses on objects with line symmetry along the y-axis, so the rotational offset need not be considered.

If the path the user takes from his/her initial position to the object in the VE consists of n straight-line walks and $n - 1$ turns, the offsets must be distributed among the redirections applied during these $2n - 1$ transitions, so that when the user reaches the end of the final transition, O_t is equal to 0 and O_r is equal to 0; i.e., the position and orientation of the object in the VE is equal to the position and orientation of its proxy.

Hence, the problem becomes one of keeping track of the user's path in the VE and adjusting it with respect to the *FoR* of the RE such that he/she is at the desired position in the RE at the end of the path. A set of redirections to apply along the VE path has to be selected from all possible redirections in order to minimize the offset between the end points of the paths in the RE and VE. This is similar to the time dynamic model, i.e., the MPCRed algorithm defined by Nescher, Huang, and Kunz [3]. Their model works to minimize redirections by minimizing the overall cost for redirections along any path. This algorithm is modified in this research to also minimize the overall offset at the end of user's path, thus overlapping the virtual and proxy objects to the best extent possible.

The original MPCRed Algorithm deals with minimizing redirection for any path traversed by the user. In this research however, the user is assumed to walk towards a virtual object; this research also assumes that user will be able to touch and feel the virtual object by means of a real proxy object. The costs proposed in the original MPCRed algorithm are inadequate to accommodate these changes because the mapping of the objects in real time require a continuous cost.

For example, consider the scenario shown in figure 3.5.

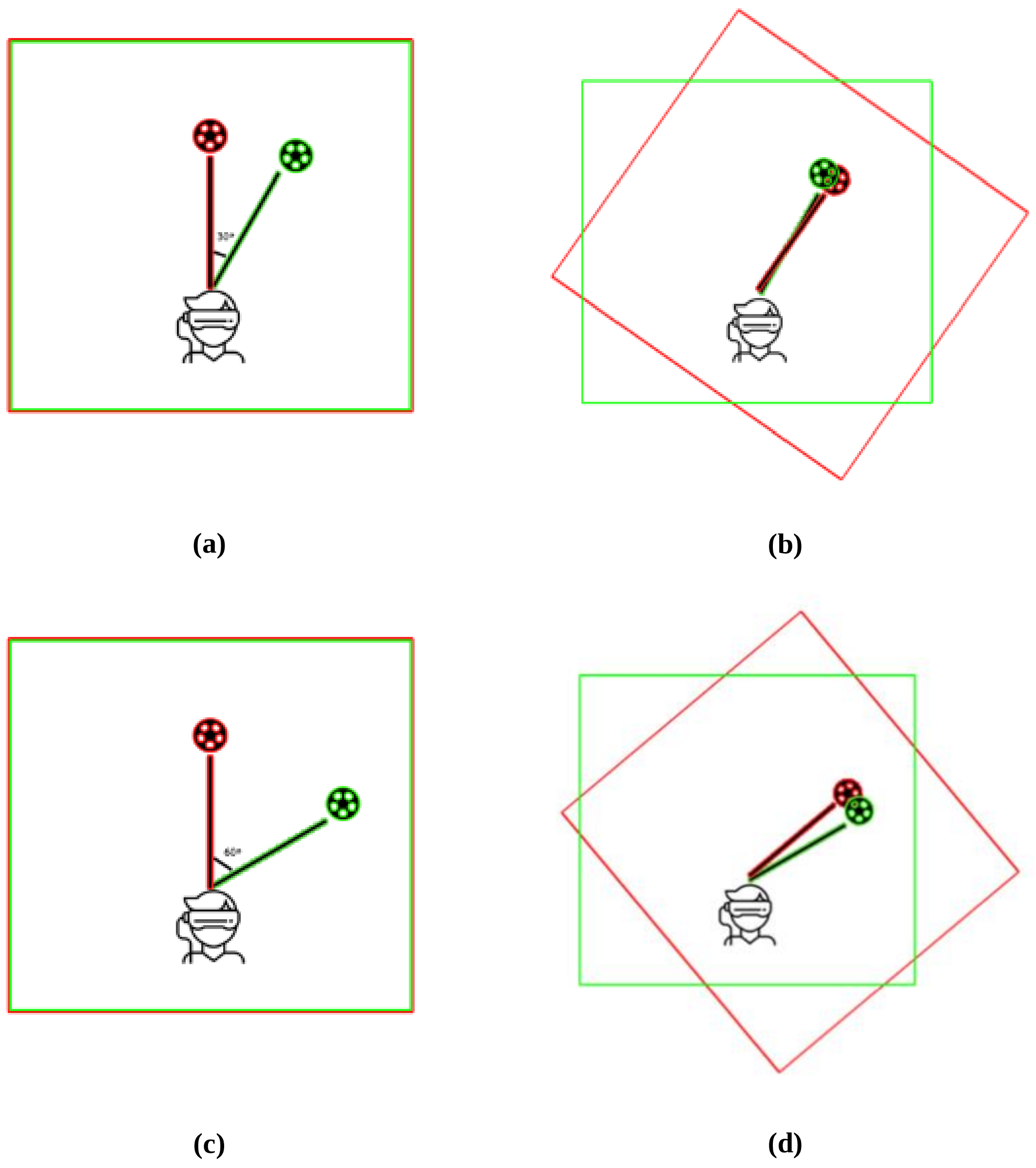


Figure 3.5: Example of how preferability of RETs differ during runtime

In the case shown in figure 3.5(a), the user has a straight path to the virtual object, which is at some distance from the user in the VE. In the RE, the proxy object is also the same distance from the user along a straight-line path. But the difference is, the user's forward direction in the VE is at a 35° angle from the straight-line path to the proxy object in the RE. As described in Chapter 2, reset RETs are defined at 30° increments, e.g., a 30° reset RET, a 60° reset RET, and so on.

Therefore, a 30° clockwise (CW) reset RET would minimize the offset as compared to a 60° or greater CW reset RET.

But, if the objects were aligned at a 50° difference, as shown in figure 3.5(c), a 60° reset RET would be the most desirable choice because it would minimize the offset between the object in the VE and its proxy in the RE, as compared to other reset RETs, as shown in figure 3.5(d). This shows that the choice of RET changes dynamically depending on the offsets. So, the costs associated with different RETs must change dynamically, depending on the current offset as the user moves. And these costs should aptly represent the likelihood that a given RET will be chosen.

To resolve this, a new dynamic cost is introduced. The dynamic cost is calculated based on the original costs from the MPCRed Algorithm, which present a measure of redirection, and a new continuous cost introduced in this research, which presents the measure of overlap between the virtual object and its proxy. These two costs are weighted based on the distance between the user and the virtual object; a final dynamic cost is calculated as a sum of the two weighted costs. The weight of the original MPCRed costs is increased and the weight of the new continuous cost is decreased as the distance between the user and the virtual object increases, and vice-versa; i.e., when user is closer to the virtual object, the default costs of RETs mentioned from the original MPCRed algorithm are assigned lower importance than the continuous cost, as the goal is to achieve offsets of zero.

CALCULATION OF DYNAMIC COST

For an RET R, the dynamic cost of that RET, $DC(R)$ is calculated based on the cost of that RET from the original MPCRed algorithm, $C_r(R)$ and the continuous cost that denotes how much that RET minimizes the offsets is $C_o(R)$.

Let the cost of the RET from the original MPCRed algorithm be k. Then

$$C_r(R) = k. \tag{3.3}$$

For minimizing offsets, the costs associated with an RET vary depending upon how much it minimizes the offset. The lower the offset caused by the RET, lower its cost must be. Therefore, the cost is directly proportional to O_t. Since we assume the virtual object and its proxy to be line symmetrical about y-axis passing through their midpoint, O_r is always 0 and can be ignored. Thus, the continuous cost can be calculated as

$$C_o(R) \propto O_t \tag{3.4}$$

or

$$C_o(R) \; = \; P \; \times \; O_t, \tag{3.5}$$

where P is a large constant cost. Varying the value of P can help control the final cost for offset-minimizing redirections, which helps to control the distance at which the minimizing redirection is given more importance over minimizing offsets. An example of this is explained later.

Each RET has only one cost, which encompasses both of the above characteristics to a degree that depends upon the distance between the user and the virtual object, as mentioned before. A costing function can be defined that gradually slides from one minimization factor to other depending upon distance, so that the redirections are more well-distributed. The influence of C_r and C_o on final cost depends upon the distance from the user to the virtual object, d. Hence d is used as a weighting factor.

$C_o(R)$ is inversely proportional to d i.e.,

$$C_o(\mathrm{R}) \propto {}^{1}\!/_{d} \tag{3.6}$$

or

$$C_o(R) = P \times O_t \times {}^{1}/_{(d+1)}. \tag{3.7}$$

Equation 3.7 implies that $1/(d+1)$ part of the final cost of an RET is the cost of minimizing offsets, and the rest $(1 - 1/(d+1))$ is the cost of minimizing redirection. This assumes that adding 1 does not make much difference because the size of the VE is extremely large compared to 1 unit, but it keeps the cost from being negative when distance is less than 1.

Therefore, the part of $C_r(R)$ that is included in the dynamic cost is given by

$$C_r'(R) = k * (1 - {}^{1}/_{(d+1)}) \tag{3.8}$$

Hence, the total dynamic cost, DC(R) is given by,

$$DC(R) = C_r'(R) + C_o(R) \tag{3.9}$$

or

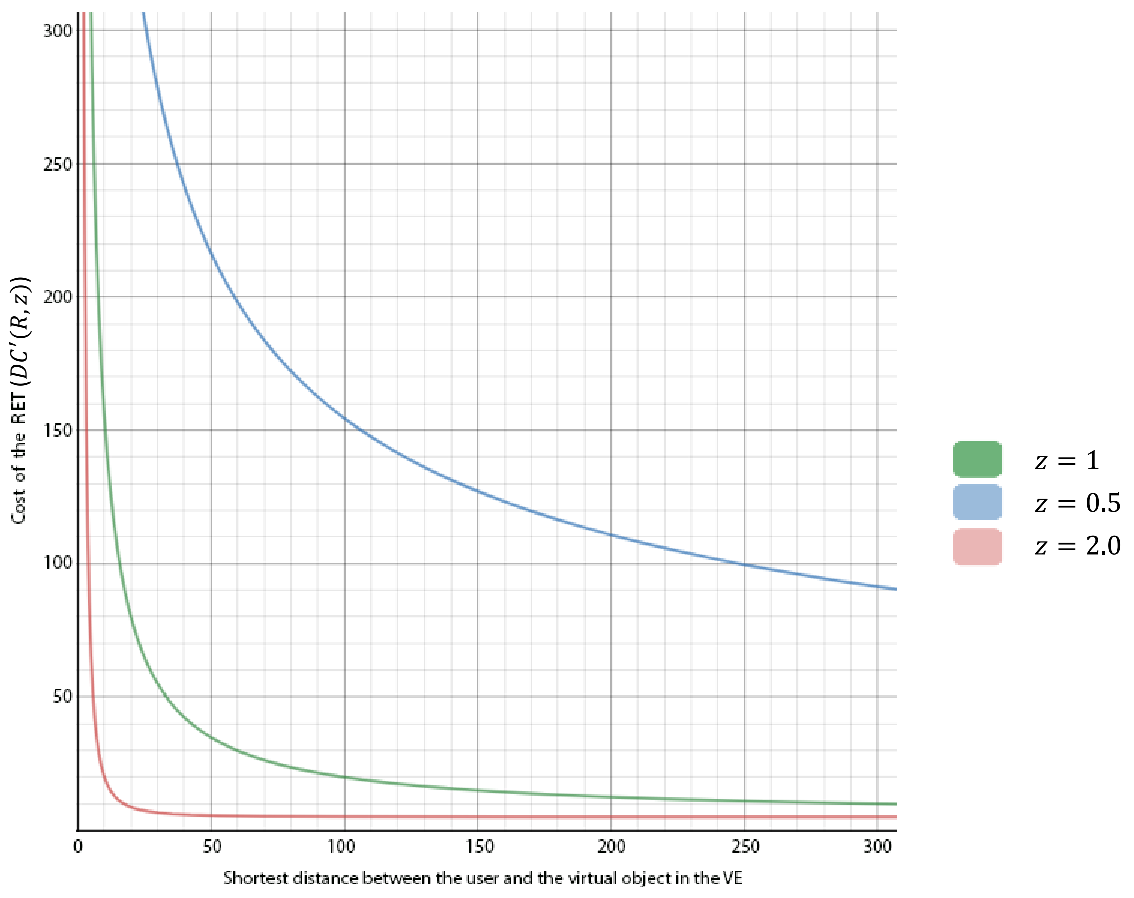
$$DC(R) = k * (1 - {}^{1}/_{(d+1)}) + P \times O_t \times {}^{1}/_{(d+1)}. \tag{3.10}$$

The curvature of cost function can be controlled by exponentiation of d. So, the final cost function is

$$DC'(R,z) = k * (1 - {}^{1}/_{(d^z+1)}) + P \times O_t \times {}^{1}/_{(d^z+1)}, \tag{3.11}$$

where z controls the curvature of the cost function. Some examples of how z controls the cost functions are shown in graph 3.1. Here the following assumptions are made: k is 5, P is 100, and O_t is 15.

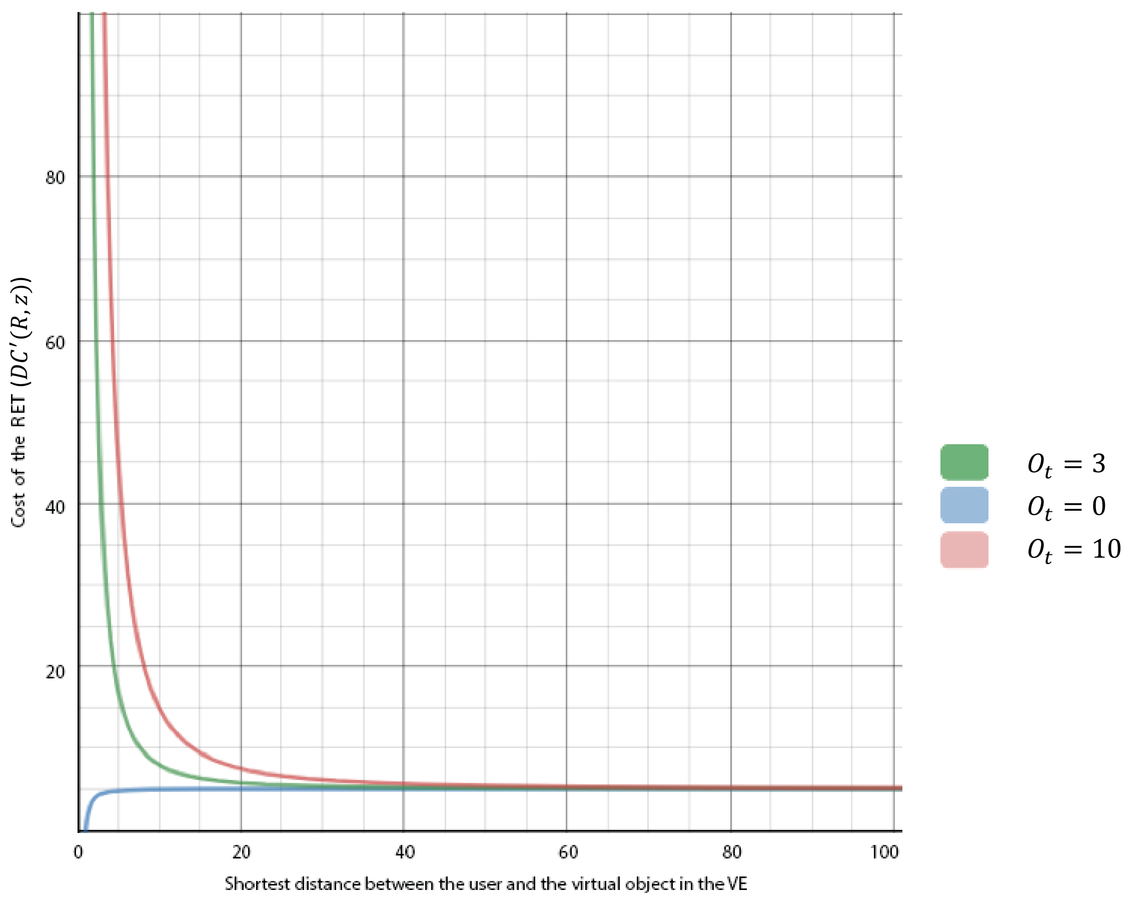
As the graphs show the rate of change of cost is influenced by z.



Graph 3.1: Cost of RET for varying z and constant k, P and O_t

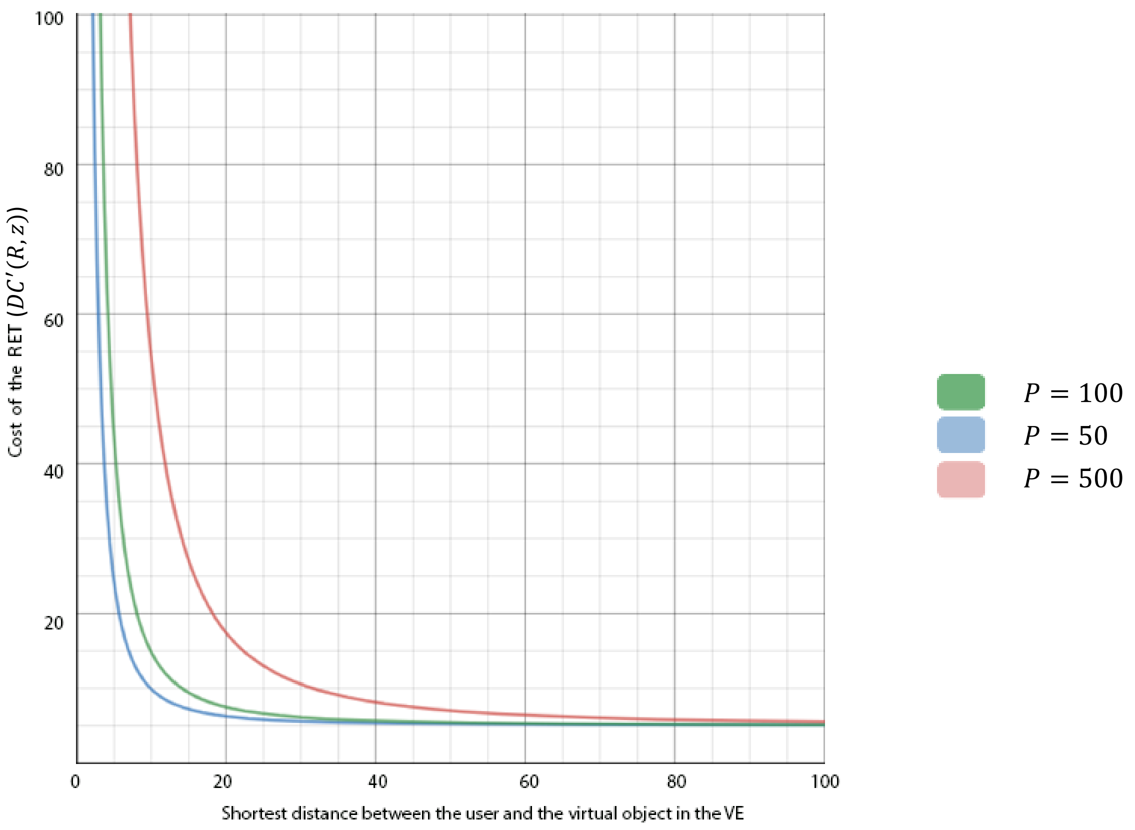
If an RET overlaps the virtual and real objects perfectly at the end of the path, i.e., if $O_t = 0$, when $d = 0$ then the RET has zero cost. But if the RET does not overlap the objects perfectly, the cost increases as d decreases. And as d increases, the costs from the original MPCRed algorithm are given more priority, hence the cost approaches the constant value k.

Graph 3.2 shows how cost varies with O_t. Here the following assumptions are made: z is 2, k is 5, and P is 100. As d decreases the RET must reduce O_t. If it does not, cost of the RET must increase. Graph 3.2 shows the dynamic cost function returns cost as expected.



Graph 3.2: Cost of RET for varying O_t and constant $z, k,$ *and* P

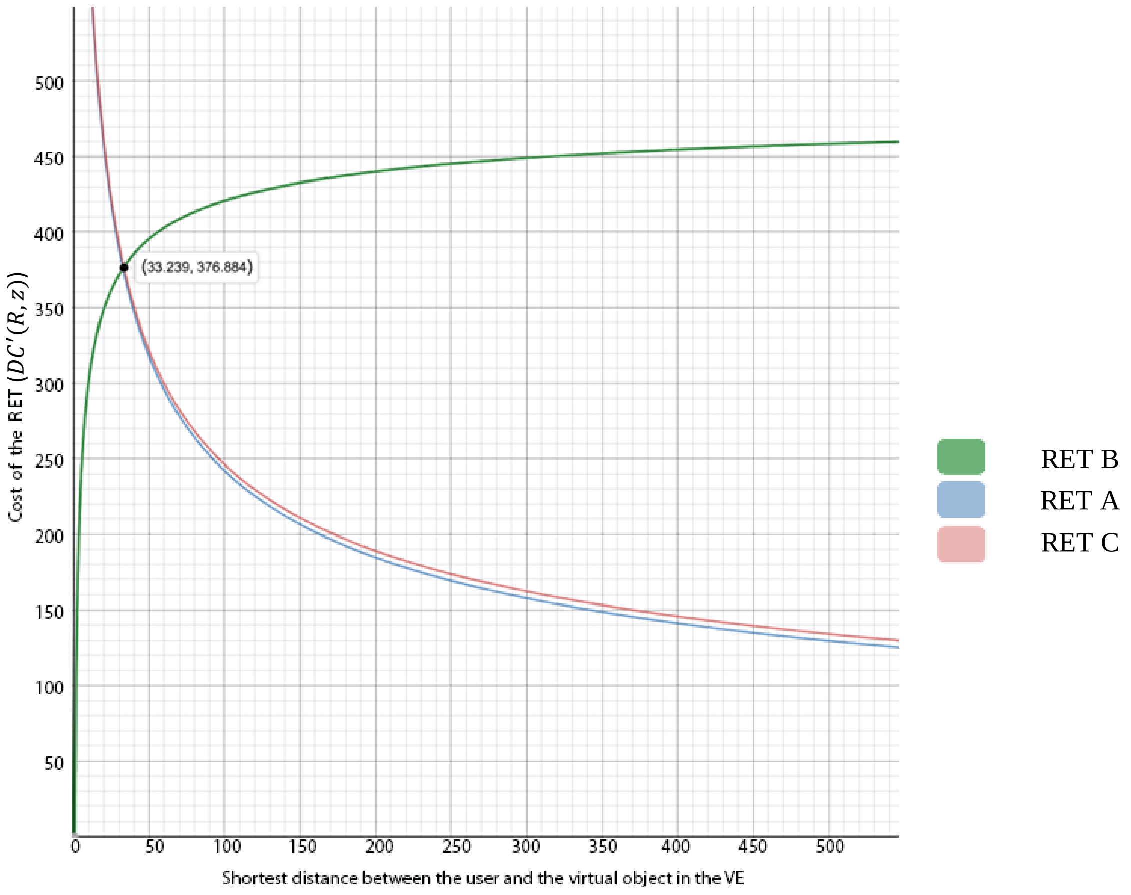
As mentioned before, P controls the distance at which the minimizing redirection is given more importance as compared to the minimizing offsets. Graph 3.3 shows how the dynamic cost function return costs with varying P and constant O_t, z, and k. Here assumptions are made as following: z is 2, k is 5, and O_t is 10.



Graph 3.3: Cost of RET for varying P and constant $z, k,$ *and* O_t

Graph 3.4 shows the costs returned by the cost function for three different RETs and breaks down the priorities in which these RETs are selected. In graph 3.4 following assumptions are made: z is 0.4, P is 100. The type of RET, k, and O_t for the three different RETs is as following:

- RET A – Curvature RET, $k = 5$, $O_t = 15$
- RET B – Reset RET, $k = 500$, $O_t = 0$
- RET C – Rotational RET, $k = 10$, $O_t = 15$



Graph 3.4: Cost of different RETs returned by the dynamic cost function

The graph shows how different RETs are selected during run time, depending upon distance to the object. When the distance between the user and the object is below 33 units, the reset RET has priority over curvature or rotational RETs. Since the total cost for the complete path is considered before selecting an RET for a segment of path, the gradual decrease of the costs is necessary instead of just finding the point where preference of RETs changes. Such an instance would be the point (33,376) in graph 3.4.

Clearly, depending upon the current distance to object d and O_t, the modified MPCRed algorithm may return different set of RETs for the same path. This is because minimizing offsets may sometimes lead to a greater amount of redirection than the minimizing-redirection technique in the original MPCRed algorithm.

For example, consider an VE that is much smaller than the available RE, as shown in figure 3.6. The RE can accommodate the entire VE, but the virtual object and its proxy are misaligned.

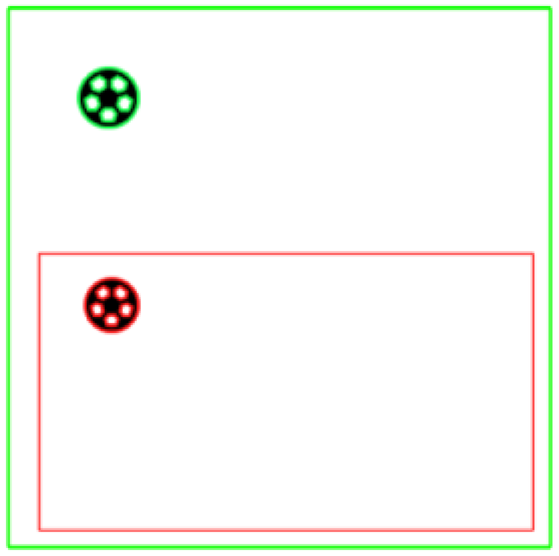


Figure 3.6: The RE can accommodate the entire VE, but the virtual object and its proxy are misaligned.

Since the RE can fit all of the VE, redirecting the user to avoid running into the walls is not necessary (zero redirection). However, if the virtual object and its corresponding real object are not aligned, redirection must be made to align the objects (i.e., a non-zero redirection) as shown in figure 3.7. Hence, minimizing offsets yields a greater redirection.

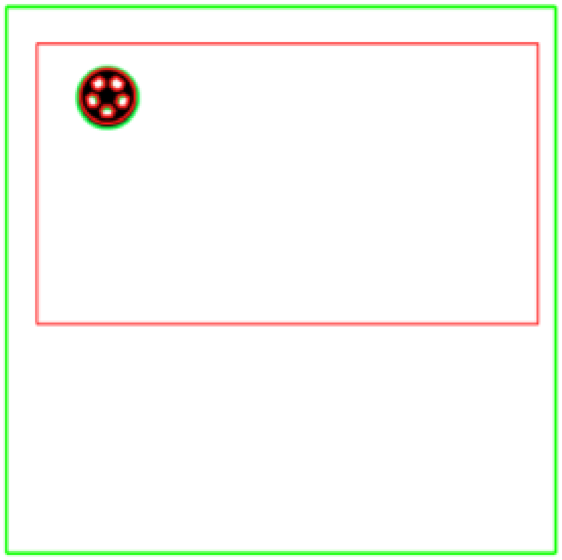


Figure 3.7: Redirection applied when the RE is large enough to accommodate the VE.

COMPARISON WITH PREVIOUS METHODS

The MPCRed algorithm by Nescher et al. [3]is a simplified case of the new Dynamic Cost MPCRed algorithm developed in this research. If there is no object in the VE which needs passive haptics, the distance to object d is infinity. The redirection cost function returns the default RET cost (that is, the one from the original MPCRed algorithm) in that case. I.e., in equation 3.11 as $d \rightarrow \infty$,

$$DC'(R,z) \rightarrow k * (1-0) + P \times O_t \times 0 = k = C_r(R) \tag{3.12}$$

Hence, the proposed method, while encompassing the existing algorithm, provides a way to allow passive haptics for an object in VE. This also can be used to steer the user toward a specific point in the RE, like the S2C algorithm [13], which steers the user towards the center of the RE.

CHAPTER IV

IMPLEMENTATION AND RESULTS

Both the VE and RE were developed using Unity3D [18] on an iMac with a 3.2Ghz Intel Core i5 processor, 16GB RAM, and NVIDIA GeForce GTX 675MX GPU. Two flat surface were defined to act as a real world and a virtual world. Two distinct characters also have been defined, which act as the human player and his/her virtual-world avatar. The movement of the real player is controlled by means of keyboard and mouse, whereas the movement of the avatar is derived from the real player. Even though the user controls the player character, the projected screen shows the avatar's view as shown in figure 4.1, hence simulating a player walking in a virtual environment while wearing a head-mounted display. Each player is assigned a pair of eyes to keep track of user's forward direction as shown in figure 4.2. Real Player is assigned a trail renderer [] to keep track of his path; this tail renderer highlights the path traversed by player. A rough approximated path of the virtual player is drawn in Adobe Photoshop [19] after each run; this is because the trail renderer does not modify the path drawn by it to match with the changing frame of reference of the VE, and the path of the Virtual Player in the VE is relatively simple compared to path of the real player.

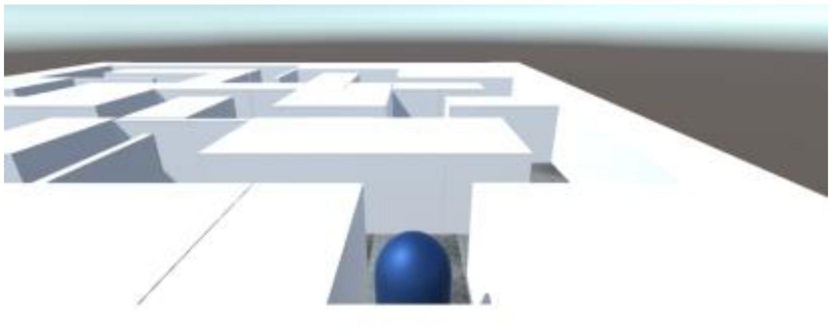


Figure 4.1: Players View

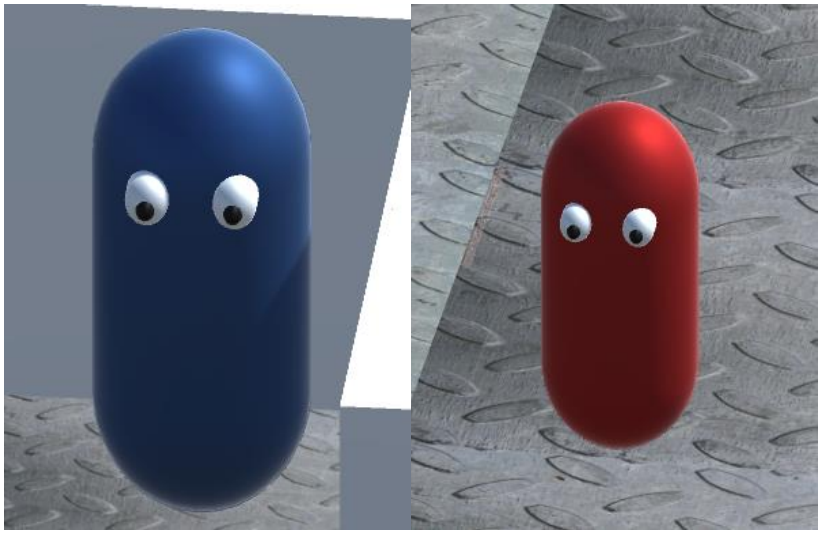


Figure 4.2: Avatar of Virtual Player(left) and Real Player(right)

The map of the VE, designed by Nescher et al. [3], is shown in figure 4.3. It is comprised of a set of hallways and junctions. Each junction is stored as a node in a database, and each node is denoted by a golden sphere in the map. The VE is defined so that the trajectories could be pre-defined [3] i.e. user can walk in the corridors defined but not in a free open world. The blue dot represents the avatar and his/her initial position.

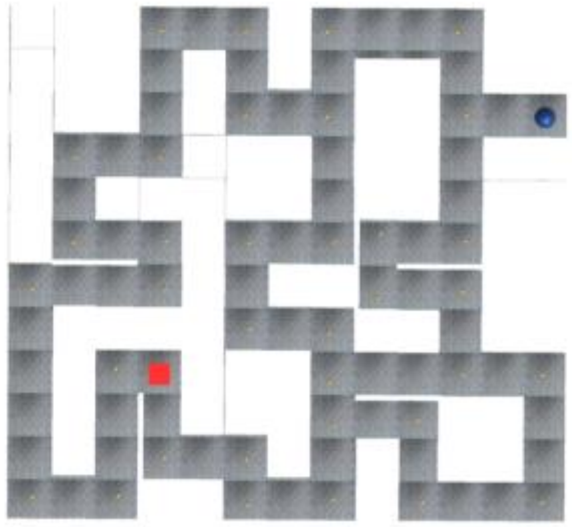


Figure 4.3: Map of the VE

Figure 4.4 shows the map of the RE. The red dot represents real player and the green cube represents the real object.

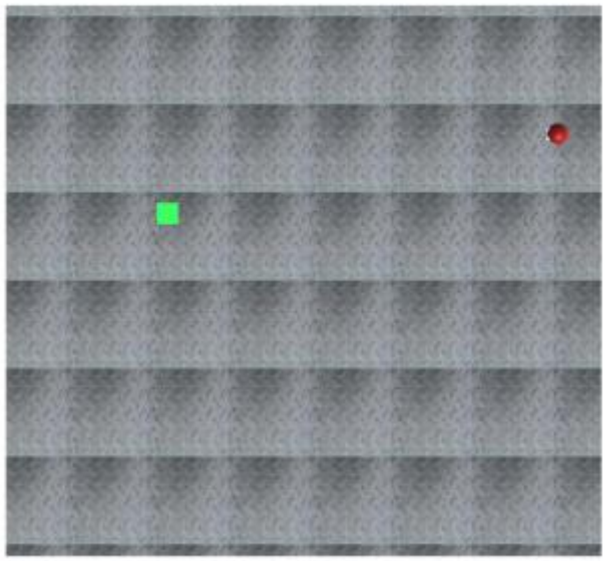


Figure 4.4: Map of the RE

INITIALIZATIONS

Curvature gain ρ_c and rotational gain ρ_q, defined in Chapter 2 are initialized as following: ρ_c is set to $\frac{1}{7.5}$ and ρ_q is set to $(1 - 0.67)$ for rotations in the direction of head movement and $(1 - 1.24)$ for rotations in the opposite direction [10]. The modified MPCRed algorithm is used to redirect the user. Value of P is set to 100 and z is set to 0.5. These values are determined by tuning for the desired behavior of the algorithm. i.e., the graph of redirection can be varied based upon the selected values of z, and P.

EXECUTION

The modified MPCRed algorithm calculates the new dynamic costs and returns the desirable RET whenever the user enters or leaves a node. (This could be done on a separate thread to improve performance, but Unity has limitations that do not allow multithreading.) When this algorithm is applied, the path of the user in the RE and the path he/she perceives in the virtual world are very different just like the regular RDW. This is shown if Figures 4.5 and 4.6.

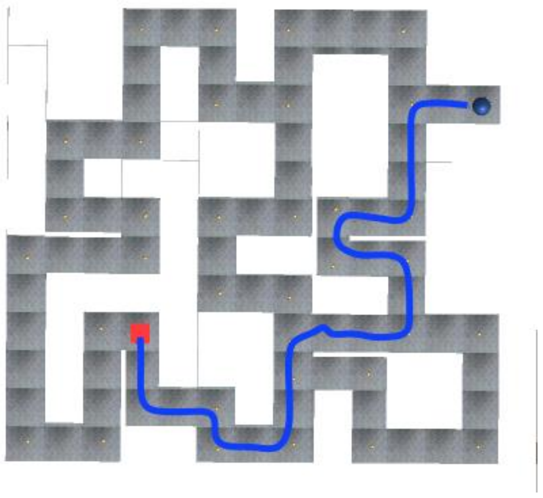


Figure 4.5: Users path in VE

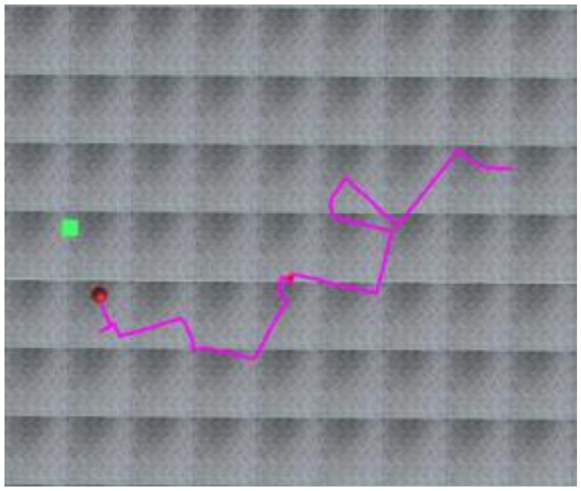


Figure 4.6: Corresponding path in RE

The red path indicates the player's path in real world. The blue path is the path taken by avatar in the VE. The modified MPCRed algorithm applies RETs along the path. These RETs attempt to minimize the offsets when user gets close to the virtual object and are minimal when user is distant from the object, as described in Chapter 3.

To test the operation of the modified MPCRed algorithm, a series of test cases are designed. In each test case, the player starts walking towards and away from the virtual object in the same VE defined by Nescher et al. [3], but the RE's size is varied. The modified MPCRed algorithm is used to select the RETs applied along the player's path. The distance between the player and the virtual object d, and the distance between the virtual object and its proxy O_t help the algorithm to determine which RET to select, as described in Chapter 3. Both distances are collected over time and plotted on a graph to show how the modified algorithm behaves during the player's traversal.

The y-axis in the graph is the distance in Unity units and the x-axis is the time in seconds. The blue line represents the shortest distance between the user and the virtual object. The orange line represents the offset (i.e., the distance between the virtual object and its proxy). Each test case has been modified for different size of RE, different placement of objects. The results obtained are described below.

RESULTS

To test the operation of the modified MPCRed algorithm, a series of test cases are designed.

In the most ideal case, the RE is large enough to accommodate all of the VE, and the virtual and real objects are perfectly overlapping. In this case, there should be zero redirection. This is shown in figures 4.7(a) and 4.7(b). In the figures, the path of the avatar in the VE is the same as the player's path in the RE. This makes sense, because the RE is larger than the VE, there is a one-to-one map from the RE to the VE, and objects also align in this map. Hence, there is no need for a redirection and the offset should stay zero throughout the run. The data in graph 4.1 shows that there is zero redirection over all.

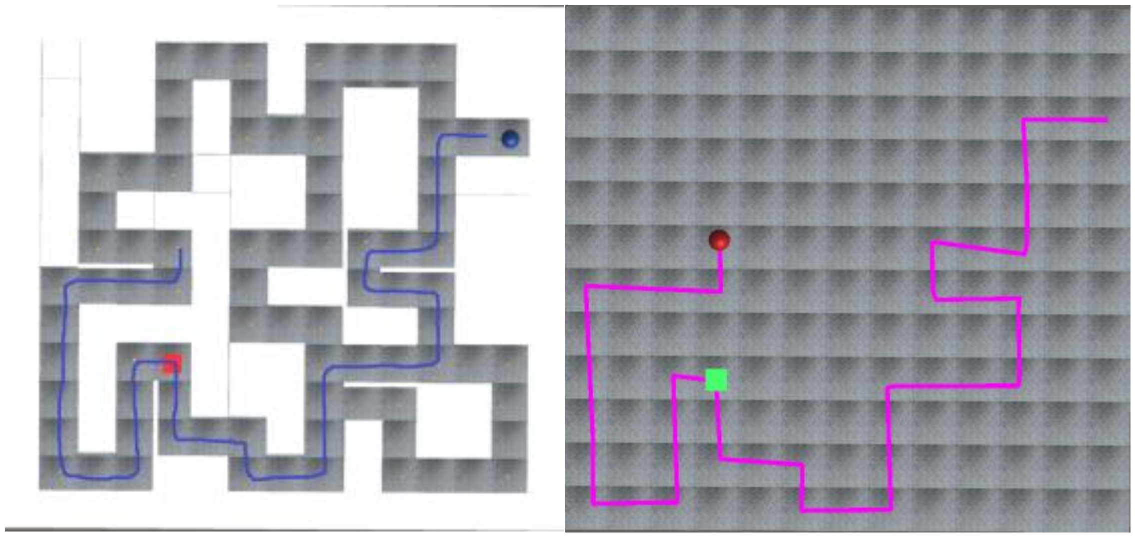
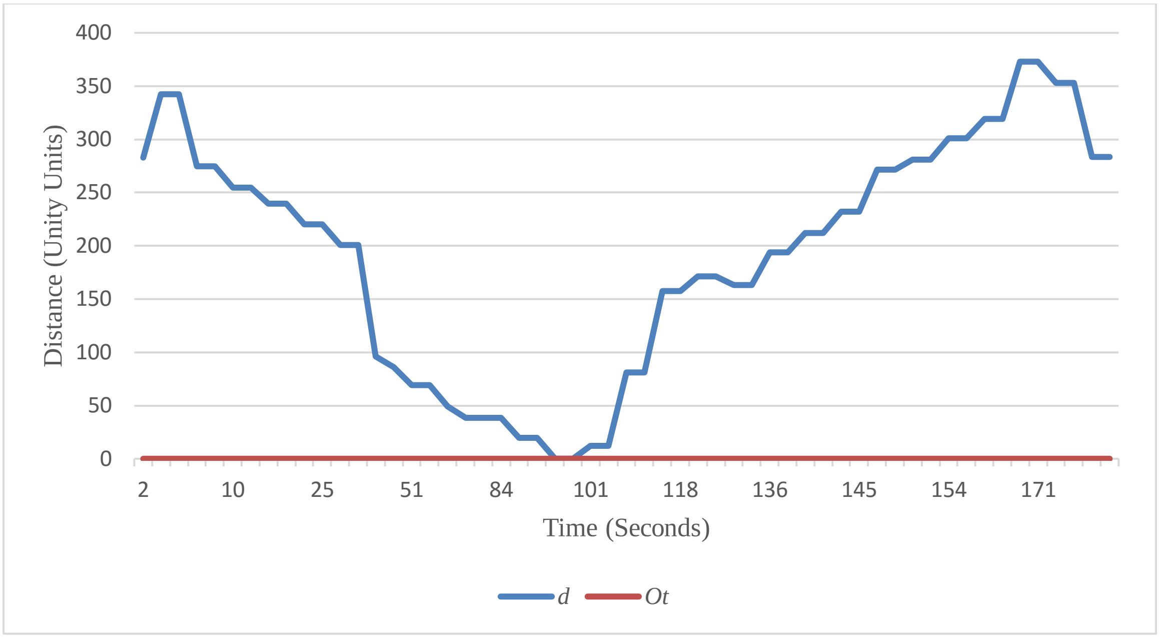


Figure 4.7: (a) Path in the VE (Left) (b) Corresponding path in the RE(Right) where, scale of VE to RE is 1:1 in length and 1:1 in width, and objects are overlapping



Graph 4.1: Shows the distance between user and object, and O_t for the test case shown in figure 4.7

In the case of a larger RE and no overlap between the objects, as the player gets closer to the object in the VE, redirection is made to overlap the objects.

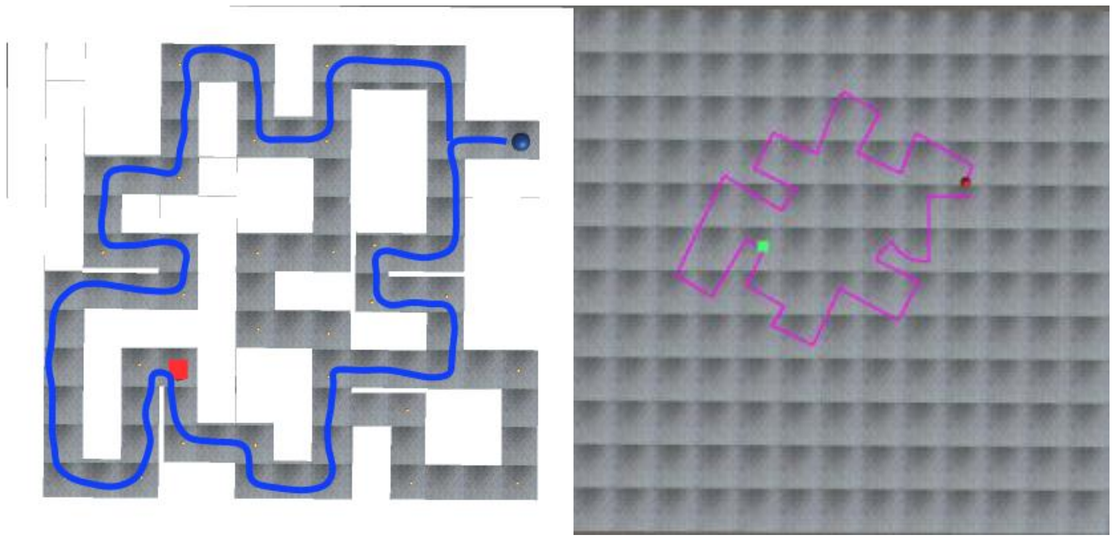
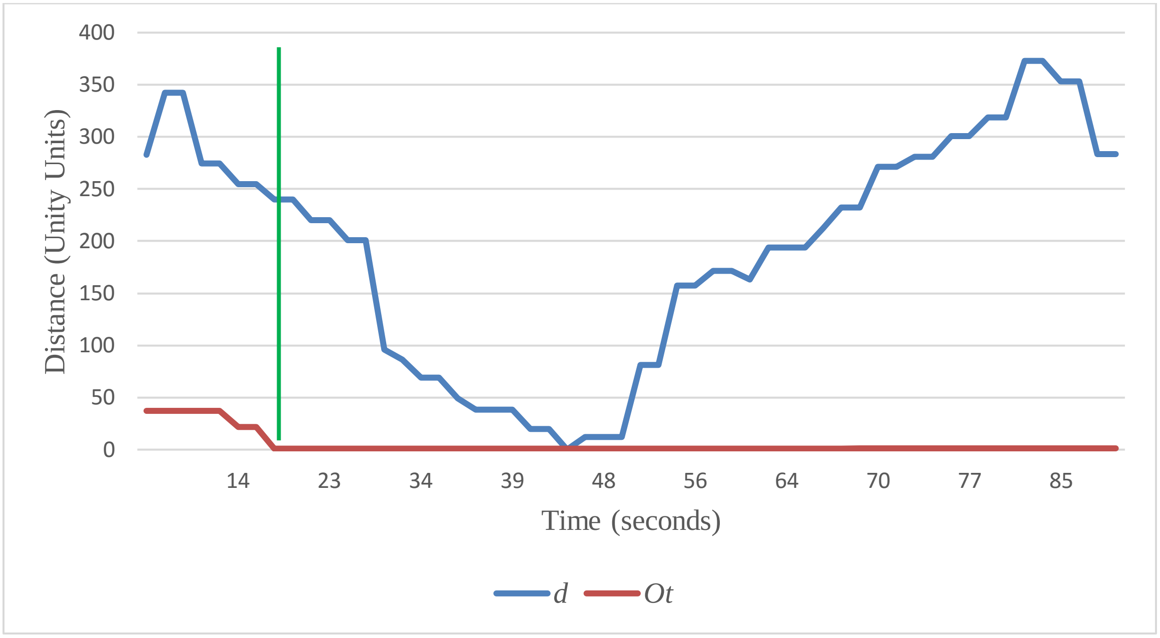


Figure 4.8: (a) Path in the VE (Left) (b) Corresponding path in the RE(Right) where, scale of VE to RE is 13:20 in length and 12:18 in width, and objects are overlapping

After that, since the objects are overlapped and the RE can accommodate all the VE it becomes similar to the previous test case. Hence, there is zero redirection after that. Figure 4.8 shows the player's path in the RE and the VE. It's corresponding graph (4.2) shows that after 18 seconds (denoted by vertical green line) the objects overlap and zero redirection is applied there forth, hence O_t does not change.



Graph 4.2: Shows the distance between user and object, and O_t for the test case shown in figure 4.8

Figures 4.9, 4.10, 4.11 and 4.12 show the path of the player in the RE and in the VE for different sizes of RE.

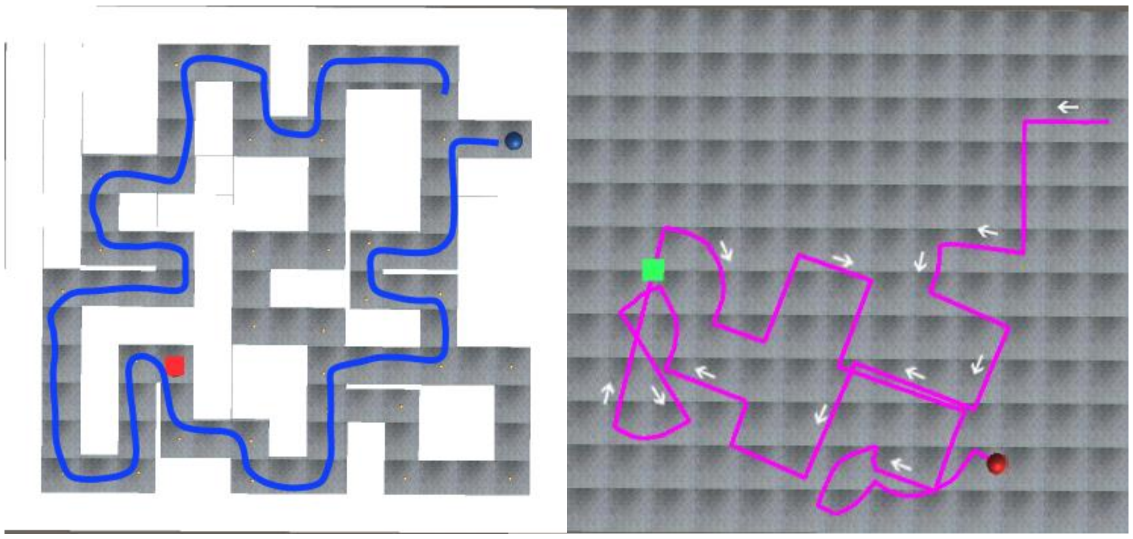
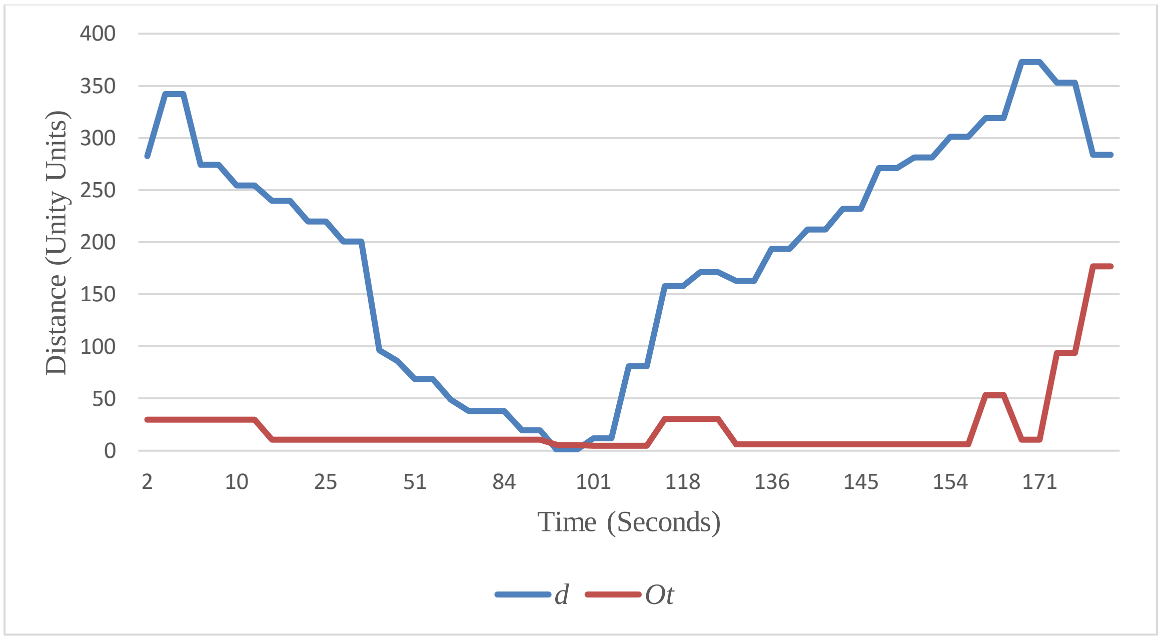


Figure 4.9(a) Path in the VE (Left) (b) Corresponding path in the RE(Right) where, scale of VE to RE is 1:1 in length and 1:1 in width, and objects are not overlapping



Graph 4.3: Shows the distance between user and object, and O_t for the test case shown in figure 4.9

In the above test case the VE and RE are of equal size. They overlap perfectly forming a one-to-one mapping. But, similar to the previous test case the objects are not aligned. Hence, to overlap the objects and allow the user to walk in the whole VE, the mapping changes continuously. As a result, the distance between the objects changes as well. The graph shows that as the distance between user and the virtual object increases, O_t can increase or decrease. I.e., the RETs are not being applied to minimize O_t but instead they are being applied to minimize redirection. And as the distance decreases, O_t is strictly minimized to allow passive haptics. The bump in O_t at 123rd second is because the player may have gone out on the RE if the particular RET was not applied even though it caused a raise in O_t when user was near the object.

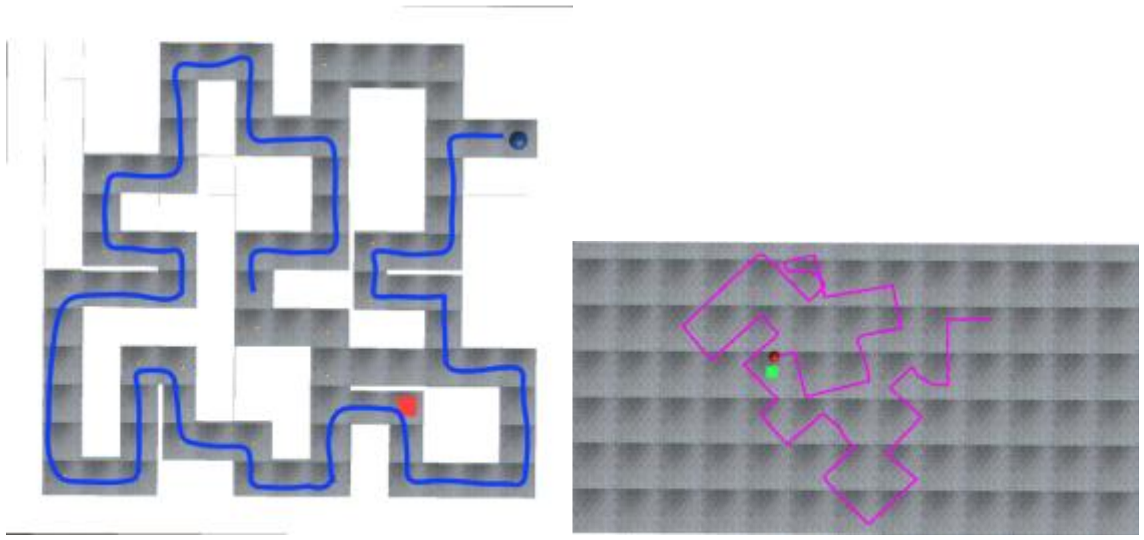
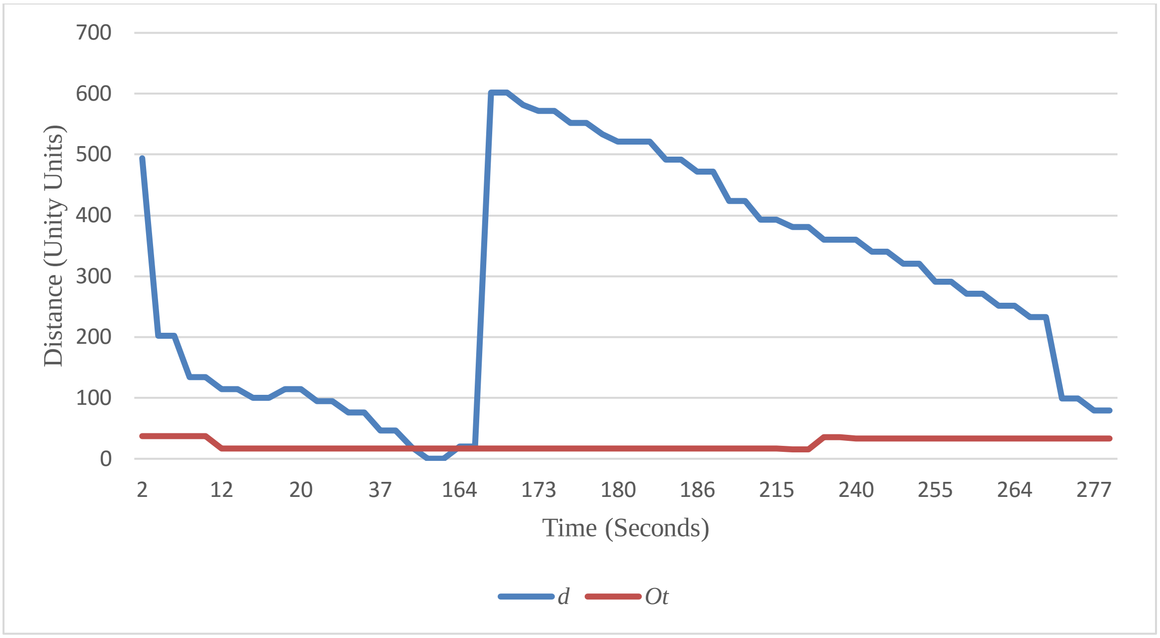


Figure 4.10(a) Path in the VE (Left) (b) Corresponding path in the RE(Right) where, scale of VE to RE is 1:1 in length and 40:21 in width, and objects are overlapping



Graph 4.4: Shows the distance between user and object, and O_t for the test case shown in figure 4.10

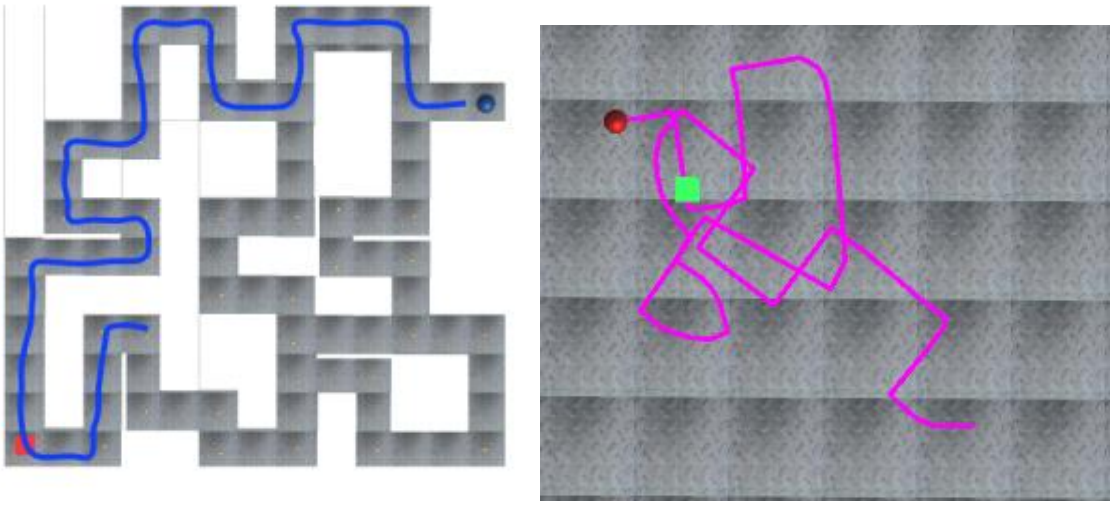
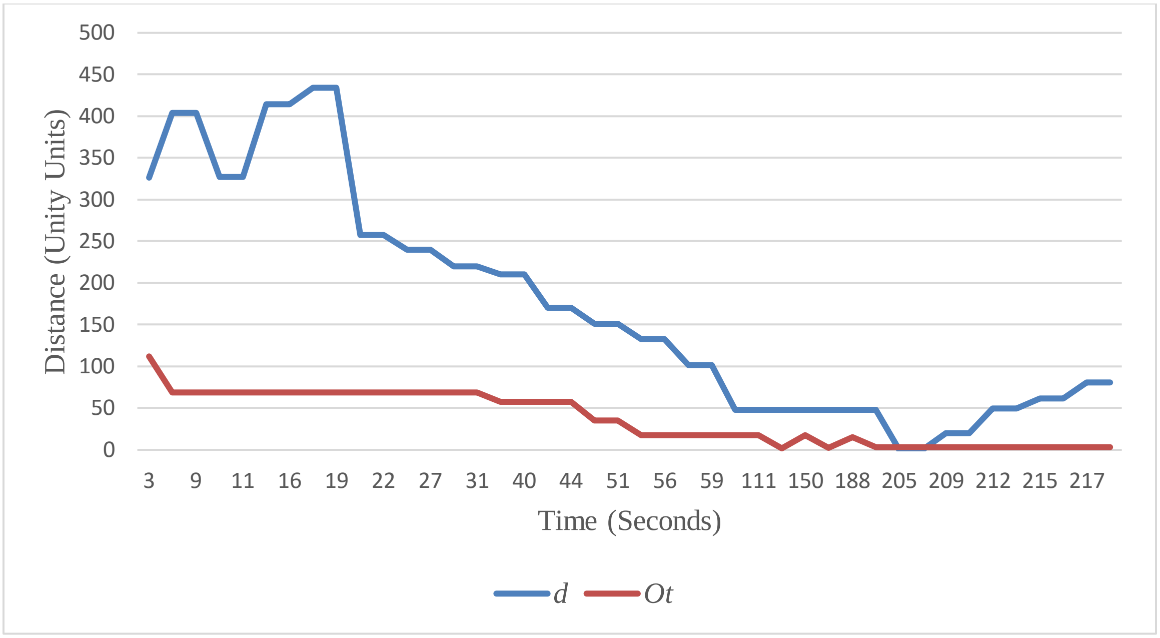


Figure 4.11: (a) Path in the VE (Left) (b) Corresponding path in the VE(Right) where, scale of VE to RE is 13:6 horizontally and 12:5 vertically



Graph 4.5: Shows the distance between user and object, and O_t for the test case shown in figure 4.11

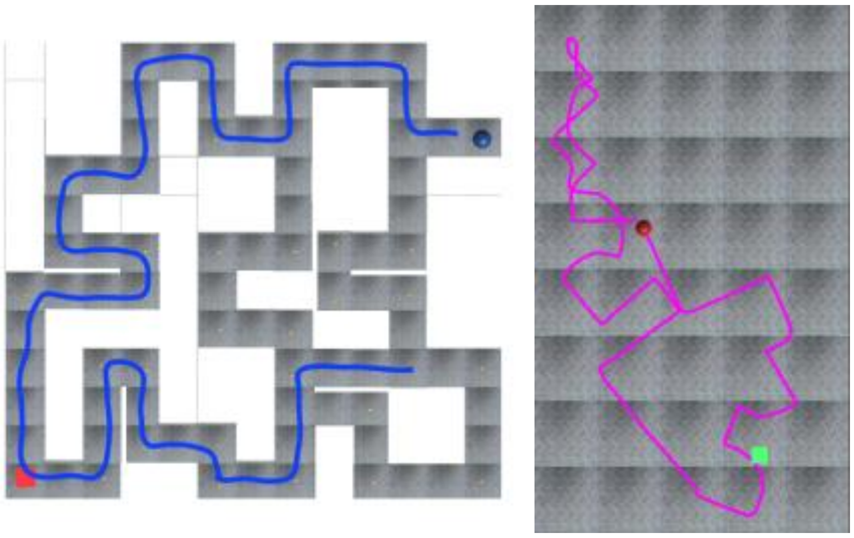
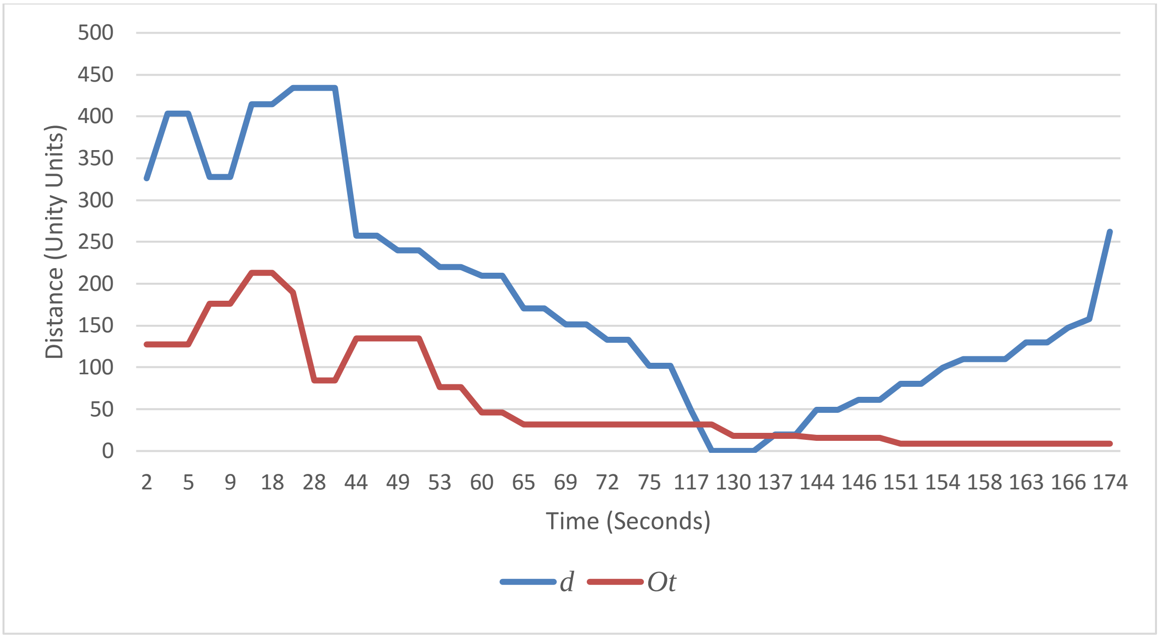


Figure 4.12: (a) Path in the VE (Left) (b) Corresponding path in the VE(Right) where, scale of VE to RE is 13:5 horizontally and 3:2 vertically



Graph 4.6: Shows the distance between user and object, and O_t for the test case shown in figure 4.12

In each of these test cases depicted in figures 4.9 – 4.12, the RE is smaller (lesser area) than the VE, and different in shape as well. The grid patters show how the scale differentiates. The objects in the virtual world and their proxies are placed at different locations.

OBSERVATIONS

As the graphs show, when player is distant from the object, the offsets are increasing or decreasing independent to player's movements, as the modified MPCRed algorithm assigns low importance to minimizing offset. But, as the player gets close to the object, the offsets also decrease. This shows the modified MPCRed algorithm works as expected. But it was discovered that, as seen in graphs 4.3 and 4.4 the overlap is not always zero.

This is because the redirections can only minimize the deviation of overlap between the object in the VE and its proxy within the constraints of the RETs. The minimization may not be zero, because all the redirection yields only a particular set of points where the overlap will be zero. This is the main drawback of this method.

An example of this is shown in the figure 4.13. In this example, only curvature RET and zero RET are assumed to be applied. For the path in VE shown in figure 4.13(a), using curvature and zero RETS, only a few paths can be created in the RE as shown in 4.13(b). The blue footballs show all the possible locations where if the proxy object is placed then O_t will be zero.

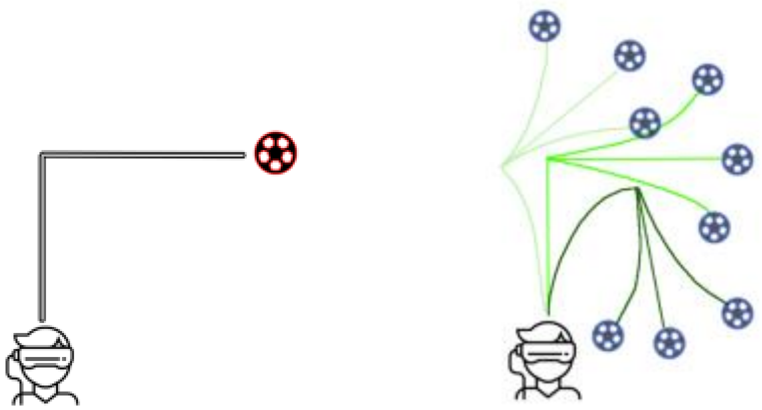


Figure 4.13: (a) Virtual Path (left) (b) Corresponding Possible Real Paths (right) showing O_t cannot always be zero

Although using rotational and reset RETs may maximize the area where the proxy object can be placed for a zero offset, it is clear that there will be limitations even then.

One other drawback of using this method is since, the goal is minimization of deviation of overlap, there can be cases where the deviation is more initially but may lead to lower offset overall. Such cases are discarded in this model. But, this is an advantage if it is used to redirect player to certain areas of interest. For example, the S2C Algorithm constantly redirects user towards the center of the available RE, and the MPCRed algorithm only applies redirection when one is needed, unlike S2C; the proposed method gradually can slide from one behavior to another dynamically, as needed. But instead of center of the room, areas of interest in the RE can be defined.

It also is clear that the more the RETs, the better the redirection can get and the lower the unreachable areas described above. In this research zero, curvature, reset and rotational RET's are used.

CHAPTER V

CONCLUSION

In this research, the state of the art redirection algorithm, Model Predictive Control Redirection algorithm (MPCRed) has been modified to allow passive haptics along with redirected walking (RDW).

RDW is used to traverse a large virtual environment (VE) in a much smaller real environment (RE). A redirection algorithm defines how RDW must work. However not all of the algorithms provide a way to allow passive haptics and that includes the MPCRed algorithm.

Passive Haptics is the process of providing touch feedback to a user experiencing virtual reality. Placing proxy objects for the objects in the VE helps provide this feedback. Using RDW to traverse a VE creates nondeterminism in the mapping of VE to the RE; as a result, placing proxy objects in the RE to provide passive haptics for the objects in VE is a challenging task. In this research, this problem is addressed and solved by modifying the MPCRed Algorithm.

The goal of MPCRed algorithm is to minimize redirection for any path the player takes using redirection techniques (RETs). For this, each redirection technique (RET) is assigned a cost. An overall cost of redirection for any path is calculated by adding up the costs of RETs applied along that path; by minimizing this cost, redirection is minimized in MPCRed algorithm. These costs have been modified in this research to allow passive haptics along with minimizing redirection.

Since the mapping from the VE to the RE changes during runtime, the modified costs are dynamic. These costs are based on how much each RET minimizes the translational offsets; same can be adapted for rotational offsets, which is beyond the scope of this work. When the dynamic costs are used, results show that the player is redirected closer to the proxy object when he/she reaches the virtual object. Results also show that the player is redirected less when the object in the VE is not close, and redirected to allow passive haptics as he/she gets closer.

Thus, the proposed method while encompassing the existing MPCRed algorithm provides a way to allow passive haptics dynamically.

Although the existing RETs provide a good map, it is not perfect as there is a limit on how much area the RETs can reach. Future work can include overcoming these limitations by creating new RETs.

REFERENCES

[1] Razzaque, S., Kohn, Z., and Whitton, M.C. Redirected walking. in Proceedings of EUROGRAPHICS. 2001. Citeseer.

[2] Souman, J.L., Frissen, I., et al., Walking straight into circles. Current Biology, 2009. 19(18): p. 1538-1542.

[3] Nescher, T., Huang, Y.-Y., and Kunz, A. Planning redirection techniques for optimal free walking experience using model predictive control. in 3D User Interfaces (3DUI), 2014 IEEE Symposium on. 2014. IEEE.

[4] Insko, B.E., Passive haptics significantly enhances virtual environments. 2001, University of North Carolina at Chapel Hill.

[5] Ruddle, R.A. and Lessels, S., The benefits of using a walking interface to navigate virtual environments. ACM Transactions on Computer-Human Interaction (TOCHI), 2009. 16(1): p. 5.

[6] Williams, B., Narasimham, G., et al. Updating orientation in large virtual environments using scaled translational gain. in Proceedings of the 3rd symposium on Applied perception in graphics and visualization. 2006. ACM.

[7] Interrante, V., Ries, B., and Anderson, L. Seven league boots: A new metaphor for augmented locomotion through moderately large scale immersive virtual environments. in 2007 IEEE Symposium on 3D User Interfaces. 2007. IEEE.

[8] Williams, B., Narasimham, G., et al. Exploring large virtual environments with an HMD when physical space is limited. in Proceedings of the 4th symposium on Applied perception in graphics and visualization. 2007. ACM.

[9] Peck, T.C., Fuchs, H., and Whitton, M.C., Evaluation of reorientation techniques and distractors for walking in large virtual environments. IEEE Transactions on Visualization and Computer Graphics, 2009. 15(3): p. 383-394.

[10] Steinicke, F., Bruder, G., et al., Estimation of detection thresholds for redirected walking techniques. IEEE Transactions on Visualization and Computer Graphics, 2010. 16(1): p. 17-27.

[11] Steinicke, F., Bruder, G., et al. Analyses of human sensitivity to redirected walking. in Proceedings of the 2008 ACM symposium on Virtual reality software and technology. 2008. ACM.

[12] Peck, T.C., Fuchs, H., and Whitton, M.C., The design and evaluation of a large-scale real-walking locomotion interface. IEEE transactions on visualization and computer graphics, 2012. 18(7): p. 1053-1067.

[13] Hodgson, E. and Bachmann, E., Comparing four approaches to generalized redirected walking: Simulation and live user data. IEEE transactions on visualization and computer graphics, 2013. 19(4): p. 634-643.

[14] Zmuda, M.A., Wonser, J.L., et al., Optimizing constrained-environment redirected walking instructions using search techniques. IEEE transactions on visualization and computer graphics, 2013. 19(11): p. 1872-1884.

[15] Kohli, L., Burns, E., et al. Combining passive haptics with redirected walking. in Proceedings of the 2005 international conference on Augmented tele-existence. 2005. ACM.

[16] Azmandian, M., Hancock, M., et al. Haptic Retargeting: Dynamic Repurposing of Passive Haptics for Enhanced Virtual Reality Experiences. in Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems. 2016. ACM.

[17] Sun, Q., Wei, L.-Y., and Kaufman, A., Mapping virtual and physical reality. ACM Transactions on Graphics (TOG), 2016. 35(4): p. 64.

[18] Unity3D [Computer software]. (Version: 5.6.0f3). Retrieved from https://store.unity.com/

[19] Adobe Photoshop [Computer Software] (Version: 2017.1.1 Release) Retrieved from http://www.photoshop.com/products

VITA

Rohit Nutalapati

Candidate for the Degree of

Master of Science

Thesis: PLANNING REDIRECTION FOR DYNAMIC PASSIVE HAPTICS USING MODEL PREDICTIVE CONTROL

Major Field: Computer Science

Biographical:

Education:

Completed the requirements for the Master of Science in Computer Science at Oklahoma State University, Stillwater, Oklahoma in December, 2017.

Completed the requirements for the Bachelor of Technology in your Computer Science at GITAM University, Visakhapatnam, India in 2015.