A Steering Algorithm for Redirected Walking Using Reinforcement Learning

Ryan R. Strauss, Raghuram Ramanujan, Andrew Becker, and Tabitha C. Peck *Member, IEEE*

Fig. 1. Three different steering algorithms simulated on a real user's path through a virtual environment. The physical tracking space is a square with 5.79 meter sides. The user's virtual distance traveled was 6.4 meters with no redirection (left), 8.5 meters with steer-to-center (middle), and 9.2 meters with the Steering algorithm learned via Reinforcement Learning (SRL) (right). SRL resulted in a more arched path than steer-to-center, which allowed it to reach a greater virtual distance before colliding with the boundary of the tracking space. Gaps in the paths are a result of resetting that occurred when a user collided with a boundary of the tracking space, as we do not consider movement during the reset to be part of the virtual path. The user started in the bottom center of the environment and walked north. The passage of time is indicated by the light to dark transition.

Abstract— Redirected Walking (RDW) steering algorithms have traditionally relied on human-engineered logic. However, recent advances in reinforcement learning (RL) have produced systems that surpass human performance on a variety of control tasks. This paper investigates the potential of using RL to develop a novel reactive steering algorithm for RDW. Our approach uses RL to train a deep neural network that directly prescribes the rotation, translation, and curvature gains to transform a virtual environment given a user's position and orientation in the tracked space. We compare our learned algorithm to steer-to-center using simulated and real paths. We found that our algorithm outperforms steer-to-center on simulated paths, and found no significant difference on distance traveled on real paths. We demonstrate that when modeled as a continuous control problem, RDW is a suitable domain for RL, and moving forward, our general framework provides a promising path towards an optimal RDW steering algorithm.

Index Terms—Virtual Reality, Locomotion, Redirected Walking, Steering Algorithms, Reinforcement Learning

1 Introduction

It is well known that walking is the most natural method of travel within virtual environments (VEs) [54]. However, the use of a real-walking locomotion technique constrains the VE to the size of the tracked space. Redirected walking (RDW) is a common locomotion technique that enables participants to really walk in VEs that are larger than the tracked space by imperceptibly transforming the VE around the user [40]. The original RDW implementation imposed the constraint that participants walk along a predefined path guided by waypoints. Follow-up research has focused on relaxing this constraint to enable free exploration of larger-than-tracked-space VEs.

Waypoints and predefined paths give the developer knowledge of a user's future path. When such information is available, it can be exploited effectively by *predictive algorithms* to transform the VE in a manner that places the user's trajectory within the tracked-space. *Reactive algorithms*, on the other hand, can operate in the absence of predefined paths and thus offer a more general solution for RDW. However, these algorithms require the developer to be able to forecast the user's future path; an incorrect prediction may result in the user's path straying outside the bounds of the tracked-space. Indeed, predictive algorithms are known to regularly outperform reactive algorithms due to this very reason [32, 58].

If a user reaches the bounds of the tracked-space as they attempt to walk to their desired virtual location, a hard reset will be required and the user will be forced to stop their motion, or risk leaving the tracked-space and walking into a physical wall. Reorientation and resetting techniques have been developed for these worst-case situations. These techniques stop the user or redirect their motion and can cause a break-in-presence or a disruption to the virtual experience [12, 37, 56]. In addition to reorientation techniques, researchers have also investigated other adjustments such as interactively modifying the VE [49, 50].

Reactive RDW implementations that allow free-exploration focus on reducing the total number of these worst-case situations, where the user is about to leave the tracked space. Previous research, through simulation and user studies, has evaluated numerous steering algorithms [5, 18]. Steering algorithms, such as *steer-to-center*, *steer-to-orbit* [39], and steering with artificial potential functions [53] have been developed to determine how to best transform the VE so as to keep

• *Ryan R. Strauss, Raghuram Ramanujan, and Tabitha C. Peck are with Davidson College. E-mail: {rystrauss | raramanujan | tapeck}@davidson.edu .*

Manuscript received 10 Sept. 2019; accepted 5 Feb. 2020.
Date of publication 18 Feb. 2020; date of current version 27 Mar. 2020.
Digital Object Identifier no. 10.1109/TVCG.2020.2973060

the user within the tracked-space. These reactive steering algorithms have been painstakingly crafted by human designers, who have deep intuitions about RDW and how best to steer users. However, recent advances in reinforcement learning (RL), a branch of artificial intelligence concerned with sequential decision-making, raise an intriguing possibility: could a computer, starting from a blank slate, discover a superior reactive steering algorithm purely through experimentation and by learning from experience? In this work, we investigate this very question.

This paper brings together redirected walking and reinforcement learning, both defined in section 2, by outlining a framework for the end-to-end learning of steering algorithms. Our system, which is explained in section 3, trains a neural network to directly prescribe redirection gains based on the user's position and orientation within a tracking space. In section 4, we apply our system to simulated and real user paths, and in section 5 we find that it performs significantly better than steer-to-center on simulated paths and has no difference on real paths. Finally, we discuss the results and avenues for future work in section 6.

2 Previous Work

2.1 Redirected Walking

RDW was proposed by Razzaque et al. as a technique for exploring large VEs while really walking in a limited physical space [40]. RDW allows users to experience the benefits of real walking, which is more natural and provides a greater sense of presence, while mitigating the physical limitations of a smaller tracking area. The generic form of RDW, as described by Steinicke et al., specifies an interface of three *gains* with which the user's motion can be manipulated in the VE [48]. The gains control the extent to which the user's *rotation*, *translation*, or *curvature* is compressed or expanded in the VE. When gains are applied within certain thresholds, the discrepancy between movement in the real and virtual worlds is imperceptible [47], and users are able to feel as though they are naturally exploring VEs that are larger than the physical space they inhabit. See Nilsson et al. for a recent overview of RDW techniques [34].

If the user's virtual path is known beforehand, predictive redirection algorithms can be applied such that the user's future path is kept within the tracking area [32, 58]. Unfortunately, the problem is seldom this simple as the path is generally not known ahead of time. Reasoning about what gains should be applied must rely solely on information about the user's current and past positions within the tracking space. Several heuristic-based steering algorithms have been developed to address this problem, including *steer-to-center* (S2C), *steer-to-orbit*, and *steer-to-multiple-targets* [39]. These algorithms seek to determine what gains should be applied at any point in time so as to minimize the number of required hard resets. S2C, an algorithm that attempts to steer the user back to the center of the tracked space, regularly outperforms other steering algorithms regardless of environment shape [4, 18]. Results suggest that steer-to-orbit, an algorithm that steers the user to walk in a circle, may outperform S2C when walking on long straight paths [18]. Recent work by Thomas and Suma Rosenburg demonstrated that their Push/Pull Reactive (P2R) algorithm outperforms S2C when obstacles are placed within the tracked space. However, the P2R algorithm did not outperform S2C in a convex environment with no obstacles [53].

The steering algorithms discussed thus far all rely on heuristics crafted by humans, and as such, their efficacy is subject to the same limitations as human intuitions about how best to redirect walkers in VEs. RL approaches offer an intriguing and rigorous data-driven alternative. They have notched up a series of significant successes in recent years in challenging domains where previous state-of-the-art methods required significant human input [7, 21, 25, 31, 35, 46]. Yet, there is little prior research on bringing RL methods to bear on RDW steering algorithms. One exception is the work of Lee et al., who proposed the *steer-to-optimal-target* (S2OT) algorithm. This algorithm uses RL to select a location within the tracking area towards which the user should be steered [27]. S2OT was found to result in significantly fewer hard resets than S2C. However, S2OT still relies on human intuition to define the discrete steer-to-target locations. On the other hand, the work of Chang et al. attempts to use RL to directly select transformation gains, therefore solving the harder continuous control problem and aligning most closely with our own work [8]. However, the focus of their work is on environments with obstacles, and in a large (15×15 meter), unobstructed tracking space, their method does not reduce the number of required hard resets compared to a heuristic controller. To the best of our knowledge, no reactive steering algorithm that directly chooses all three transformation gains has been shown to outperform heuristic controllers in unobstructed environments. The focus of this paper is to investigate the potential of RL to determine an optimal RDW steering algorithm and surpass current heuristic-based methods.

2.2 Reinforcement Learning

Reinforcement learning (RL) is a setting in which a learning agent interacts with an environment and, through trial and error, learns how to behave so as to achieve some goal [52]. The RL problem is formulated as a Markov Decision Process [38], formally defined by the 5-tuple (S, A, T, R, γ), where

- S is the (finite) set of states of the environment,
- A is the (finite) set of actions the agent can take,
- $T : S \times A \times S \to [0, 1]$ is the transition function that captures the dynamics of the environment, defining a distribution over the states s' that can be reached by taking action a in state s,
- $R : S \times A \to \mathbb{R}$ is a function that defines the reward received by the agent after taking action a in a state s, and
- $\gamma \in [0, 1)$ is the factor by which the rewards the agent receives in the future are discounted.

Informally, at each time step t, the agent finds itself in some state s_t. It takes an action a_t, collects a reward $r_t = R(s_t, a_t)$ and then enters the state s_{t+1} by sampling from all possible states s' according to $T(s_t, a_t, s')$, at which point the process repeats. The agent's objective is to maximize the expected sum of discounted rewards, or *return*, it receives over the long run, i.e. to maximize:

$$\mathbb{E}\left[\sum_{t=1}^{t=\infty} \gamma^{t-1} r_t\right] = \mathbb{E}\left[r_1 + \gamma r_2 + \gamma^2 r_3 + \gamma^3 r_4 + \ldots\right]$$

A *policy*, $\pi : S \to A$, is a function that prescribes an action to take in any given state. The RL problem is to find the optimal policy π^* for the agent that would maximize its return, when T and R are unknown. There are two standard approaches for solving for π^*. *Value-based* approaches, like the classic Q-learning algorithm [55], use dynamic programming to estimate the long-term utility of a state. In particular, Q-learning uses an iterative algorithm to estimate for all state-action pairs (s, a), the value of $Q(s, a)$ — the return that would be obtained from taking action a in state s, and acting optimally thereafter. Once learned, the optimal policy can be extracted from the Q-function using greedy one-step look-ahead: $\pi^*(s) = \text{argmax}_a Q(s, a)$. *Policy gradient* methods like the REINFORCE algorithm [57], on the other hand, bypass the step of learning a value function and directly attempt to learn a policy instead. These algorithms represent the policy in a parameterized form, and based on the agent's interactions with the environment, update these parameters in the direction that increases the probability of beneficial actions and reduces the probability of harmful actions. *Actor-critic* methods [52] combine the two approaches and simultaneously learn both a value function and a policy, using one to inform and aid the learning of the other. We refer the interested reader to Sutton and Barto's book [52] for a more complete technical treatment of these algorithms.

2.2.1 Deep Reinforcement Learning

In most practical applications of reinforcement learning, the state space of the problem is too large to admit the use of an explicit representation for the value function and/or the policy. For example, in the

RDW setting, one could attempt to build a look-up table that, for every position (x,z) the walker could occupy, prescribes an action to take. Indeed, the theoretical soundness of algorithms like Q-learning rely on the tractability of such a construction. However, such a table would be infeasible to store in practice. The standard workaround is to use an implicit functional representation instead. For example, rather than a table, one could represent the Q-function in a parameterized form such as $\hat{Q}(x,z,a) = w_1\, x + w_2\, z + w_3\, a$, where w_1, w_2, and w_3 represent scalar weights that can be tuned using standard linear regression. More generally, one may use any off-the-shelf *function approximator* like decision trees, support vector machines or neural networks, to represent the value function or policy, and fit it using standard techniques from supervised machine learning [6].

While the introduction of function approximators makes RL tractable, it also makes the training process brittle. For decades, this was a limiting factor in practical RL, and indeed, machine learning more broadly. Success in a particular application domain often relied on human insight and the careful crafting of *features* — abstractions that succinctly captured the information necessary for effective decision-making in a given state, while paring away extraneous details [11]. For example, when using RL to learn how to play the game of PacMan, one might design a function approximator that took as inputs the distance to the nearest ghost and the distance to the nearest food pellet as inputs, rather than the raw pixel information from the screen. While this approach may be successful in settings where the human designer is already a domain expert, results can be mixed in more challenging problems. Recent advances in *deep learning*, a powerful tool for representation learning, have helped obviate the need for such feature engineering. Deep learning methods are able to effectively train large artificial neural networks, a class of statistical models that loosely imitate the structure and behavior of biological neurons in the brain. These networks are able to represent complex nonlinear functions and discover compact, abstract representations of high-dimensional data [14, 26]. Deep neural networks have enjoyed a meteoric rise in popularity in the last decade due to their success in countless domains, including image classification [24], language translation [51], and autonomous driving [9].

Deep reinforcement learning refers to the use of deep neural networks as function approximators in RL problems. This approach has enjoyed several high-profile successes recently, enabling machines to surpass human performance in classic video games [31], defeat the world champion in the combinatorially daunting game of Go [44], achieve superhuman performance in multiplayer poker [7], and approach human-level dexterity for object manipulation [3]. In all these settings, a system was able to learn an effective control policy *tabula rasa* and with minimal human input, simply through experimentation in an environment with unknown dynamics and reward structure.

2.2.2 Proximal Policy Optimization

Proximal Policy Optimization (PPO) is an actor-critic method for deep reinforcement learning [42]. The algorithm uses deep neural networks to represent both the value function and the policy. The learning agent collects *experiences* — tuples of state-action-reward — by interacting with the environment and uses these to update the network parameters with gradient-based optimization methods. A key challenge when using deep neural networks as function approximators in RL is instability: even modest parameter updates can often cause very dramatic changes to the policy and cause the learning to collapse. PPO addresses this problem by using a novel clipping term in the objective function that ensures that network updates seldom move the new policy far from the previous policy [42]. PPO also employs a second corrective in the form of generalized advantage estimation [41]: rather than directly estimate $Q(s,a)$, the learned value network instead estimates $A(s,a)$, the *advantage* associated with an action a in state s. Intuitively, this is a lower-variance estimate of $Q(s,a)$ that seeks to uncover actions that have unusually high payoffs in comparison to a random action in a given state. PPO is state-of-the-art and has been shown to outperform many other methods across a variety of high-dimensional continuous control problems [42].

3 Steer by Reinforcement Learning (SRL)

We aim to produce a system that, without restricting users to a predetermined path, can redirect them in a VE such that the virtual distance traveled between hard resets is maximized. We provide a brief overview of our approach. First, a steering agent is trained in an environment that simulates a user walking throughout a VE while in a 5.79 meter $\times$ 5.79 meter physical tracking space (to mimic the tracked space of a single HTC Vive Pro), with no obstacles. At discrete intervals of time, the agent receives the user's physical position and orientation, which it then uses to choose the redirection gains that will be applied. Based on feedback from these decisions, the agent's policy is updated. The agent is continually trained in this manner until convergence, at which point the parameters of the deep neural network that encodes the learned policy are frozen. This network may then be used to make steering decisions in new scenarios, i.e., to output the rotation, translation and curvature gains to apply, as a user continues to move around the VE. In the remainder of this section, we describe the details of the RL environment and the training of the SRL agent.

3.1 Environment

Our RL environment simulates a user moving throughout a 5.79 meter $\times$ 5.79 meter physical space while following an unbounded virtual path. The training phase comprises *episodes*, each of which begins with the walker positioned at a randomly chosen coordinate along one of the boundaries of the tracking area, with a random orientation that faces away from this boundary. A virtual path is randomly generated for the user, with the walk terminating when the user collides with a boundary (for example, see Figure 2). One user walk is simulated each episode. These simulated paths use a constant velocity of 1.4m/s, the average human walking speed, and are generated by following a straight line with changes in direction randomly occurring every 0.5 to 3.5 meters. When a change in direction occurs, the turn radius is chosen as a random value between 4 and 8 meters, and the adjustment to the heading (in degrees) is randomly sampled from the normal distribution with zero mean and scale 22.5. On every frame, there is a 1 in 30 chance that the user chooses a new direction to look (separate from the direction of travel), assuming they are not already turning their head. The new looking angle is randomly sampled from a normal distribution with zero mean and scale 45, where zero degrees corresponds to looking in the direction of travel. The user's head turns at a constant rate of 9 degrees per second. While many of these random walk parameters are arbitrary, we note that the specific choices are moot, since the learning system can simply be retrained for new parameter choices as necessary.

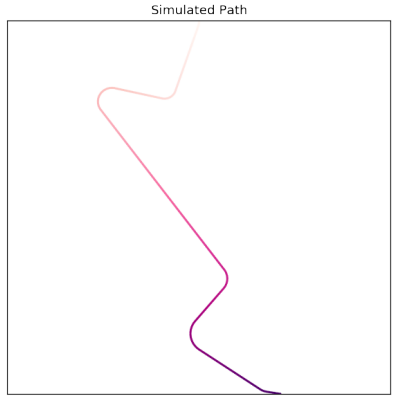

Fig. 2. A simulated path with no redirection applied. This is an example of a path the agent would see during training. The darkening tint on the path represents the passage of time.

The instantaneous state of the user at time t is encoded by the 4-vector

$$s_t = \left[x_t, z_t, \phi_t^H, \phi_t^L\right]$$

Hyperparameter	Value	Description
discount factor	0.99	Discount factor (γ) used in computing state values. This value corresponds to a time horizon of 100 actions into the future.
batch size	2048	The number of experiences collected for each gradient descent update.
learning rate	0.001	The learning rate used by the Adam optimization algorithm.
clipping range	0.2	The clipping parameter epsilon used in the PPO surrogate objective.
λ	0.95	The parameter in the Generalized Advantage Estimation [41] calculation that controls the tradeoff between bias and variance.
entropy coefficient	0	The coefficient for the entropy term in the objective function that encourages exploration. We found that removing this exploration bonus slightly stabilized learning.

Table 1. The hyperparameter values used when training our final agents.

where x_t and z_t are the user's coordinates in the tracking space and ϕ_t^H and ϕ_t^L are the user's heading angle and looking angle respectively. All four values are normalized to the range $[0,1]$. The 10 most recent states observed by the agent are concatenated to form the 40-vector

$$o_t = \left[x_{t-9}, z_{t-9}, \phi_{t-9}^H, \phi_{t-9}^L, x_{t-8}, z_{t-8}, \phi_{t-8}^H, \phi_{t-8}^L, \ldots, x_t, z_t, \phi_t^H, \phi_t^L\right]$$

which the agent receives as an *observation* at time t. Note that due to our use of action repetition (discussed below), the 10 states in an observation may not be contiguous in time — rather, they represent the last 10 states when the agent was called upon to make a steering decision. The policy learned by the agent maps each such observation to an action. Such state concatenation (also referred to as "frame stacking") is a technique that is commonly employed in deep RL in domains that require temporal awareness [31]. In our setting, we use frame stacking to provide the agent with information about the user's recent path, and therefore, potential future trajectory, which is not inferable from the instantaneous state alone.

Upon receiving an observation, the agent selects an action to execute. An action is a 3-vector

$$a_t = \left[g_t^R, g_t^T, g_t^C\right]$$

that specifies the rotation, translation, and curvature gains that will be applied to the user's movement at time t. The range of permissible values for the gains are the imperceptible gain values as determined by Steinicke et al. [47]: $g_t^R \in [-0.2, 0.49]$, $g_t^T \in [-0.14, 0.26]$ and $g_t^C \in [-0.045, 0.045]$. Once an action is selected, the environment is stepped forward to time $t+1$, by advancing the walker to their next position in the virtual path, with the gains given by a_t applied to the user's movement in the physical world.

However, the agent is not called upon to make decisions on every time step t due to our use of *action repetition* — when the agent sees an observation and selects an action, that action is applied for k consecutive time steps in the environment, before the agent is shown another observation and allowed to act again. During training, k is uniformly sampled from the range $[15, 25]$ every time an action is selected. Hence, any two consecutive states in the frame stack will be between 15 and 25 atomic time steps apart. The action repeat technique is standard in RL [31], and is known to improve learning by helping the agent learn a smooth policy with minimal jitter [43].

After each action is taken, the agent receives a reward signal from the environment. The structure of this reward often has a significant impact on the success of the learning agent as a poorly designed reward function can bias an agent's policy in unexpected and counter-intuitive ways [2]. Reward functions generally fall into one of two categories, *sparse* or *shaped*. Sparse reward functions signal the agent very infrequently, often only when a primary objective has been achieved (for example, send a reward of -1 when the user leaves the tracking space and a reward of 0 otherwise). Sparse rewards are generally desirable because they are easily specified and impose minimal bias on the learned policy. Unfortunately, sparse rewards are notoriously difficult to learn from due to the *credit-assignment problem* [29] — if a reward is only received after a long sequence of actions, it can be challenging to determine which of those actions meaningfully contributed to the eventual outcome. On the other hand, shaped reward functions send a signal at every time step (for example, the user's distance from the nearest boundary of the tracking space). Shaped rewards require the human to encode more information about the problem's desired solution and are liable to inject those biases into the policy. However, they are easier to learn from and can lead to faster convergence and better policies in practice [25, 33].

In our experiments, we use a shaped reward function that attempts to prevent collisions by incentivizing scenarios in which there is a lot of free space in the directions the user is likely about to walk. For example, a state where the user is standing at the middle of a boundary and is moving towards the center of the tracked space would receive a high reward. Formally, at time t, the agent receives the reward

$$R\left(x_t, z_t, \phi_t^H\right) = \hat{R}\left(\phi_t^H\right) + \min\left[\hat{R}\left(\phi_t^H - 90\right), \hat{R}\left(\phi_t^H + 90\right)\right],$$

where x_t and z_t are the user's physical x- and z-coordinates on the ground plane, and $\hat{R}(\phi)$ is the distance between the agent and the boundary in the direction given by angle ϕ. The min-term encourages states where the user has space directly to their left and right. The values are also normalized to the size of the room, i.e., distances are divided by the length of the diagonal of the tracking space. An exception is made when the user collides with the boundary of the tracking space, in which case $R\left(x_t, z_t, \phi_t^H\right) = -1$. $R\left(x_t, z_t, \phi_t^H\right)$ is calculated for all k timesteps in the action repeat, and the sum of all k rewards is returned to the agent as the reward signal for that repeat sequence.

The environment is run at a frame rate of 60 fps, and each time step represents one frame. An episode terminates when the user reaches the boundary of the physical space or after the simulation has been advanced 3600 time steps.

3.2 Training

We train an RL agent to learn a policy that prescribes redirection gains given the user's position and orientation in the tracking area. Our agent is trained with the PPO algorithm [42] on the environment described in Section 3.1. We use the OpenAI Baselines [10] implementation of PPO in our experiments, and the agent is trained on 8 vCPUs and a single NVIDIA K620 GPU. We experimentally tuned the various hyperparameters used in the PPO algorithm, and the best performing settings are listed in Table 1.

We represent the policy and value functions with two fully-connected artificial neural networks. Each network has two hidden layers of 128 units each, and each hidden layer is followed by a layer of rectified linear units (ReLU) [13]. While these are relatively small networks in the context of modern deep learning systems, they compare favorably with network sizes that are typically employed in deep RL [42]. Further, the relatively small size allows for short computation times during action selection, which is an important consideration when redirecting users in real-time.

The value network outputs a single scalar corresponding to the value of a state s, which is used to compute the advantage term $A(s,a)$ for the action a. The policy network outputs three pairs of scalars, corresponding to the mean and standard deviation of three Gaussian distributions. The gains $a_t = [g_t^R, g_t^T, g_t^C]$ to apply are determined by sampling independently from these three distributions. The network

parameters are updated with the Adam optimization algorithm [23]. Each agent was allowed to make 1.5 million steering decisions during training, by which point performance had typically plateaued. This is equivalent to about 140 hours of real time spent redirecting users.

4 METHODS

In order to assess the performance of our system, we evaluate and compare three unique steering algorithms:

- **NO**: No redirection is applied. This is the baseline algorithm for comparison.
- **S2C**: The steer-to-center algorithm as described by [19].
- **SRL**: Gains are applied according to the policy learned by the agent.

4.1 Simulated Paths

We evaluate the algorithms on a set of 5000 simulated paths. The paths are generated with the same method used during training (see subsection 3.1), with the exception that each path begins with the user facing directly north, south, east, or west, and the user is initially placed in the center of the appropriate wall (for example, the south wall if initially facing north).

4.2 Real Paths

In order to test the effectiveness of our agent's learned policy in a more realistic setting, we recorded the paths walked by 10 volunteers through an immersive VE. The goal was to collect a set of virtual paths that were not generated by our simulator to determine if the policy learned on the simulated paths was effective when applied to real human walking behaviors. We note that *transfer learning* — where the agent is expected to perform well on a task after training on a related, but different, task — is still an open-problem in machine learning [36]. However, given the time-consuming nature of collecting a large enough sample of real user paths for reliable learning, we chose to instead investigate the viability of a steering algorithm trained on simulated data.

The VE used was a virtual replica of the historic Johnson house in New Canaan, Connecticut (see Figure 3). Participants were initially positioned at an entrance to the house and instructed to simply explore the environment for 60 seconds. As the participant moved throughout the environment, their virtual position, heading, and looking direction were recorded at every frame, and were subsequently used for evaluating our steering algorithms.

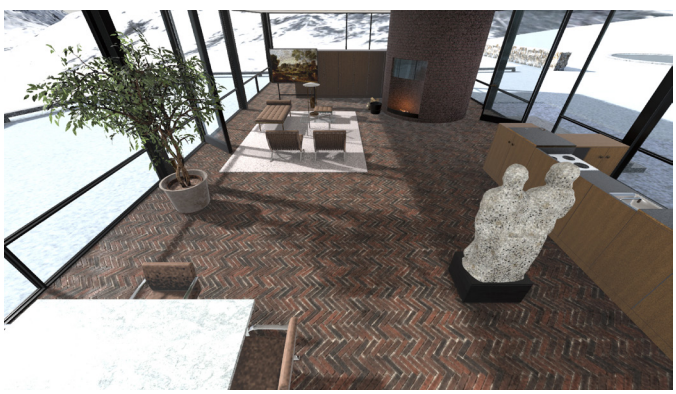

Fig. 3. The VE that participants interacted with when we collected real user paths. Participants were placed inside the house and simply instructed to explore. The house is a replica of the Phillip Johnson Glass House and is 17 meters $\times$ 9.8 meters.

The VE is rendered in the Unity3D engine. Participants use an HTC Vive Pro to enter the VE, and a Vive Pro Puck worn on the waist captures the user's heading direction. The physical tracking space is 3.5 meters $\times$ 3.5 meters. When the user collides with the boundary of this tracking space, they are reset using the method described by Williams et al [56]. This allowed participants to explore the entire VE even though it was more than four times the size of the tracked space. Path data was not recorded while the user was being reset.

condition	mean	sd	median	min	max
NO	6.09	1.37	6.23	2.78	9.68
S2C	8.69	3.31	8.87	1.96	18.32
SRL	8.96	3.19	9.12	1.90	17.61

Table 2. Descriptive statistics of distance traveled (*meters*) before running into a wall of the NO, S2C, and RL steering algorithms on 5000 simulated paths. Statistics include the mean, standard deviation (sd), median, minimum distance traveled (min), and maximum distance travelled (max).

4.3 Evaluation

We applied each steering algorithm to the 5,000 simulated paths and 10 real paths, and measured performance by recording the virtual distance traveled before a collision with a boundary of the tracking space. During evaluation, the SRL agent used a constant action repeat of 20, rather than a random repeat amount, which allows for deterministic evaluation.

It is well known that changing random seeds between trials can have a significant impact on the performance of deep RL algorithms [15]. Thus, when evaluating RL algorithms, it is common practice to train multiple instances of an agent and average the performance across all trials. We follow this practice by training 10 identical agents (the only difference is the random seed used for each run), each of which is evaluated on the 5,000 simulated paths and 10 real paths. For each path, we then average the virtual distance traveled across all 10 agents. The averaged results are used in our analyses.

5 RESULTS

5.1 Simulated Paths

We calculated the virtual distance traveled by the user before colliding with a boundary on 5,000 simulated paths, using each of the three steering algorithms, NO, S2C, and SRL. Before analysis, we replaced the outliers for each steering algorithm, determined to be more than 1.5$\times$ the inter-quartile range, by each algorithm's median value. Descriptive statics for the three algorithms can be found in Table 2. Due to heavy-tailed distributions in SRL and S2C, Levene's test for homogeneity of variance failed, $F(2,14997) = 1048.7$, $p < .0001$. The difference in virtual distance traveled by the steering algorithms was therefore analyzed using a robust one-way repeated measures ANOVA with 20% trimmed-means. Analysis was performed with R version 3.6.1 using the WRS2 package for robust statistical methods.

A significant effect of steering algorithm was found, $F(2,5984.37) = 2870.90$, $p < .0001$. Due to the large sample size, we report our effect sizes through confidence intervals [28]. A confidence interval that does not include zero is considered significant, however interpreting the results requires considering the bounds of the confidence interval range. We performed a pairwise robust post hoc analysis using linear contrasts (see Table 3 and Figure 4). As expected, S2C significantly outperformed NO ($\hat{\psi} = -2.35$ $[-2.45, -2.25]$). On 95% of trials, paths using the S2C algorithm traveled 2.25 to 2.45 meters farther, before running into a boundary, than when traveling the same path using no redirection. Importantly, our SRL algorithm similarly outperformed NO, ($\hat{\psi} = -2.60$ $[-2.70, -2.50]$). Finally, SRL slightly outperformed S2C ($\hat{\psi} = .10$ $[\,0.04,\ 0.16]$). On 95% of simulated paths, the SRL algorithm allowed the walker to travel an additional 0.04 to 0.16 meters, before running into a boundary, than when traveling the same path using S2C.

5.2 Real Paths

We analyzed the virtual distance traveled on paths walked by real people using each of the steering algorithms, to study the effectiveness of our SRL algorithm, even though it was trained on simulated paths. The

Contrast	$\hat{\psi}$	CI	p
NO vs. SRL	-2.60	$[-2.70, -2.50]$	$< .001$
NO vs. S2C	-2.35	$[-2.45, -2.25]$	$< .001$
SRL vs. S2C	0.10	$[0.04, 0.16]$	$< .001$

Table 3. Pairwise robust post hoc analysis of the NO, S2C, and SRL steering algorithms on 5000 simulated paths. S2C outperformed NO, and SRL outperformed both NO and S2C.

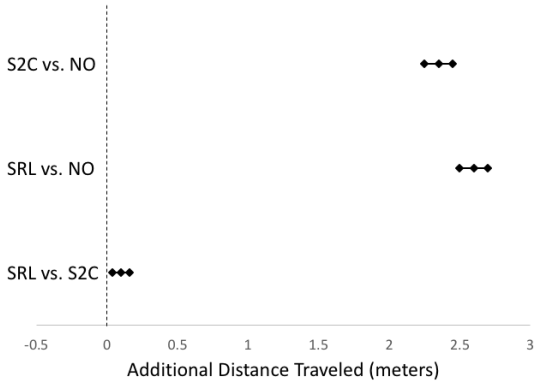

Fig. 4. The 95% confidence intervals (CIs) of the additional distance traveled comparing each of the steering algorithms pairwise. SRL significantly outperformed both S2C and NO (95% CI does not include zero). S2C significantly outperformed NO.

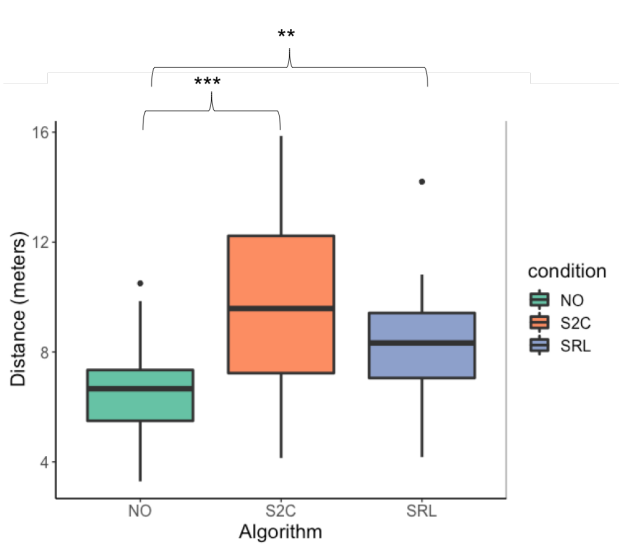

Fig. 5. Boxplots for each of the three steering algorithms of the virtual distance traveled in 10 real paths. SRL and S2C significantly outperformed NO. *** $p < .001$, ** $p < .01$

assumption of homogeneity of variance was violated, $F(2,87) = 7.22$, $p = .001$. The difference in virtual distance traveled by the steering algorithms was therefore analyzed using a robust one-way repeated measures ANOVA with 20% trimmed-means. A significant effect of steering-algorithm was found, $F(1.44, 24.55) = 17.27$, $p < .001$. Pairwise robust post hoc analysis was performed. The NO steering algorithm was significantly outperformed by both S2C ($\hat{\psi} = -2.61$ $[-3.98, -1.23]$, $p = .0001$) and SRL ($\hat{\psi} = -1.52$ $[-2.44, -.59]$, $p = .0004$). No significant difference was found between S2C and SRL, ($\hat{\psi} = .78$ $[-.47, 2.04]$, $p = .12$) (see Figure 5).

6 Discussion

We found that deep RL techniques can learn a steering policy that marginally outperforms S2C on simulated paths. Additionally, we demonstrated that even when trained on simulated paths, our SRL algorithm was able to significantly outperform no redirection on real paths, a critical demonstration of transferring simulation to the real-world. No significant difference was found between SRL and S2C on real paths, further supporting SRL's transfer learning capabilities. Our results provide the critical evidence that reinforcement learning can provide additional insight into developing more effective RDW steering algorithms.

In this work, we analyzed the virtual distance traveled before reaching a boundary of the tracking space, rather than the number of hard resets in a given amount of time. For the simulated paths, translating our results into resets per minute gives the following: NO ($M = 13.64 \pm 5.21$), S2C ($M = 9.29 \pm 5.4.86$), and SRL ($M = 9.18 \pm 4.64$). Our number of resets is noticeably higher than those reported by Lee et al. [27]. Their non-predefined path in a slightly smaller tracker space had fewer resets per minute for both no redirection (8.55 resets/minute), and S2C (7.18 resets/minute). This suggests that the path type used dramatically affects the number of resets, the distance traveled, and the learned algorithm. Our simulated paths were generally long straight paths with occasional turns. Using different paths with more curves and turns would likely produce different results.

After training agents with SRL, we more closely examined a policy learned by our agent. First, we find that the agent always outputs the maximal translation gain. While this trivial policy may seem surprising at first, it makes intuitive sense when considering the agent's goal — a higher translation gain corresponds to a larger distance traveled in the virtual world, which in turn maximizes the agent's future reward. More interesting is SRL's policy for rotation gain: SRL chooses either the maximum or minimum value the vast majority of the time. Otherwise, the policy outputs in-between values with relatively uniform probability (see Figure 6). A similar pattern emerges from S2C, with the exception that S2C prescribes zero rotation/curvature with much greater frequency than SRL.

We attempt to gain some insight into the spatial patterns of SRL's rotation policy compared to S2C with the simple example described in Figure 7. The perfectly straight paths used in this setting, although unrealistic, nonetheless allow for a straightforward interpretation of the results. While SRL's policy in this example is not perfect, it does seem intuitive — rotation is maximized against head rotation (minimum negative value) while the user still has space in front of them, whereas it is maximized with head rotation (maximum positive value) when a boundary is approaching to help turn the user around towards open space. The heatmap for SRL's curvature gains (omitted) looks almost identical to the one shown in Figure 7, except that the red and blue regions are flipped. This is surprising, as it implies that the rotation and curvature gain are working against each other. However, when we evaluated the best performing agent on the simulated paths but enforced a constant curvature gain of zero (the agent only selected rotation and translation gains), the median virtual distance traveled increased by 0.02 meters compared to when the agent was allowed to determine all three gains. Given the relatively low impact of applying curvature gains, it is unsurprising that the agent would have difficulty in learning a meaningful curvature policy.

The main difference between S2C and SRL appears to be that SRL always redirects the user, while S2C does not redirect the user when facing the center of the tracked space. When walking directly north through the center of the tracked space, S2C performs no rotation until the user passes through the center. S2C then attempts to arch the user back toward the center of the room.

An important advantage of an RL system is that it can be trained to fit specific types of environments or paths. We chose to train the agent on random virtual paths in a generic square tracking space, but this was merely for the purpose of producing a general-purpose algorithm that is applicable to typical environments. Consider a VE in which

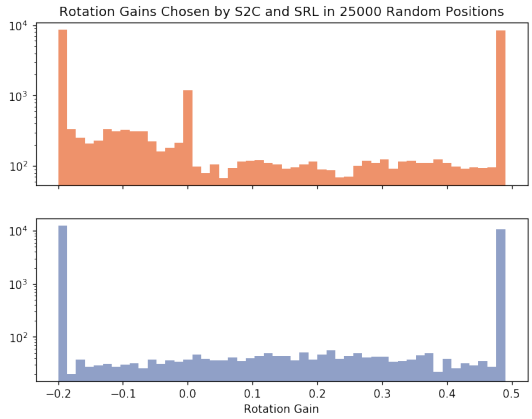

Fig. 6. S2C and SRL determined the rotation gain that should be applied for 25000 random positions from generated paths. The histograms show the distributions of the gains selected by S2C (orange) and SRL (blue). Note the logarithmic scale on the y-axis. We find that a similar pattern emerges for the curvature gain.

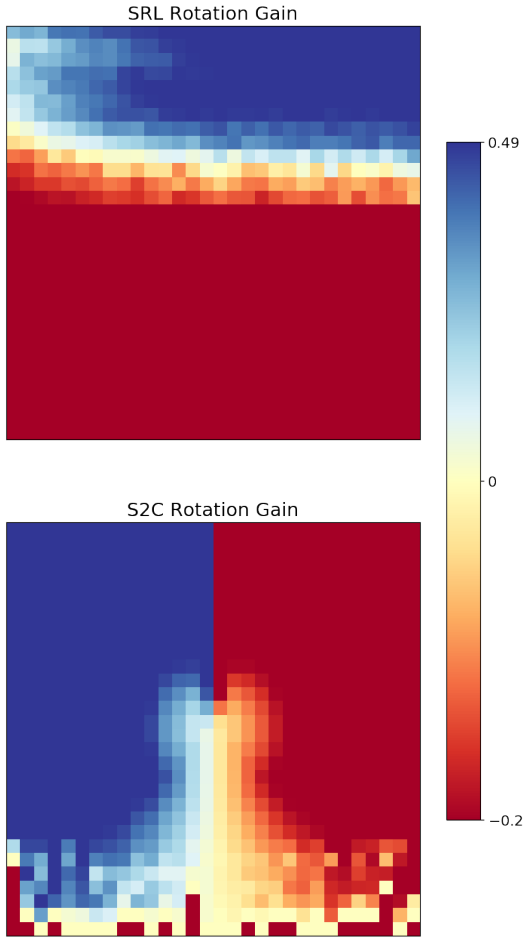

Fig. 7. Each heatmap represents the physical tracking space, and the color corresponds to the average rotation gain chosen at that location in the space. The gains are collected from simulated paths that start at a random position on the south wall and move north in a straight line (the gains are not actually applied to the paths, so they are straight in the physical world as well). The user's head is unrealistically simulated to continuously move clockwise.

humans are likely to follow predictable paths, such as navigating an office. As long as virtual paths can be simulated that are representative of how users will behave in that environment (or real paths from that environment can be captured), an agent can be trained specifically for that VE. In this scenario, we can think of the agent as learning an implicit predictive algorithm — the agent has a chance to extract meaningful patterns from users' interactions with the environment and make informed decisions based on forecasts of where they are likely to go. Indeed, neural networks have demonstrated the ability to predict human walking paths in the computer vision domain [20]. Similarly, an agent could be tailored to an unusually shaped tracking area.

Once an agent has been trained, SRL is very straightforward to use in practice. The neural network that represents the policy can be extracted from the training infrastructure to serve as an efficient lookup table. With the user's recent history of states given as input, a forward computation is performed through the network to produce the three gains that should be applied. On our modest hardware, the average time for this computation is very fast, about 0.407 milliseconds.

6.1 Other Experiments

Despite the excitement surrounding it, deep RL is still a relatively nascent research area and the reliable reproducibility of results remains an ongoing challenge for the field [15]. It is not uncommon for algorithms to excel at learning policies in one domain, but completely fail in another; or to display an extreme sensitivity to hyperparameter settings [15]. In this section, we briefly discuss some experimental directions we pursued that ultimately proved unfruitful, in the hope that it may be of use to future researchers.

As discussed in subsection 3.1, a sparse reward function is preferable for RL problems as they allow the designer to succinctly specify the agent's goal, with minimal domain-specific knowledge engineering. We performed several experiments with sparse reward structures — for example, with the agent receiving a -1 signal upon reaching a boundary, and a score of 0 at all other steps. However, the learned policies always performed considerably worse in these cases than when using the shaped reward. Additionally, we experimented with shaped rewards that simply considered the distance to the nearest wall (median distance traveled = 8.90 meters), but we found our final reward function (median distance traveled = 9.12 meters) performed significantly better, $F(1, 5920.78) = 15.01$, $p = .0001$. See Figure 8.

Another factor we considered was network architecture. While our final agent uses a simple fully-connected neural network, we also tested long short-term memory (LSTM) networks [17]. LSTMs maintain an internal state that persists across time steps, effectively endowing a neural network with some "memory". They have proven adept at modeling long-term dependencies, particularly in natural language tasks like translation [51]. LSTMs are also used in deep RL applications to enable the policy/value networks to perform some rudimentary temporal reasoning [30]. We found that adding LSTM layers did not result in any significant performance improvement in our setting. We suspect that our state stacking technique (described in subsection 3.1) sufficiently models temporal effects in our domain, and the LSTM machinery is thus superfluous. Additionally, we found that increasing the number of hidden units or layers in the fully-connected architecture had no impact on performance.

Finally, we also studied the impact of discretizing the action space of the agent. Specifically, rather than allowing the agent to pick any real number in the permitted range for each of the gains, we limited the agent to a small set of quantized values in each range. In this discretized setting, we were able to run learning experiments using a second deep RL algorithm, Rainbow [16], in addition to PPO. Rainbow and its predecessors have been more widely studied in the research literature and are somewhat better understood and easier to tune — however, they cannot handle continuous control problems, unlike PPO which can handle both continuous and discrete action spaces. However, policies learned in this discretized setting (with both algorithms) consistently performed worse than those learned using PPO in the continuous setting.

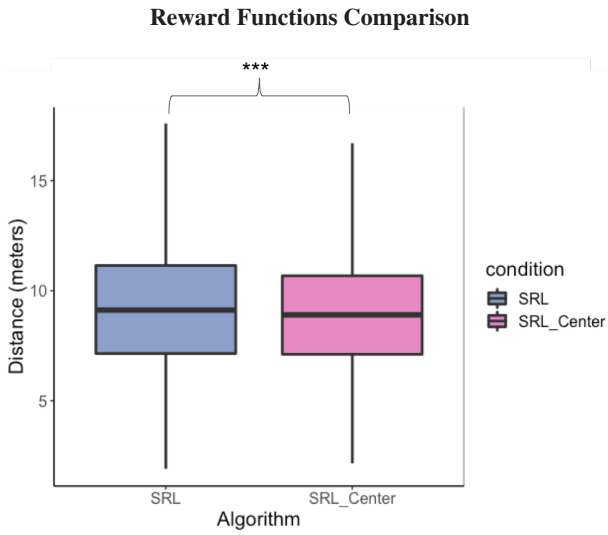

Fig. 8. Boxplots of the number of steps taken in 5000 simulated paths for SRL with two different reward functions, SRL with the reward function defined in section 3.2.1, and SRL_Center with the reward function defined based on the distance to the nearest wall of the tracked space (outliers removed). SRL significantly outperformed SRL_Center. *** $p < .001$

6.2 Future Work

While our results in this paper demonstrate that RL techniques can be successfully used to learn steering algorithms for RDW, there are several promising avenues for future work that hold the potential to advance the state-of-the-art.

We found the choice of reward function to have a significant impact on performance (see subsection 6.1). Given the difficulties in learning from sparse rewards, many successes in RL arise from carefully crafted, and often complex, shaped reward functions, and it is likely that modifications to our reward function could lead to improved performance. An interesting approach could be to combine unsupervised auxiliary tasks with SRL — a technique that attempts to simultaneously maximize pseudo-reward functions in the absence of extrinsic rewards [22]. Ultimately though, sparse rewards should be revisited for this problem, as using a shaped reward imposes biases on the learned policy.

Given the promise of RL for RDW, we also note the need for a dataset of paths walked by real humans. While this paper considered the transfer learning problem of training on simulated paths to steer human walkers, it is well-known that machine learning methods work best when they are trained on data drawn from the same distribution as the evaluation data [1]. Our results on the 10 real user paths reflect this. Thus, future work should focus on either creating a more realistic path simulator or, preferably, collecting a large dataset of real paths. If our agent were to be trained on sufficient real data, we would expect to see a large boost in performance on real user paths.

Finally, as is the case with all methods that rely on local optimization techniques, deep RL methods can get stuck in local optima from which recovery can be challenging. One promising workaround is to instead start with a high-quality policy and simply attempt to "fine-tune" it using RL. The first version of Google DeepMind's AlphaGo system, for playing the board game Go, used this approach [45]. The system first learned a baseline value function by training on a database of Go games played by humans, using standard supervised learning methods. RL was subsequently used to iteratively improve the system's positional evaluation ability, to a superhuman level. One could attempt something similar in the RDW setting: we could train a policy network that simply mimicked the prescriptions of S2C in any given scenario, using actual S2C gains as the supervising signal. RL could then subsequently be used to improve on the S2C steering policy.

7 Conclusions

In this work, we presented an approach for learning a RDW steering algorithm with RL. We frame the redirection problem as a Markov Decision Process where at each time step, a learning agent receives an observation that encodes the user's position and orientation in the tracked space and takes an action that determines the rotation, translation, and curvature gains to apply. When trained and evaluated on simulated user paths, our algorithm, SRL, significantly increased the virtual distance traveled between collisions with the boundaries of a 5.79×5.79 meter tracking space compared to steer-to-center. When evaluated on 10 real user paths, we find that the policy learned on the simulated paths still significantly outperforms no redirection and has no significant difference from S2C — evidence of successful transfer learning.

Because we model the problem as a continuous control problem where the agent directly outputs redirection gains, SRL is a significant step in the direction of a steering algorithm that is not reliant on extensive human engineered logic. This is the first work to apply RL to RDW in such a manner, to the best of our knowledge. Under the general framework we present here, we see enormous potential for RL to drive the future of RDW and ultimately allow for the free exploration of unbounded virtual environments in limited physical spaces.

Acknowledgments

The authors wish to thank the Davidson Research Initiative for supporting this work.

References

[1] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning From Data*. AMLBook, 2012.

[2] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, and D. Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

[3] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *arXiv preprint arXiv:1808.00177*, 2018.

[4] M. Azmandian, T. Grechkin, M. Bolas, and E. Suma. Physical space requirements for redirected walking: how size and shape affect performance. In *Proceedings of the 25th International Conference on Artificial Reality and Telexistence and 20th Eurographics Symposium on Virtual Environments*, pp. 93–100. Eurographics Association, 2015.

[5] M. Azmandian, R. Yahata, M. Bolas, and E. Suma. An enhanced steering algorithm for redirected walking in virtual environments. In *Virtual Reality (VR), 2014 iEEE*, pp. 65–66. IEEE, 2014.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[7] N. Brown and T. Sandholm. Superhuman ai for multiplayer poker. *Science*, p. eaay2400, 2019.

[8] Y. Chang, K. Matsumoto, T. Narumi, T. Tanikawa, and M. Hirose. Redirection controller using reinforcement learning. *arXiv preprint arXiv:1909.09505*, 2019.

[9] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2722–2730, 2015.

[10] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. Openai baselines. `https://github.com/openai/baselines`, 2017.

[11] P. Domingos. A few useful things to know about machine learning. *Commun. ACM*, 55(10):78–87, Oct. 2012. doi: 10.1145/2347736.2347755

[12] S. Freitag, D. Rausch, and T. Kuhlen. Reorientation in virtual environments using interactive portals. In *3D User Interfaces (3DUI), 2014 IEEE Symposium on*, pp. 119–122. IEEE, 2014.

[13] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. Gordon, D. Dunson, and M. Dudk, eds., *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15 of *Proceedings of Machine Learning Research*, pp. 315–323. PMLR, Fort Lauderdale, FL, USA, 11–13 Apr 2011.

[14] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[15] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep reinforcement learning that matters. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[16] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[17] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

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

[20] S. Huang, X. Li, Z. Zhang, Z. He, F. Wu, W. Liu, J. Tang, and Y. Zhuang. Deep learning driven visual path prediction from a single image. *IEEE Transactions on Image Processing*, 25(12):5892–5904, 2016.

[21] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castaneda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

[22] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

[23] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pp. 1097–1105. Curran Associates Inc., USA, 2012.

[25] G. Lample and D. S. Chaplot. Playing fps games with deep reinforcement learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[26] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[27] D.-Y. Lee, Y.-H. Cho, and I.-K. Lee. Real-time optimal planning for redirected walking using deep q-learning. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 63–71. IEEE, 2019.

[28] M. Lin, H. C. Lucas Jr, and G. Shmueli. Research commentarytoo big to fail: large samples and the p-value problem. *Information Systems Research*, 24(4):906–917, 2013.

[29] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[30] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[31] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[32] T. Nescher, Y.-Y. Huang, and A. Kunz. Planning redirection techniques for optimal free walking experience using model predictive control. In *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, pp. 111–118. IEEE, 2014.

[33] A. Y. Ng. *Shaping and policy search in reinforcement learning*. PhD thesis, University of California, Berkeley, 2003.

[34] N. C. Nilsson, T. Peck, G. Bruder, E. Hodgson, S. Serafin, M. Whitton, F. Steinicke, and E. S. Rosenberg. 15 years of research on redirected walking in immersive virtual environments. *IEEE computer graphics and applications*, 38(2):44–56, 2018.

[35] OpenAI. Openai five. `https://blog.openai.com/openai-five/`, 2018.

[36] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, Oct. 2010. doi: 10.1109/TKDE.2009.191

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

[38] M. L. Puterman. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[39] S. Razzaque. *Redirected walking*. University of North Carolina at Chapel Hill, 2005.

[40] S. Razzaque, Z. Kohn, and M. C. Whitton. Redirected walking. In *Proceedings of EUROGRAPHICS*, vol. 9, pp. 105–106. Citeseer, 2001.

[41] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[42] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[43] S. Sharma, A. S. Lakshminarayanan, and B. Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

[44] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, Jan. 2016. doi: 10.1038/nature16961

[45] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[46] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

[48] F. Steinicke, G. Bruder, T. Ropinski, and K. Hinrichs. Moving towards generally applicable redirected walking. In *Proceedings of the Virtual Reality International Conference (VRIC)*, pp. 15–24. IEEE Press, 2008.

[49] 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 2011 IEEE Virtual Reality Conference*, pp. 159–166. IEEE, 2011.

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

[51] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pp. 3104–3112. MIT Press, Cambridge, MA, USA, 2014.

[52] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[53] J. Thomas and E. S. Rosenberg. A general reactive algorithm for redirected walking using artificial potential functions. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 56–62. IEEE, 2019.

[54] M. Usoh, K. Arthur, M. Whitton, R. Bastos, A. Steed, M. Slater, and F. Brooks Jr. Walking > walking-in-place > flying, in virtual environments. In *Proceedings of the 26th annual conference on Computer Graphics and Interactive Techniques*, pp. 359–364. ACM Press/Addison-Wesley Publishing Co., 1999.

[55] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. doi: 10.1007/BF00992698

[56] 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*, pp. 41–48. ACM, 2007.

[57] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, May 1992. doi: 10.1007/BF00992696

[58] M. A. Zmuda, J. L. Wonser, E. R. Bachmann, and E. Hodgson. Optimizing constrained-environment redirected walking instructions using search techniques. *IEEE transactions on visualization and computer graphics*, 19(11):1872–1884, 2013.