Planning Redirection Techniques for Optimal Free Walking Experience Using Model Predictive Control

Planning Redirection Techniques for Optimal Free Walking Experience Using Model Predictive Control

Thomas Nescher[*] ICVR - IWF - ETH Zurich
Ying-Yin Huang[†] EIM - TIM - ETH Zurich
Andreas Kunz[‡] ICVR - IWF - ETH Zurich

ABSTRACT

Redirected Walking (RDW) is a technique that allows exploring immersive virtual environments by real walking in a small physical room. RDW employs so-called redirection techniques (RETs) to manipulate the user's real world trajectory in such a way that he remains within the boundaries of the physical room. Different RETs were suggested and evaluated in the past. In addition, steering algorithms were proposed that apply a limited set of RETs to redirect a user away from the physical room's boundaries.

Within this paper, a generalized approach to planning and applying RETs is presented. It is capable of dynamically selecting suitable RETs and also controlling parameters like their strengths. The problem of steering a user in a small physical room using RETs is formulated as an optimal control problem. This allows applying an efficient probabilistic planning algorithm to maximize the free walking experience. The proposed algorithm uses a map of the virtual environment to continuously determine the optimal RET that has to be applied next.

The suggested algorithm is evaluated within a user study and compared to a state-of-the-art steering algorithm. Results show that for the given virtual environment, it is able to reduce the number of collisions with the room boundaries by 41% and furthermore reduces the amount of applied redirections significantly.

Keywords: Virtual reality, locomotion, redirection techniques, optimal control, model predictive control, redirected walking.

Index Terms: H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems—Artificial, augmented, and virtual realities; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Virtual reality

1 INTRODUCTION

Immersive Virtual Environments (IVEs) intend to address human perception entities like video, audio, and haptics. While early Virtual Environments (VEs) allowed for a navigation using mouse, keyboard, or joystick, later research changed the haptic interaction metaphor from a "fly-through" to a "walk-through". Starting with stepping-in-place installations that only partly generated the sensation of real walking, virtual treadmills and magic carpets were used later, which mechanically pulled the user back to the center of the walking platform. While all devices allowed experiencing a VE that is much larger than the physical installation, the haptic sensation of walking in VEs still was not perfectly addressed.

Thus, recent research uses Redirected Walking (RDW) [15] to "compress" a large VE into a physical room that could be significantly smaller, while allowing for the sensation of real walking. Since the user typically wears a head-mounted display (HMD), he cannot see the real environment anymore and thus sophisticated redirection techniques (RETs) [20] have been developed to redirect the user away from the physical room's boundaries.

Many of the existing algorithms for RDW were tailored to a limited subset of available RETs, and focused on keeping away the user as much as possible from the physical boundaries. However, this might introduce user disturbances and a noticeable reduction of immersion, which could have been avoided if a probabilistic planning strategy would have been applied that takes into account the VE's geometry and the possible walking trajectories of the user.

Following the idea of probabilistic planning of redirection [25], the paper introduces a planning framework that is based on a rigorous definition of redirection as an optimization problem. This allows applying optimization techniques from control theory. The proposed system allows minimizing the user's disturbance and is capable of applying different RETs alternately or simultaneously. The paper thus introduces a new paradigm which moves away from planning or applying redirection on geometrical aspects only, but focuses also on the current action, e.g. the user disturbance caused by an applied RET. The model predictive control (MPC) approach therefore builds the basis for applying perceivable redirections. While the proposed approach requires a map of the trajectories in the VE, this does not reduce the general applicability of the proposed methods on goal oriented applications, as human walking trajectories were shown to be highly stereotyped [7].

After presenting related work in this field, the paper presents the optimal control approach for planning and applying RETs, followed by an evaluation and a discussion of the achieved results.

2 RELATED WORK

Initially, Razzaque et al. [15] proposed RDW. Here, a user's head is tracked within the boundaries of a tracked space and the VE is presented on a HMD while the real head motions are transferred to motions in the VE. Hence, a user can walk freely within a VE as long as it fits the real tracked space. However, redirected walking adds imperceivable rotations to the VE to guide a user on curved paths or scales rotations. This way, the user can be kept within the boundaries of the tracked space even if the VE is larger.

This original approach was further researched and extended with other techniques. Williams et al. [23] proposed to scale translational movements. Interrante et al. [10] further improved this method by robustly determining the user's direction of movement.

Further, Williams at al. [24] suggested so-called reset techniques, which reorient a user in the tracked space when other techniques fail. A reset technique stops a user's walking in a VE and reorients or repositions him. Peck et al. [14] compared different reset techniques and proposed so-called distractors. They are improved reset techniques that distract a user while the VE rotates.

Steinicke et al. [16] generalized different RETs and defined so-called gains (rotation, curvature, and translation) that describe their strengths. Furthermore, they evaluated detection thresholds for different RETs [17, 18]. Neth et al. studied the velocity dependency of so-called curvature gains, i.e. how much imperceptible rotation can be added for redirection for a user depending on his walking velocity [11]. Recent research for redirected walking focuses on how and when to apply a RET. Therefore, Suma et al. [20] presented a taxonomy for RETs.

[*]e-mail: nescher@iwf.mavt.ethz.ch
[†]e-mail:yingyinhuang@ethz.ch
[‡]e-mail:kunz@iwf.mavt.ethz.ch

So-called steering or redirection algorithms were developed to decide which RET is to be used to keep a user within the real environment's boundaries. The steer-to-center (S2C) algorithm continuously rotates the user with redirections towards the real environment's center [13]. S2C was also used by Hodgson et al. [9] who evaluated the potential of redirected walking for spatial inference. Further, they compared different generalized approaches for redirected walking and proposed improvements to the S2C algorithm [8]. Engel et al. [3] proposed a psychophysically calibrated controller that employs a cost function based on the noticeability of redirection. Nitzsche et al. [12] and Su et al. [19] suggested an approach that dynamically minimizes the introduced curvature, but does not make imperceptibility a primary goal. A new way for planning RETs was proposed by Zmuda et al. [25]. Their algorithm (FORCE) employs probabilistic planning and uses a terminal state evaluation function to determine feasible RETs that have to be used. Suma et al. [21, 22] proposed completely different types of RETs that make use of human change blindness or special self-overlapping VEs.

3 Optimal Control for Planning Redirection Techniques

Optimal control deals with problems where several decisions have to be made in stages that influence a system. The goal is to minimize a cost function that captures the system's undesirable outcomes [1]. The applied decisions are referred to as actions.

Redirecting a user inside a physical room can be regarded as such a problem. Obviously, we want to avoid that the user collides with a physical obstacle. Hence, different RETs can be applied to change a user's geometrical configuration in the real room, e.g. his orientation, in relation to the VE. However, some RETs might be less desirable than others. E.g. a subtle RET is typically preferred over an overt reorientation reset. Every RET, i.e. action, changes the user's configuration, but it will also affect all future decisions that have to be made. Thus, some actions might be desirable at present but could cause undesirable outcomes in the future. Finally, the outcome of an action is not fully predictable as we do not know for sure where the user will go.

In principle, an optimal control algorithm describes the user's geometrical configuration and his walking state like his velocity with a state variable and models the user's potential behavior as a function that updates the state variable. E.g. such a function could describe a trajectory in a VE. Hence, this function predicts the user's future states. Furthermore, it takes a RET as input and predicts how the future state will look like if a RET is applied while the user walks along the trajectory. Given this model, a simplified optimal control algorithm recursively applies different RETs, e.g. in a depth-first-search manner, to find the best one. Hereby, a cost function allows evaluating the state together with the applied action. Therefore, the algorithm looks into the future considering different RETs that could be applied in series and chooses the best RET with regard to its future consequences.

Below we present the basic stochastic system model suited to the redirection problem. The formulation of the basic model is adapted from [1]. The model is limited to discrete time systems. A subscripted capital R means that a parameter describes a state in the real room, like in θ_R and a subscripted V denotes that a parameter refers to the VE respectively. The planar coordinate system that is used for transformations or rotations is as follows: x-axis forward, y-axis left, and rotations from the positive x-axis (0 radians) to the positive y-axis are positive. Angles are in radians.

3.1 Basic Model

The discrete time dynamic system is described as follows

$$\mathbf{x}_{k+1} = f_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) \qquad k = 0, 1, \ldots, N-1 \tag{1}$$

- k denotes the discrete time or stages running from 0 to $N-1$. The duration of a stage is typically constant. N is the time horizon considered for the system evolution or planning.
- $\mathbf{x}_k \in \mathbb{R}^7 = [x_R, y_R, \Theta_R, x_V, y_V, \Theta_V, v_R]^T$ is the user's current state vector. The state captures the user's configuration, i.e. his position (x, y), orientation Θ in the real and virtual environment, and his tangential velocity v_R. The state space could be further extended to contain other characteristics like the user's angular velocity ω_R or a direction prediction.
- $\mathbf{u}_k \in U(\mathbf{x}_k) \subset C_k$ is the current applied action. The set of available actions $U(\mathbf{x}_k)$ is a subset of the complete set of actions C_k and can depend on the current state. An action is a RET or a combination of RETs, see Section 3.2
- $\mathbf{w}_k \sim P(\cdot|\mathbf{x}_k, \mathbf{u}_k)$ denotes the noise that captures the uncertainty of the system.
- f_k is the state update function that captures the evolution of the system, see Section 3.3.

In the given system, f_k only depends on the previous state, current action, and current noise, but not on any older state or action. All relevant information for the decision making must therefore be summarized in the current state $\mathbf{x}_k$. The function f_k models the user's behavior or the state update given his current velocity, position and orientation from one time step to the next. We focus on the case where the uncertainty $\mathbf{w}_k$ models the probabilities of the limited, discrete set of possible future trajectories. In other words, f_k and $\mathbf{w}_k$ capture the map or the walking trajectories of the VE.

The total cost describing the undesirability of the redirected walking system accumulates over time

$$G = g_N(\mathbf{x}_N) + \sum_{k=0}^{N-1} g_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) \tag{2}$$

Where $g_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$ is the cost at each stage k and $g_N(\mathbf{x}_N)$ represents the terminal cost at the end of the planning horizon.

Due to the stochastic nature of the state evolution, the goal is to choose those actions that minimize the expected cost $\mathbb{E}[G]$. I.e. the cost incurred on different branches of the future trajectories are weighted with their probability. This results in an optimization problem that can be solved recursively using the dynamic programming algorithm:

$$\begin{aligned} J_N(\mathbf{x}_N) &= g_N(\mathbf{x}_N) && \text{initialization} \\ J_k(\mathbf{x}_k) &= \min_{\mathbf{u} \in U_k} \mathop{\mathbb{E}}_{\mathbf{w}_k} \left[g_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k) + J_{k+1}(\mathbf{x}_{k+1}) \right] \end{aligned} \tag{3}$$

The resulting $J_0(\mathbf{x}_0)$ is the optimal cost and the minimization in (3) gives the set of optimal control laws $\mu_k(\mathbf{x}_k)$ for each stage k, see [1]. The control laws provide the optimal action $\mathbf{u}_k$ for each state at the corresponding stage. However, to calculate the full set of control laws, the optimization would have to be performed over all possible states $\mathbf{x}_k \in \mathbb{R}^7$ and for all k. $J_k(\mathbf{x}_0)$ represents the cost-to-go at each stage k.

3.2 Actions from RETs

The complete set of actions C_k can be composed of different RETs with specific configurations or combinations thereof. E.g. a single action could be a strong clockwise rotation gain combined with a weak upscaling translation gain.

Below we summarize how RETs can be treated as actions and how the state update function looks like for these actions. We focus on rotational RETs as they are well researched and can be applied generically. For other RETs, we present a short sketch of the basic idea.

3.2.1 Rotational RET

Rotational RETs add some - typically subtle - rotation to a user's movement. Rotation gains up- or downscale a user's rotation and curvature gains add some rotation while a user is walking along a virtual straight line. A rotational RET function consisting of rotation and curvature gains is given by applying the max function [8, 15]

$$\Delta\hat{\phi}(\mathbf{x}) = \max \begin{cases} v_R \rho_c \Delta t & \text{curvature} \\ \omega_R \rho_q \Delta t & \text{rotation} \end{cases} \tag{4}$$

Therefore, (4) describes the rotational redirection $\Delta\hat{\phi}$ as an additive angle per time step that modifies to the user's real orientation Θ_R. ρ_c is the curvature gain as defined in [18]. The sign of ρ_c determines the direction of the redirection. ρ_q is the rotation gain which is different for rotations with or against the head movement. Due to the additive formulation of the rotational redirection function, ρ_q is given as $(1 - gain_R)$ for $gain_R$ as defined in [18]. A specific rotational RET action is now defined by a specific ρ_c and ρ_q.

3.2.2 Other RETs

- Translation gains scale a user's translational movement with the factor ρ_t. As for the rotational RET, a translational RET function can be formulated as a positional offset per time step. An action is thus a translational RET with a specific ρ_t.
- Reset techniques that reorient a user or reposition him are basically actions. An example is given in Section 4.3.
- Teleportation techniques [2] or similar metaphors can be transformed to actions like resets. The fact that actions can depend on the state makes it possible to associate these techniques with specific locations in the VE.
- Architectural illusions are more challenging to model as they modify the underlying system (state update function and probabilities). I.e. they are not actions. If only a few discrete changes are made to a VE's geometry as in [21], it is possible to use the online planning algorithm introduced below (the best cost-to-go is evaluated for all geometries and the cheapest geometry is selected). However, some techniques, e.g. [22] can also be used in parallel to the proposed approach.

3.3 State Update with RETs

The state update function models the effects of a redirection technique on a trajectory. To simplify the calculations below, we assume that the trajectory is given as a single curve $\gamma_V : [0,S] \to \mathbb{R}^2$. The curve is arc-length parametrized with a parameter $s \in [0,S]$, where $S \in \mathbb{R}$ is the total length of the curve. Furthermore, we assume that a function $s(t) : \mathbb{R} \to \mathbb{R}$ and its inverse exist and are differentiable. It associates the current arc position s given the current time $t \in [0,T]$. E.g. in the simplest case $s(t) = v_R t$ where v_R is the constant walking velocity (this is also used in our study setup).

This means, the state update function (1) for a single future trajectory transforms the virtual trajectory $\gamma_V(s)$ to a real trajectory $\gamma_R(s)$ if a rotational or translational RET is applied. These transformations should be done analytically to reduce the computational costs for planning.

3.3.1 Rotational RET

A rotational RET applies a curvature transformation to a trajectory curve. Arc-length parametrized curves can be defined by integrating over its unit tangent vector. Hence, given the user's real orientation $\Theta_R(s)$, the tangent vector is given using the sine and cosine. The transformed real curve then becomes

$$\gamma_R(s) = \int_0^s \begin{bmatrix} \cos(\Theta_R(\sigma)) \\ \sin(\Theta_R(\sigma)) \end{bmatrix} \mathrm{d}\sigma \ + \mathbf{p}_{R0} \tag{5}$$

Where $\mathbf{p}_{R0}$ represents the initial real position of the user. To calculate the user's real orientation, we integrate over the curve's curvature. It is given by the sum of the curve's virtual curvature $\kappa_V = \frac{\mathrm{d}\Theta_V(\sigma)}{\mathrm{d}\sigma}$ and the rotational RET function,

$$\Theta_R(s) = \int_0^s \frac{\mathrm{d}\Theta_V(\sigma)}{\mathrm{d}\sigma} + \frac{\mathrm{d}\phi}{\mathrm{d}\sigma}\, \mathrm{d}\sigma \ + \Theta_{R0} \tag{6}$$

Θ_{R0} is the user's initial orientation. However, in order to analytically transform a given curve using a rotational RET, a continuous formulation is required, i.e. we need $\frac{\mathrm{d}\phi}{\mathrm{d}s}$. From (4) we can derive $\frac{\mathrm{d}\hat{\phi}}{\mathrm{d}t}$ and by applying the chain rule using $t(s)$ we get

$$\frac{\mathrm{d}\phi}{\mathrm{d}s} = \max \begin{cases} \rho_c & \text{curvature} \\ \frac{\mathrm{d}\Theta_R(s)}{\mathrm{d}s}\rho_q = \kappa_R(s)\,\rho_q & \text{rotation} \end{cases} \tag{7}$$

Transformation of a Regular Arc Segment Within the study presented below, trajectories are modeled using regular arc segments and straight line segments. Hence, we show the transformation of such a segment. We assume that the curvature is high so that in (7) the rotational gain redirection has to be applied. For a regular arc we have $\kappa_V = 1/r_V$ where r_V is the arc's radius. Thus, from inserting (7) into (6) we get

$$\Theta_R(s) = \int_0^s \frac{1}{r_V} + \rho_q \frac{\mathrm{d}\Theta_R(\sigma)}{\mathrm{d}\sigma}\, \mathrm{d}\sigma \ + \Theta_{R0} \tag{8}$$

This is an ordinary differential equation which can be solved for $\frac{\mathrm{d}\Theta_R}{\mathrm{d}\sigma}$ by differentiating by $\mathrm{d}\sigma$. The resulting integral then becomes

$$\begin{aligned} \Theta_R(s) &= \int_0^s \frac{1}{r_V} + \frac{\rho_q}{r_V(1-\rho_q)}\, \mathrm{d}\sigma + \Theta_{R0} \\ &= s\kappa_R + \Theta_{R0} \\ \text{with } \kappa_R &= \left(\frac{1}{r_V(1-\rho_q)}\right) \end{aligned} \tag{9}$$

Applying the same transformation using curvature gain redirection in (7) gives a different $\kappa_R = \left(\frac{1}{r_V} + \rho_c\right)$. The resulting transformed curve remains a regular arc and is given by

$$\begin{aligned} \gamma_R(s) &= \int_0^s \begin{bmatrix} \cos(\sigma\kappa_R + \Theta_{R0}) \\ \sin(\sigma\kappa_R + \Theta_{R0}) \end{bmatrix} \mathrm{d}\sigma + \mathbf{p}_{R0} \\ &= \frac{1}{\kappa_R} \begin{bmatrix} \sin(\Theta_{R0} + \kappa_R s) - \sin((\Theta_{R0}) \\ \cos(\Theta_{R0}) - \cos(\Theta_{R0} + \kappa_R s) \end{bmatrix} + \mathbf{p}_{R0} \end{aligned} \tag{10}$$

Once the suitable κ_R is calculated, (10) provides the user's path in the real room for a given s. (10) also correctly describes transformed straight line segments. In fact, this is just a limit case for $r_V \to \infty$. As expected for straight lines, only a curvature gain redirection will change the curve.

3.3.2 Other RETs

- The state update for a translational RET simply changes the virtual speed of a user walking along the real curve $\gamma_R(s)$ and thus scales the curve. However, it is not clear how to apply a translational RET in parallel to a rotational RET as by scaling the trajectory, a translational RET also changes the curvature of the trajectory. Hence, predicting a user's behavior becomes difficult and it is also unclear how a user compensates the scaling, see also [11].
- Reset techniques simply update the user's real position or orientation while keeping the virtual position and orientation constant.
- Similarly, teleportation techniques perform a single discrete update of the virtal position/orienation and the real position/orientation.

3.4 Cost

The choice of the cost function $g_k(\mathbf{x}_k, \mathbf{u}_k, \mathbf{w}_k)$ is the primary measure to adjust how the controller decides. For instance, real room positions close to the real room boundaries could be assigned high costs. Furthermore, to avoid real positions outside the room boundaries, such illegal states are assigned infinite costs. I.e. the physical room's geometry is modeled with a cost function that takes the user's real position and performs a collision check. This way arbitrary real room geometries are supported as long as efficient collision checks can be performed.

However, subtle RETs are more preferable than resets. So action costs can be assigned that give the reset actions a higher cost. Hence, the resulting controller will try to avoid expensive states by applying RETs and will favor cheaper RETs if possible. Assigning costs to actions is crucial if perceivable RETs are employed. If the problem has a terminal state, e.g. at the end of a VE path, the terminal cost $g_N(\mathbf{x}_N)$ describes its undesirability.

In general, finding a suitable cost function for controlling RETs is out of the scope of this paper. Engel et al. [3] showed how such a cost function based on the noticeability might look like. In the study presented below, a cost function for rotational RETs and simple resets is proposed and evaluated.

3.5 Optimization

3.5.1 Offline Planning

In (3) we have shown the dynamic programming algorithm for the redirection problem. However, it requires to perform the optimization over all possible states $\mathbf{x}_k \in \mathbb{R}^7$. This is typically hard to model and solve analytically. The state update function is nonlinear or discrete for some RETs and the geometry cannot be modeled easily as a continuous function. Therefore, the state space has to be discretized.

In addition, the time horizon is often not finite. I.e. the user keeps walking around in the VE and we neither know when he stops walking nor what his terminal state will be. Therefore, it makes sense to exclude the terminal state and formulate the problem as infinite horizon optimal control problem. Excluding the terminal state requires adding a discount factor which is discussed in the next section. Further, if the system (1) is time invariant and the costs are bounded from above, the system satisfies the Bellman equation [1]. This allows applying efficient methods like policy iteration or value iteration to solve for the optimal control law.

Nevertheless, finding the control law for all possible discrete states is infeasible in practice as the state space can become huge. In addition, some RETs like the curvature gain change the user's state only by a small amount which requires a fine discretization.

3.5.2 Online Planning

Instead of precalculating the control law, we propose an online planning algorithm. Our algorithm 1, MPCRed, is based on model predictive control concepts. I.e. it takes the user's current state and recursively applies all actions to all possible future trajectory segments. This way it finds the best action that has to be applied now. The planning horizon is fixed and defines the algorithm's recursion depth. MPCRed does not require any state discretization but only considers a limited time horizon for finding the best action. It runs continuously to adapt to unpredicted changes. The cost update in line 14 contains a discount factor $\alpha \in (0,1]$ that allows reducing the effect of future costs on the current action. E.g. the stage costs at the end of the planning horizon N are weighted with α^{N-1}. This increases the stability of the controller. α therefore adjusts if and how much the controller tries to shift expensive costs into the future. Similarly in line 14, the cost is weighted with the probability of the trajectory segment to consider only its relative importance.

Computationally, MPCRed is similar to a depth first search algorithm. This means that the computational cost grows exponentially with the horizon. In the worst case, it is in $O((|C|D)^N)$ where $|C|$ is the number of actions, D is the number of possible future directions per planning stage, and N is the planning horizon or the recursion depth. This limits the feasible maximum planning horizon. To reduce the computation costs, MPCRed employs efficient cost bounds to remove branches for irrelevant actions, see lines 5 and 9. Furthermore, it applies the actions sorted by their costs.

Algorithm 1 MPCRed online planning algorithm

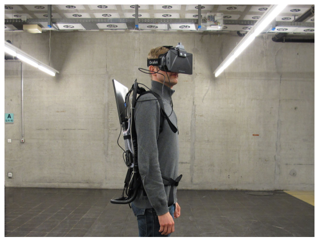
```
1: function PLAN($X_{current}, depth_{recursion}$)
2:     $bestcost \leftarrow \infty$                        ▷ initialize best cost to infinity
3:     for all $u \in Actions$ do ▷ loop through by increasing cost(u)
4:         $c \leftarrow 0$                              ▷ c is the current cost counter
5:         if $cost(u) < bestcost$ then
6:             for all $seg \in$ GETSEGMENTS($X_{current}$) do
7:                 $[X_{next}, stagecost] \leftarrow$ APPLY($u, X_{current}, seg$)
8:                 $c \leftarrow c + probability(seg) * stagecost$
9:                 if $c \geq bestcost$ then
10:                    break
11:                end if
12:                if $depth_{recursion} > 0$ then
                                                ▷ continue recursively on $X_{next}$
13:                    $[nextaction, nextcost] \leftarrow$
                               PLAN($X_{next}, depth_{recursion} - 1$)
14:                    $c \leftarrow c + \alpha * probability(seg) * nextcost$
15:                end if
16:            end for
17:            if $c < bestcost$ then
18:                $bestcost \leftarrow c$
19:                $bestaction \leftarrow u$
20:            end if
21:        end if
22:    end for
23:    return $[bestaction, bestcost]$
24: end function
```

3.6 Implementation

The suggested approach based on model predictive control decouples the optimization process from the actual application of the action on tracking data. When tracking data is received by the software, it just has to look up the most recent action and apply it. This is similar as for other steering algorithms (compare [13, 9, 8]) and adds virtually no latency to the system. The planning can run as an independent thread at a far lower frequency, e.g at 0.4 Hz as used during the evaluation. It is usually specified by the stage duration τ. In a typical redirected walking setup, a τ of 1 to 4 sec is suitable. This provides enough time for a RET to take effect. The total duration for the planning horizon is then given by $\tau \times N$.

3.7 Relation to Existing Algorithms

The S2C algorithm can be regarded as a simplified case of MPCRed. Its basic working principle according to [13, 9, 8] is as follows. S2C continuously applies rotational RETs to orient a user towards the center of the real room. I.e. at every frame (τ_{S2C}=1/framerate) the user's future direction is predicted using his past virtual positions (e.g. over a 1 second backlog). The virtual direction is then transformed to the walking direction in the real room. Finally, for every frame, a rotational RET is applied in the direction that decreases the absolute angular deviation between the user's walking direction and the vector pointing from his position to the room's center. Hence, in terms of MPCRed, S2C is a greedy approach with a planning horizon of N=1 and employs two rotational RETs (left and right rotation) in the direction that minimizes the stage cost. The stage cost is given by the absolute deviation angle of the user's real orientation to the room's center. The future


(a)

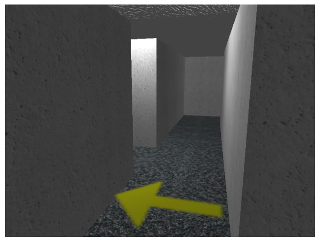

(b)

Figure 1: (a) The wearable VR system. (b) The participant's view on the virtual environment used for the evaluation.

trajectory is simply assumed to be the current walking direction. Typically, S2C refrains from redirection only if the user is oriented approximately towards the room's center or is very close to it.

FORCE [25] is a special case of MPCRed where only terminal costs are evaluated and the horizon is given by the predefined paths which have terminal positions. Additionally, FORCE uses infinite stage costs to avoid collisions and has a discount factor $\alpha = 1$.

4 Evaluation

A user study was conducted to evaluate MPCRed. For comparison, another generalized redirected walking algorithm was required. For this purpose, the S2C algorithm was chosen. S2C was shown to outperform other generalized steer-to-target algorithms in many conditions or at least has a similar performance [8]. Hence, a comparative study was designed using a within-subjects design including two conditions. FORCE [25] would also be a candidate for comparison. However, it is not clear how to choose the proper terminal state costs and integrate a reset technique.

4.1 Equipment

Hardware An Intersense IS-1200 tracking system [6] was installed in a 12.6 m $\times$ 6.2 m room (tracks 1 point with 6 degrees of freedom at 180Hz). The tracking device was attached to an Oculus Rift[1] HMD (640x800 resolution per eye, 110 degrees diagonal field of view). Participants wore the HMD and a backpack with a laptop computer for processing the tracking data and rendering the virtual scene. The laptop was equipped with an Intel Core i7-2630QM CPU running at 2.0GHz, 8GB RAM and an NVIDIA Quadro 1000M graphics card. Figure 1(a) shows the wearable system in the tracking space. Since safety margins of 0.6 m were added to all sides of the room, the available space for the study was reduced to 11.4 m $\times$ 5 m.

Software For rendering the virtual environment, the Unity3D[2] game engine was used. The steering algorithms, the optimization and the logging features for the study were developed as a multi-threaded C++ library. The library was directly accessed in Unity3D using its native plugin feature to have minimal overhead.

4.2 Experimental Setup

4.2.1 Virtual Environment

The MPCRed algorithm requires a map of the trajectories in a VE. However, generating such trajectories from a given geometry or points of interest is out of the scope of this paper and is still an open research question. E.g. [4, 5] present dynamic walking models which can generate natural human walking trajectories numerically. Hence, we designed a VE that constrained the walk paths so that the trajectories could be predefined. The layout of the constructed VE is shown in Figure 2. The layout of the VE is motivated by the paths found in a local furniture store. However, it was scaled down so that the path used for the study had an approximate length of 100 m. The VE had a corridor width of 1.6 m and contained 5 crossings. The rendered view of the VE is shown in Figure 1(b).

In order to have all participants walking the same path for the evaluation, direction signs were added at each crossing, see Figure 1(b). The circular path for the study and the start/terminal position is also shown in Figure 2 (participants had to turn left at the first crossing).

[1]www.oculusvr.com

[2]www.unity3d.com

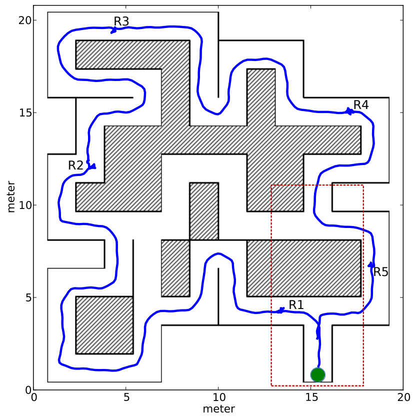


Figure 2: Layout of the study VE and a recorded virtual trajectory. The green point marks the start/terminal position. Participants walked clockwise. The red dashed rectangle shows the size of the real room (start configuration). R1 to R5 label the resets.

4.2.2 Trajectory Prediction

The trajectories of the study VE were predefined using a bidirectional graph that also allowed cycles. The graph's edges were modeled as trajectory primitives. Such primitives were either straight line segments for the corridors or regular arcs for the turns. The arcs modeling the turns had a radius of 0.8 m and a turn angle of -90°, -180°, 90°, or 180°. Clearly, this is only an approximation for the true walking trajectories. However, this simplified model has shown to be sufficient for the optimization. The vertices therefore either modeled crossings and/or switches between different trajectory primitives.

For predicting the future trajectories, a participant's virtual position was mapped to the closest trajectory primitives in the graph. Then, the user's smoothed virtual orientation was used to determine the correct trajectory primitive, i.e. matching the direction. A similar approach was used by Peck et al. [13]. The smoothed virtual orientation was determined by exponentially smoothing the virtual head positions over an approximate window of 1 second (smoothing factor 0.009 at 180Hz). If the participant was not walking ($\Leftrightarrow$ speed $\leq$ 0.2 m/sec) the exponentially smoothed facing direction was used as orientation (smoothing factor 0.004 at 180Hz).

MPCRed needs to know the future set of trajectory primitives for planning. Thus, the trajectory graph had to be resampled for a given stage length to provide a list of primitives for a total length equal to the required stage length and an associated probability.

At crossings, all directions were given equal probability and a forward walking assumption was applied. I.e. when a participant arrives at a 'T'-shaped crossing from below, the predictor only considers left or right as future directions and both have the probability 0.5. It should be noted that although the path was fully defined for the participants, the planning algorithm considered all future directions at crossings during the study.

4.2.3 Reset Technique

For both study conditions explained below, a reset technique was required. A reset stops a participant and reorients or repositions him in the real room if a collision with the room's boundaries is unavoidable. We decided to use a pure reorientation reset technique as proposed in [24]. When a reset had to be done, a large green arrow appeared on top of the VE that instructed the participant to stop walking and turn on the spot in the direction the arrow was pointing. Participants had to do a full 360° turn in the VE. However, depending on the study condition, they turned between 390° to 540° in the real room. When the reset was done, the arrow changed to a tick mark and then disappeared to notify the participants that they may continue walking. With this reset technique, the applied rotation gains comply with the detection thresholds determined in [18].

4.3 Conditions

As we wanted to evaluate and compare the performance of MPCRed, we conducted a study with two conditions. In the MPCRed condition, the MPCRed algorithm was used to steer a participant while he walked along the predefined path in the study VE. In the other condition, the S2C algorithm was used. Both conditions were based on the same VE and path.

Both algorithms utilized the same rotational RET and reset technique. For the rotational RET, the curvature gain ρ_c was 1/7.5 [8]. The rotation gain ρ_q was (1-0.67) for rotations with and (1-1.24) for rotations against the head movement [18].

4.3.1 S2C Condition

The S2C algorithm was realized as proposed by [8]. The damping distance to the room's center position was 1.25 m and the damping angle range was 35°. I.e. the rotational redirection was reduced or turned off when a user was standing close to the room center or walking approximately towards it. We performed no damping of the applied rotation and no temporary steering targets were generated for a user walking away from the room center. Instead, the trajectory graph was used to retrieve a robust direction prediction. Reorientation resets were activated whenever a user reached the boundary of the real room and turned the user towards the center of the real room, see also [13].

4.3.2 MPCRed Condition

We defined 14 different actions for the MPCRed algorithm. A "zero" action represented the action where no redirection was applied at all. Two actions were defined using the rotational RET. One that turned a participant clockwise and one that turned him counterclockwise. The other 11 actions defined different rotation angles of the reorientation reset. The reset actions turned a participant ±30, ±60, ±90, ±120, ±150, or 180 degrees.

Previous tests have shown that the suitable time horizon which results in low average computation times is 8. We defined the cost of the rotational RET action as 1. Hence the cost of the resets must be defined relative to this cost. Our goal was to force the algorithm to avoid or postpone a reset action whenever possible. Hence, we defined the reset cost as 500. Basically, this is just a large number for which the discounted cost at the end of the horizon ($\alpha^{8-1} \times 500$) is still larger than any rotational RET action. In other words, the algorithm should always use a rotational RET to avoid resets.

0.8 was found to be a suitable discount factor α. The stage duration was 2.5 sec and the participant's exponentially smoothed walking velocity was used to determine the stage length (smoothing factor 0.001 at 180Hz). The stage costs were only given by the action costs except if the applied action caused a collision. In that case, the stage costs were infinite.

4.4 Participants

A total of 26 participants (19 male, 7 female) recruited among the department took part in the user study. Their age was between 22 and 37 years (median=28.5 years, σ=4.1). All participants had normal or corrected-to-normal vision. Participants with corrected-to-normal vision wore contact lenses and not glasses as a requirement. This avoided problems with adjusting the Oculus Rift HMD.

4.5 Procedure and Task

In order to eliminate training effects, the order of the two conditions was rotated and counterbalanced. An initial training session also had the purpose to further reduce order effects.

First, the participants were introduced to the wearable VR system and received instructions how the wear and adjust the HMD and backpack. Then, all participants were allowed to test the system in the training VE for 5 minutes. During the training, participants received further instructions regarding the direction signs and the reset techniques. After the training, they had to walk to the marked start position in the tracking space. They were told that their task is to walk naturally through the study VE, follow the corridors and the direction signs at the crossings. The walk was finished when they arrived again at the (virtual) start position marked with a green pillar. This procedure was repeated for the second condition.

The simulator sickness was measured using Kennedy's Simulator Sickness Questionnaire (SSQ). All participants answered the SSQ three times: before the training, after the first condition, and after the second condition at the end of the study.

4.6 Measures

By design, generalized redirected walking algorithms try to prevent the user from leaving the boundary of the tracking space. Hence the most relevant performance measure is the number of wall contacts, i.e. the number of reorientation resets. Further, the resets cause breaks in presence and are clearly noticed by the user whereas the rotational RET should remain unnoticed by study design. Finally, the number of resets is also a technical measure because it allows comparing the costs between MPCRed (resets are the most significant costs) and S2C even though S2C has no notion of cost.

Similarly, the injected rotation for redirection is minimized. S2C only applies a redirection if the user is facing away from the center of the tracking space and MPCRed tries to minimize the cost of the applied redirection. Therefore, the mean rate of redirection is also considered for the comparison. It is defined as the mean of the unsigned injected redirection rotation per second for a single walk through the study environment excluding the resets and therefore measures the effect of the rotational RETs. It is technically independent of the number of resets.

Further general measures are the total time required for a walk, the mean walk speed, the walked distance, and the mean time between the resets. For the MPCRed condition, we also analyzed the times required for the computation of the optimal action.

5 Results

Out of 26 participants, 24 successfully completed the evaluation study (6 women and 18 men). One participant did not follow the defined path correctly and another had problems with wearing the backpack which caused an unnatural walking behavior. The following results are based only on the data from these 24 participants. For both genders, the order of the conditions were counterbalanced.

On average for all participants and both conditions, the distance for a single walk through the study VE had a length of 108.6 m with a standard deviation σ=6.0. The mean walk speed was 0.64 m/sec, σ=0.08 and the mean duration was 208 sec, σ=41.

For comparing both conditions, 2-tailed t-tests were performed. We report the means, the standard deviations, and the corresponding t and p values.

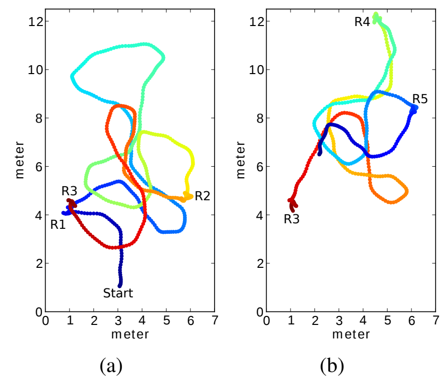


Figure 3: Real room trajectory of the virtual trajectory shown in Figure 2, MPCRed condition. The trajectory is split in 2 parts at the third reset ((a) first part, (b) second part). The trajectory is colored to distinguish overlapping segments. R1 to R5 label the resets.

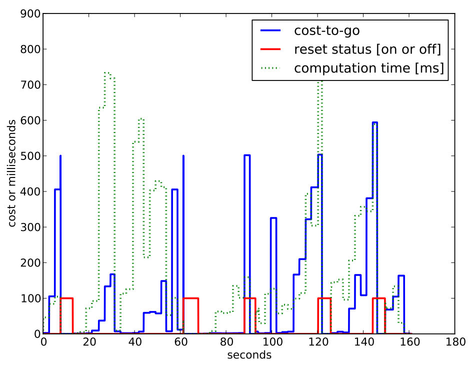


Figure 4: System evolution of the MPCRed algorithm for the trajectory shown in Figure 3.

One participant answered the SSQ incorrectly and was excluded from the evaluation of the SSQ. The mean pre-SSQ score for the remaining 23 participants was 4.0 with σ=3.8. The post-SSQ score after the MPCRed condition averaged to 7.3, σ=5.6 and for the S2C condition to 7.35, σ=5.8. There was no significant difference between the two conditions (t(22)=0.082, p=0.935). The significant differences between the pre-SSQ and post-SSQ for both conditions were similar (pre-SSQ to S2C condition (t(22)=3.27, p<0.01), pre-SSQ to MPCRed condition (t(22)=3.39, p<0.01)).

5.1 MPCRed

A representative trajectory recorded during the study in the MPCRed condition is shown in Figure 2. The corresponding trajectory in the real room is given in Figure 3. In both, the virtual and the real trajectory, the resets are labeled and visible as point-like marks on the trajectory. The corresponding participant had to do 5 resets to complete the walk. The real trajectory in Figure 3 is split into two parts at the third reset.

The evolution of the MPCRed algorithm over time for the trajectory in Figure 3 is shown in Figure 4. It shows the currently estimated best cost-to-go, the computation time per epoch (=2.5 sec), and when the participant had to do a reset.

Computation Time The mean computation time for all calculated redirection actions during the study was 82.5 millisec, with σ=145.1. Typically the computation time was below 200 millisec. However, for 6 single actions the computation time was above 1 sec and the maximum overall encountered computation time was 1826 millisec. In the study VE, the computational cost was in $O((14 \times D)^8)$ given the 14 actions, a time horizon of 8 and a trajectory branching factor D that varied from 1 to 3. On average however, D was between 1 and 2.

5.2 Comparison

Number of Resets The mean number of resets in the MPCRed condition was 6.63, σ=1.44 per walk through the study VE. During the S2C condition, participants had to do 11.25 resets on average, σ=2.11. The MPCRed algorithm significantly reduced the number of required resets (41% reduction) compared to the S2C algorithm (t(23)=8.15, p<0.001).

Mean Rate of Redirection The mean rate of rotational redirection in the MPCRed condition was 0.088 rad/sec, σ=0.024 and during the S2C condition 0.120 rad/sec, σ=0.021. The MPCRed algorithm significantly reduced the mean rate of redirection (27% reduction) compared to the S2C algorithm (t(23)=5.50, p<0.001).

Other Measures The fact that the number of resets is reduced significantly in the MPCRed condition clearly affects the task completion time and the mean walk speed. The same holds true for the mean time between two resets. Hence, we omit the further comparison of these measures.

6 Discussion

Summarizing from the results, MPCRed provides a significant improvement over the S2C algorithm. Further, it proves that an MPC based RET controller is realizable in practice if the trajectories of the VE can be determined. On the other hand, MPCRed has a significant advantage over S2C as it knows the future trajectories. Under this perspective, S2C actually performs well as it requires only about twice as many resets as MPCRed. The main reason is that the redirection is always active (if the user is not facing towards the room center). I.e. almost any head rotation will cause a redirection. In contrast, MPCRed often applies no redirection (zero RET) if redirection is not considered useful. Therefore, unpredicted head movements will have no effect. In fact, the known trajectories just provide a conservative estimate for the user movements and typically a “looking around” behavior is not modeled.

The computation time during the whole study was low given that the worst case costs for the planning are in $O((14 \times 3)^8)$ considering the crossings. This proves that the cost bounds effectively reduce the computation time to a feasible range. This however comes at the expense that we do not know a priori how long the computation will take, which explains the high variation of the computation time in Figure 4. E.g. at reset 4, the computation time is high because the user is close to a real room corner where several cheaper actions cause a collision and the bounds do not cut off branches as effectively. Further, the varying number of trajectory branches D at crossings also influences the computation time. It should be noted that the number of actions $|C|$ and the branches D can strongly limit the feasible time horizon N for solving the optimization problem. The size of the real room has a similar influence. If the user does not reach the real room boundaries in most cases within the time horizon, MPCRed will apply no RET or cheap RETs only and the computation time will be low. On the other hand for a very small room, more collisions must be considered. During the study, it is likely that a slightly wider room would have improved the results of MPCRed. This is because MPCRed often aligned straight trajectory segments along the real room’s diagonal as the room was slightly too narrow which still caused a reset at the other end.

In Figure 4 some problems of MPCRed can be seen. Usually before a reset occurs, the best cost-to-go gradually increases as expected. However, in some cases (e.g. see time index 60 sec) the algorithm suddenly decides that a reset is required. This happens when the algorithm previously determined that a reset is not needed but the prediction was incorrect. A potential solution is to use different stage costs for the geometry. So far, a discontinuous function was used, i.e. 0 costs for positions inside the real room and infinite costs outside the real room. Hence, the planner plans very close to the boundary. Instead, a continuous function could be employed to penalize the border area, see e.g. [11].

The main error sources of MPCRed are the differences between the real and the predefined trajectories and the computation time. When MPCRed optimizes for the next action, it uses the current state. However, at the time the action is determined and applied, the user has already moved on. In other words, there is a trade-off between planning horizon, computation time, and stage duration.

7 Conclusions and Future Work

Within this paper, we presented a generalized approach for designing redirection planning algorithms by showing how to treat the RDW problem as an optimal control problem. Based on the proposed control model, we designed and evaluated an efficient online planning algorithm that can determine suitable RETs with their parameters that have to be applied. As the proposed approach needs to know the user's future behavior for planning ahead, it is most suitable for goal oriented tasks and/or geometrically constrained environments.

Therefore, future work should focus on human path prediction methods which use the user's current walking behavior to determine the future trajectories with their likeliness. Additionally, heuristics could be employed to effectively reduce the amount of considered future trajectories. Further, cost functions are needed for different RETs including e.g. stronger or perceivable rotational RETs. The proposed model could be used to fuse offline planning with online planning to effectively increase the horizon while keeping the computational costs low. The offline planner could instead plan on a coarsely discretized state mesh with a reduced set of RETs. The resulting costs-to-go for each discretized state could also serve as a robust state evaluation function, e.g. for [25]. Finally, the proposed approach can be adapted to plan redirection for dynamic passive haptics, where a user can sense a virtual object by being guided/redirected to a real world proxy object.

Acknowledgements

The authors wish to thank Markus Zank and the Swiss National Science Foundation (SNSF project number 127298).

References

[1] D. P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1. Athena Scientific, 3rd edition, 2005.

[2] G. Bruder, F. Steinicke, and K. H. Hinrichs. Arch-explore: A natural user interface for immersive architectural walkthroughs. In *Symposium on 3D User Interfaces*, 3DUI '09, pages 75–82. IEEE, 2009.

[3] D. Engel, C. Curio, L. Tcheang, B. Mohler, and H. H. Bülthoff. A psychophysically calibrated controller for navigating through large environments in a limited free-walking space. In *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '08, pages 157–164. ACM, 2008.

[4] B. R. Fajen and W. H. Warren. Behavioral dynamics of steering, obstable avoidance, and route selection. *Journal of Experimental Psychology: Human Perception and Performance*, 29(2):343–362, 2003.

[5] P. W. Fink, P. S. Foo, and W. H. Warren. Obstacle avoidance during walking in real and virtual environments. *TAP '07: ACM Transactions on Applied Perception*, 4(1):2:1–2:23, Jan. 2007.

[6] E. Foxlin and L. Naimark. Vis-tracker: A wearable vision-inertial self-tracker. In *Proceedings of the IEEE Conference on Virtual Reality*, VR '03, pages 199–206. IEEE, 2003.

[7] H. Hicheur, Q.-C. Pham, G. Arechavaleta, J.-P. Laumond, and A. Berthoz. The formation of trajectories during goal-oriented locomotion inhumans. i. a stereotyped behaviour. *European Journal of Neuroscience*, 26(8):2376–2390, 2007.

[8] E. Hodgson and E. Bachmann. Comparing four approaches to generalized redirected walking: Simulation and live user data. *TVCG '13: IEEE Transactions on Visualization and Computer Graphics*, 19(4):634–643, 2013.

[9] E. Hodgson, E. Bachmann, and D. Waller. Redirected walking to explore virtual environments: Assessing the potential for spatial interference. *TAP '11: Transactions on Applied Perception*, 8(4):22:1–22:22, 2011.

[10] V. Interrante, B. Ries, and L. Anderson. Seven league boots: A new metaphor for augmented locomotion through moderately large scale immersive virtual environments. In *Symposium on 3D User Interfaces*, 3DUI '07, pages 167–170. IEEE, 2007.

[11] C. Neth, J. Souman, D. Engel, U. Kloos, H. Bülthoff, and B. Mohler. Velocity-dependent dynamic curvature gain for redirected walking. *TVCG '12: IEEE Transactions on Visualization and Computer Graphics*, 18(7):1041–1052, july 2012.

[12] N. Nitzsche, U. D. Hanebeck, and G. Schmidt. Motion compression for telepresent walking in large target environments. *Presence: Teleoperators and Virtual Environments*, 13(1):44–60, 2004.

[13] T. Peck, H. Fuchs, and M. Whitton. The design and evaluation of a large-scale real-walking locomotion interface. *TVCG '12: IEEE Transactions on Visualization and Computer Graphics*, 18(7):1053–1067, July 2012.

[14] T. C. Peck, H. Fuchs, and M. C. Whitton. Evaluation of reorientation techniques and distractors for walking in large virtual environments. *TVCG '09: IEEE Transactions on Visualization and Computer Graphics*, 15(3):383–394, 2009.

[15] S. Razzaque, Z. Kohn, and M. C. Whitton. Redirected walking. In *Proceedings of Eurographics*, pages 289–294, 2001.

[16] F. Steinicke, G. Bruder, K. Hinrichs, J. Jerald, H. Frenz, M. Lappe, J. Herder, S. Richir, and I. Thouvenin. Real walking through virtual environments by redirection techniques. *Journal of Virtual Reality and Broadcasting*, 6(2), 2009.

[17] F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Analyses of human sensitivity to redirected walking. In *Proceedings of the 2008 ACM symposium on Virtual reality software and technology*, VRST '08, pages 149–156. ACM, 2008.

[18] F. Steinicke, G. Bruder, J. Jerald, H. Frenz, and M. Lappe. Estimation of detection thresholds for redirected walking techniques. *TVCG '10: IEEE Transactions on Visualization and Computer Graphics*, 16(1):17–27, jan.-feb. 2010.

[19] J. Su. Motion compression for telepresence locomotion. *Presence: Teleoperators and Virtual Environments*, 16(4):385–398, 2007.

[20] E. Suma, G. Bruder, F. Steinicke, D. Krum, and M. Bolas. A taxonomy for deploying redirection techniques in immersive virtual environments. In *Virtual Reality Short Papers and Posters*, VRW '12, pages 43–46. IEEE, 2012.

[21] E. Suma, S. Clark, D. Krum, S. Finkelstein, M. Bolas, and Z. Warte. Leveraging change blindness for redirection in virtual environments. In *Proceedings of the IEEE Conference on Virtual Reality*, VR '11, pages 159–166. IEEE, 2011.

[22] E. Suma, Z. Lipps, S. Finkelstein, D. Krum, and M. Bolas. Impossible spaces: Maximizing natural walking in virtual environments with self-overlapping architecture. *TVCG '12: IEEE Transactions on Visualization and Computer Graphics*, 18(4):555–564, april 2012.

[23] B. Williams, G. Narasimham, T. P. McNamara, T. H. Carr, J. J. Rieser, and B. Bodenheimer. Updating orientation in large virtual environments using scaled translational gain. In *Proceedings of the 3rd symposium on Applied perception in graphics and visualization*, APGV '06, pages 21–28. ACM, 2006.

[24] 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 *Proceedings of the 4th symposium on Applied perception in graphics and visualization*, APGV '07, pages 41–48. ACM, 2007.

[25] M. Zmuda, J. Wonser, E. Bachmann, and E. Hodgson. Optimizing constrained-environment redirected walking instructions using search techniques. *TVCG '13: IEEE Transactions on Visualization and Computer Graphics*, 19(11):1872–1884, 2013.