Redirected Walking in Static and Dynamic Scenes Using Visibility Polygons

Niall L. Williams, Student Member, IEEE, Aniket Bera, Member, IEEE, and Dinesh Manocha, Fellow, IEEE

Abstract—We present a new approach for redirected walking in static and dynamic scenes that uses techniques from robot motion planning to compute the redirection gains that steer the user on collision-free paths in the physical space. Our first contribution is a mathematical framework for redirected walking using concepts from motion planning and configuration spaces. This framework highlights various geometric and perceptual constraints that tend to make collision-free redirected walking difficult. We use our framework to propose an efficient solution to the redirection problem that uses the notion of visibility polygons to compute the free space that is visible, and therefore walkable, to the user from their position within an environment. Using this representation of walkable space, we apply redirected walking to steer the user to regions of the visibility polygon in the physical environment that closely match the region that the user occupies in the visibility polygon in the virtual environment. We show that our algorithm is able to steer the user along paths that result in significantly fewer resets than existing state-of-the-art algorithms in both static and dynamic scenes. Our project website is available at https://gamma.umd.edu/vis_poly/

Index Terms—Redirected Walking, Locomotion, Alignment, Visibility Polygon, Isovist, Motion Planning

1 INTRODUCTION

Natural walking as a means to explore virtual environments (VEs) is generally preferred over other artificial locomotion interfaces such as flying [1] or teleportation since natural walking improves the user’s sense of presence and task performance in the VE. Redirected walking (RDW) [2] is a locomotion interface that affords natural walking in virtual reality by imperceptibly steering the user along physical paths that differ from their virtual counterparts. Over the years, many researchers have developed different algorithms (known as RDW controllers) that apply RDW to steer the user to avoid collisions with physical obstacles that they cannot see or detect. If a collision is imminent, the RDW system applies a “reset” to reorient the user away from the nearby obstacle. Although significant progress has been made, even the best controllers are unable to guarantee a reset-free experience in arbitrary static and dynamic environments.

Redirection controllers generally fall into one of three categories: reactive, predictive, or scripted [23]. Reactive controllers steer the user based only on information available at the current or previous frames. Predictive controllers steer the user based on the user’s predicted future movement in the VE, and scripted controllers steer the user as they walk along a pre-defined path in the VE. Reactive controllers are usually preferred since they can be deployed in arbitrary environments and often require little to no additional setup (such as environment preprocessing). However, their ability to steer the user away from obstacles is usually lower than that of predictive or scripted controllers since reactive controllers must operate using less information. Predictive controllers can outperform reactive controllers, but they rely on accurate predictions of the user’s movements. Recently, a new paradigm of redirection controllers based on the concept of alignment has emerged.

Fig. 1: A visualization of the geometric reasoning that our redirection controller performs on every frame in order to steer the user in the physical space. First, the controller computes the visibility polygon for the user’s physical and virtual locations (the regions bounded by the blue and red edges, respectively). Next, the controller computes the region of space (part of the red visibility polygon) in front of the user that the user is walking towards in the virtual environment (yellow region in the right image). By comparing the areas of the regions, our controller computes the region in the physical space (yellow region in the left image) that is most similar to the virtual region the user is heading towards. Finally, the controller applies redirected walking gains to steer the user to walk towards the highlighted region in the physical space. Black dashed arrows indicate the user’s trajectory in the environment. Our algorithm yields significantly fewer resets with physical obstacles than prior algorithms.

© 2021 IEEE. This is the author’s version of the article that has been published in IEEE Transactions on Visualization and Computer Graphics. The final version of this record is available at: 10.1109/TVCG.2021.3106432
Instead of steering the user away from physical obstacles, alignment-based controllers steer the user in an attempt to minimize the difference between the physical and virtual environments. These controllers have been shown to be effective for lowering the number of resets [48] and enabling passive haptics for increased immersion [41,33].

Much of the prior work on RDW controllers focuses on users exploring static physical and virtual environments. Some work that instead focuses on dynamic scenes mostly looks at multi-user RDW systems [2,4,11]. Chen et al. [2] proposed ideas on how redirection controllers can steer the user away from moving physical obstacles. Though there has been work on dynamic physical scenes as well as multi-user systems, there are no known solutions that work well in all dynamic environments with no assumptions on obstacles’ motions.

**Main Results:** We provide a mathematical framework for the redirected walking problem, as well as a novel approach to redirected walking in static and dynamic environments. We frame the redirection problem as optimizing the user’s path in the physical environment by applying the appropriate redirection at each frame. Our formulation makes it easy to develop effective controllers and conduct rigorous analyses to better understand a redirection controller’s behavior. Using this formulation of the redirection problem, we also introduce a RDW controller that uses a novel alignment metric based on visibility polygons. That is, given the layouts of the physical and virtual environments and the user’s location in them, our controller computes the region of walkable space visible from the user’s position in both environments. With these representations of the local space that is available to the user, our controller steers the user towards a region in the physical space that is most similar to the region they are approaching in the virtual space. We found that our controller is able to significantly reduce the number of times that the user has to reset their orientation when they come too close to physical obstacles. Our main contributions in this work include:

- A mathematical framework for the redirected walking problem based on concepts from robot motion planning. We formulate it as an optimization problem, where the aim is to transform a virtual path to an optimal physical path, subject to constraints depending on the user and the information available.
- A novel redirected walking controller that steers the user based on alignment, with heuristics derived from visibility polygons. Our algorithm achieves significantly fewer resets than the current state-of-the-art controllers.
- Comparisons of our algorithm to the state-of-the-art algorithms in both static and dynamic simulated, single-user scenes.

## 2 Prior Work and Background

Locomotion is a fundamental problem in virtual reality (VR) since the user wishes to explore a VE that is usually much larger than the physical environment (PE) they are in. Furthermore, the locations of obstacles in the PE are usually not the same as the locations of obstacles in the VE. This creates a problem when the user wishes to travel along a collision-free virtual path but doing so may force them to walk into unseen physical obstacles. Many different techniques for locomotion have been developed [16,41,33,39,44], but in this work we focus on the RDW technique developed by Razaque et al. [33]. A recent review of VR locomotion interfaces can be found in [23].

By slowly rotating or translating the VE around the user while they walk, RDW allows users to explore virtual environments while located in smaller physical environments [33]. These rotations and translations are controlled by parameters called gains. In order to remain on their intended virtual path, users will subconsciously adjust their physical path to counteract the VE movements. The three main gains are translation, rotation, and curvature gains. Translation gains translate the VE around the user while they walk, causing their physical path to be longer or shorter than their virtual path, depending on the direction of the VE translation. Rotation gains rotate the VE around the user while they turn in place, which causes their physical rotation to be larger or smaller than their virtual rotation one, depending on the direction that the VE rotates. Similarly, curvature gains rotate the VE around the user while they are walking, which causes them to veer on a physical path with a different curvature than their virtual one. The faster the VE rotates or translates, the more the user’s physical path will deviate from their virtual path, and the easier it will be to steer the user away from physical obstacles. However, it is important that the gains are not large enough that the user can perceive the rotations since this will make it harder to explore and can lead to simulator sickness [36]. Other gains such as bending gains [20] have been developed, but they are less well-understood than translation, curvature, and rotation gains so we do not consider them in this work.

### 2.1 Redirection Controllers

A redirection controller is an algorithm that applies RDW gains to steer a user along a physical path, with the intention of minimizing the number of times the user collides with a physical obstacle [29]. A controller consists of a steering component and a resetting component. As the name suggests, the steering component is responsible for applying gains to steer the user in the physical space while walking. The resetting component is responsible for initiating a “reset,” wherein the user’s virtual movements are disabled until they turn in place to reorient themselves in the physical world. Resets are initiated when the user gets too close to any physical obstacle.

Many redirection controllers operate on different assumptions and with different amounts of information available to them. Consequently, controllers have traditionally been classified as reactive, predictive, or scripted [29]. Reactive controllers steer the user according to the information available at the current frame or any prior frames. These controllers are typically designed to function in arbitrary environments, with little to no pre-processing or setup required. Examples of reactive controllers include steer to center [33], steer to orbit [17,33], steer to multiple targets [33], and controllers based on reinforcement learning [6,22,38] or artificial potential fields [4,26,42]. Predictive controllers steer the user according to the information available on the current/prior frames and a prediction of the user’s future path in the VE. These types of controllers can perform better than reactive controllers, but their performance depends on the accuracy of the path prediction. Examples of predictive controllers include those developed by Zmuda et al. [51], Nescher et al. [28], and Dong et al. [11]. Scripted controllers are controllers that steer the user as they travel along designated paths in the VE [1,49]. Scripted controllers usually result in the fewest resets, but they require the researchers to design and plan the environments carefully. As a result, these controllers cannot be generalized to arbitrary environments.

Although the RDW community has developed this taxonomy for redirection controllers, more recent controllers that involve more complex algorithms or use new types of information are not easily categorized as reactive, predictive, or scripted. Thomas et al. [41,43] and Williams et al. [48] introduced redirection controllers that leverage the concept of alignment. Alignment is the concept of comparing the physical and virtual environments according to some environment feature(s) (such as the location of a user relative to an object). In other words, alignment measures the similarity of the two environments according to these environment features. It should be noted that other researchers have used ideas similar to alignment by editing the VE to match the physical one [35] or by making assumptions about the user’s motion based on the VE [51]. The alignment-based controllers developed by Thomas et al. and Williams et al. steer the user with RDW gains, but the steering decisions are guided by the degree of similarity (alignment) of the user in the physical and virtual environments. Furthermore, some controllers make implicit assumptions about the virtual reality system and incorporate these assumptions into the steering policy. Williams et al. [48] assume that the user walks along a collision-free path in the VE, while controllers based on reinforcement learning train an algorithm that implicitly learns a model of the user’s locomotion behavior and steers the user according to this model.
2.2 Motion Planning and Visibility Polygons

In the field of robotics, motion planning is the problem of moving a robot from an initial state to a goal state through a series of valid configurations that avoid collisions with obstacles [21]. For a robot with \( n \) degrees of freedom, its configuration space (denoted \( C \)) is an \( n \)-dimensional manifold, where each point in the manifold corresponds to a configuration of the robot. The configuration space describes the set of all states that a robot can be in. In order to successfully navigate from a starting position to a goal position, the robot must find a set of configurations that takes it from the starting position to the goal position. This can be formulated as finding a continuous path of valid configurations through \( C \). Some common desirable traits of such a path are that it yields the shortest path and that the robot trajectory is smooth, without many oscillations as it travels along this path.

There is considerable work on developing motion planning algorithms for static and dynamic environments. Search-based planners discretize the state space (the set of all possible states) and employ search algorithms to find a path from the start to the goal. An example of a search-based motion planning algorithm is the A* algorithm [15]. Sampling-based planners operate by randomly sampling the configuration space in order to build a valid path. Such algorithms can usually quickly find valid solutions, but their solutions are usually not the most efficient [12]. Potential field methods use attractive and repulsive forces to guide the robot through the environment [19]. These planners are easy to implement but are susceptible to getting the robot trapped in local minima of the potential function. Planning algorithms may also use geometric representations, such as visibility graphs and cell decomposition, to reason about the environment, detect collisions, and compute collision-free paths [10]. Motion planning algorithms may also use optimization to handle dynamic obstacles and compute smooth trajectories [10][22]. Optimization-based approaches are advantageous in that they can more easily handle complex, high-dimensional state spaces. Dynamic motion planning is the problem of computing a collision-free path in an environment with moving obstacles. A popular approach to dynamic motion planning is the use of velocity obstacles to reason about collision-free paths in terms of velocity [13][15].

In our approach, we use techniques from motion planning literature to compute collision-free paths for the user in the physical space and the virtual space. In particular, we perform geometric reasoning in each of these 2D spaces by using visibility polygons. The visibility polygon for a point \( p \) is the set of all points in the plane that are visible from \( p \). The visibility polygon thus encapsulates the entire region of space that has line-of-sight visibility from the point \( p \). Depending on the layout of the space, the visibility polygon may be unbounded. If we consider a human (or robot) observer located at position \( p \), the visibility polygon can be thought of as the entire region of space that the observer can see. The visibility polygon can be used to compute the free space corresponding to a point robot in an environment with polygonal obstacles [14]. Thus, the visibility polygon provides a well-defined region that we can use to compute a local path in the environment to move the user closer to their goal without any collisions.

Since human locomotion is largely dominated by the information that is immediately available to the person [27], we use the visibility polygon to perform geometric reasoning in a user’s local surroundings. In the architectural design literature, researchers have used visibility polygons (which they refer to as “isovists”) to describe an environment and then studied how people’s locomotion patterns change as the environment structure changes [5]. Wiener et al. [7] showed that the complexity of the visibility polygon (characterized by its jaggedness) was correlated with an observer’s task performance and locomotion speed in the environment. Christensen et al. [8] introduced occlusion maps, which are maps that define the regions visible over multiple points in an environment, to study how the layout of an environment changes as the observer moves to a new position. Within the RDW community, Zank et al. [50] used visibility polygons to predict the areas of the VE that the user might walk towards next.

3 Redirected Walking Using Visibility Polygons

To aid in the exposition of our redirection controller, we begin with formal definitions and a mathematical formulation of the RDW problem. A formal characterization of the RDW problem provides us with a framework that we can use to conduct more rigorous analyses of and reasoning about RDW steering algorithms. Note that the definitions we use are adapted from those used in the robot motion planning literature [21].

3.1 Definitions and Notation

In virtual reality, the user is simultaneously located in a PE, \( E_{phys} \), and a VE, \( E_{virt} \). For each environment, the user state describes their location \( p \) and heading direction \( \theta \) in the environment. We denote the user’s state as the pair \( q = \{ p, \theta \} \). The user’s state at a particular time \( t \) is denoted by \( q^t = \{ p^t, \theta^t \} \). Thus, the user’s state at time \( t \) in \( E_{phys} \) is denoted by \( q^t_{phys} = \{ p^t_{phys}, \theta^t_{phys} \} \). The path that a user walks along in an environment is represented by an ordered set of states \( Q = \{ q^0, \ldots, q^n \} \). For brevity, we denote the physical and virtual paths as \( path_{phys} = \{ q^0_{phys}, \ldots, q^n_{phys} \} \) and \( path_{virt} = \{ q^0_{virt}, \ldots, q^n_{virt} \} \), respectively.

The user’s state in an environment is also referred to as their configuration. The configuration space \( C \) (or \( C \)-space for short) is the set of all possible configurations of the user. Some configurations in the \( C \)-space correspond to the user colliding with an obstacle in the environment. The obstacles in an environment occupy the obstacle region, \( O \subset E \). The set of all colliding configurations \( q \) is the obstacle space \( C_{obs} \):

\[
C_{obs} = \{ q \in C | q \cap O \neq \emptyset \}.
\]

That is, the obstacle space \( C_{obs} \) is the set of all configurations \( q \) for which the user intersects with an obstacle in the obstacle region \( O \). The free space \( C_{free} \) is all of the other configurations that are not in the obstacle region. We define the free space as \( C_{free} = C \setminus C_{obs} \). The user’s configuration in both environments \( q^t_{phys} \) and \( q^t_{virt} \), the user’s state at a particular time \( t \) as the pair \( q^t = \{ p^t, \theta^t \} \). The user’s state at a particular time \( t \) is denoted by \( q^t = \{ p^t, \theta^t \} \). Thus, the user’s state at time \( t \) in \( E_{phys} \) is denoted by \( q^t_{phys} = \{ p^t_{phys}, \theta^t_{phys} \} \). The path that a user walks along in an environment is represented by an ordered set of states \( Q = \{ q^0, \ldots, q^n \} \). For brevity, we denote the physical and virtual paths as \( path_{phys} = \{ q^0_{phys}, \ldots, q^n_{phys} \} \) and \( path_{virt} = \{ q^0_{virt}, \ldots, q^n_{virt} \} \), respectively. Finally, note that \( E_{phys} = E_{free} \cup O_{phys} \) and \( E_{virt} = E_{free} \cup O_{virt} \).

3.2 Redirected Walking and Configuration Spaces

Redirected walking is typically implemented by rotating the VE around the user as they walk, so it is natural to think about a redirection controller as an algorithm that rotates the VE according to some criteria on every frame. Instead, our goal is to frame redirection controllers in terms of the simultaneous computation of collision-free trajectories in the physical and virtual spaces. We can visualize this as superimposing the user’s virtual path onto their physical location and applying RDW gains that transform the superimposed path such that the user avoids any obstacles that the path intersects with (see [fig. 2]). In our figures, black shapes represent obstacles, white space is any walkable region in the environment, and the colored regions represent the free space (i.e. the subset of the walkable region visible from the user’s position).

With our goal in mind, we can now formally describe the redirection problem using the definitions from Sect. 3.1. Given two environments \( E_{phys} \) and \( E_{virt} \), the user’s configuration in both environments are \( q_{phys} \) and \( q_{virt} \), respectively. The user explores \( E_{virt} \) by following a collision-free path \( path_{virt} \in E_{free} \). This path corresponds to some physical path \( path_{phys} \in E_{phys} \), for which it is possible that \( path_{phys} \cap O_{phys} \neq \emptyset \). If \( path_{phys} \cap O_{phys} \neq \emptyset \), we wish to find some new optimal path \( path_{phys}^* \) such that \( path_{phys}^* \cap O_{phys} = \emptyset \), i.e. \( path_{phys}^* \in E_{free} \). Let RDW \( (path) \) be a generic redirection function which applies translation \( (\delta_t) \), curvature \( (\kappa_t) \), or rotation \( (\gamma_t) \) gains at each timestep \( t \in path \) to yield a new path \( path^* \). The redirection problem is thus is to find some best function \( RDW^* \) such that \( RDW^*(path_{virt}) = path_{phys}^* \). We seek to develop a redirection controller that executes \( RDW^* \). \( RDW^* \) is subject to multiple constraints:

1. Maximum redirection constraint: Since RDW is limited by human perception, the gains applied by \( RDW^* \) must be bounded...
will collide with the obstacle to their left in the physical space. After
yield a safe
3.3 Finding
RDW\(\star()\)
does not have access to the user’s future virtual path
redirection controller (which executes
in the physical visibility polygon (which represents
RDW\((\cdot)\) is to apply a function
segments that define the boundaries of obstacles in the environment.
Fig. 2: Visualization of the redirected walking problem. If the user tries
to walk on the virtual path \(\text{path}_\text{virt}\) with no redirection applied, they
will collide with the obstacle to their left in the physical space. After
applying redirection, the user instead walks along \(\text{path}_\text{phys}\) and avoids
any collisions. The free spaces \(\text{Free}_\text{phys}\) and \(\text{Free}_\text{virt}\) are shown in blue
and red, respectively.

by the user’s empirically measured perceptual thresholds. This
limits how much redirection can be done at any given moment and
makes it more difficult to avoid \(\text{Obs}_\text{phys}\).

2. Geometric deviation constraint: \(\text{RD}(\text{path}_\text{virt})\) results in a path
\(\text{path}_\text{phys}\) that is geometrically different from \(\text{path}_\text{virt}\), where trans-
lation gains change the length and rotation and curvature gains
change the curvature of \(\text{path}_\text{phys}\) relative to \(\text{path}_\text{virt}\). Stronger
gains correspond to a greater deviation between \(\text{path}_\text{phys}\) and
\(\text{path}_\text{virt}\). Stronger redirection gains have a higher chance of in-
ducing simulator sickness in the user, so we wish to compute
\(\text{path}_\text{phys} \in \text{Free}_\text{phys}\) with the weakest gains possible. Geometri-
cally, this is defined as the path with the lowest deviation from
\(\text{path}_\text{virt}\).

3. Information constraint: Oftentimes, the user’s future virtual
path \(\text{path}_\text{virt}\) is not known, which makes optimizing \(\text{path}_\text{phys}\)
difficult. Additionally, the user’s PE may not be completely
known, depending on the capabilities of the virtual reality system.

Due to the above constraints, developing the perfect redirection function
\(\text{RD}(\cdot)\) for all \(\text{path}_\text{phys}\) is very difficult.
Since it is difficult to develop \(\text{RD}(\cdot)\), we must rely on our \(\text{RD}(\cdot)\)
function in conjunction with a resetting function \(\text{reset}(\cdot)\). The resetting
function is responsible for prompting the user to stop walking and
reorienting them to a safe configuration in \(\text{E}_\text{phys}\). Similarly to \(\text{RD}(\cdot)\),
\(\text{reset}(\cdot)\) applies redirection gains to transform \(\text{path}_\text{phys}\) such that the
resulting physical path is valid, i.e. \(\text{path}_\text{phys} \in \text{Free}_\text{phys}\). However,
\(\text{reset}(\cdot)\) differs from \(\text{RD}(\cdot)\) in that \(\text{reset}(\cdot)\) alters \(\text{path}_\text{phys}\) in order to
yield a safe \(\text{path}_\text{phys}\). In practice, this is performed by having the user
turn \(360^\circ\) in place in \(\text{E}_\text{virt}\), while reorienting by some smaller angle
in \(\text{E}_\text{phys}\) due to rotation gains. Thus, \(\text{reset}(\cdot)\) inserts configurations
into \(\text{path}_\text{virt}\) that cause the user to turn in place in \(\text{E}_\text{virt}\) while applying
rotation gains \(\text{rot}\) to alter the user’s corresponding physical rotation.

3.3 Finding \(\text{RD}(\cdot)\) Using Visibility Polygons
In this section, we present a new redirection controller that uses the
notion of visibility polygons to compute \(\text{RD}(\cdot)\). Our goal is to steer
the user along a path in \(\text{Free}_\text{phys}\). The visibility polygon computed in
\(\text{E}_\text{phys}\) at the user’s position \(p_{\text{phys}}\) is a representation of \(\text{Free}_\text{phys}\) that
can be computed in \(\Theta(n \log n)\) time \([10,40]\), where \(n\) is the number of
segments that define the boundaries of obstacles in the environment.
Thus, in order to take advantage of motion planning techniques, our
controller is based on the visibility polygon.

Considering the problem statement in Sect. 3.2 our controller’s goal
is to apply a function \(\text{RD}(\cdot)\)\(=\) \(\text{path}_\text{phys}\) such that \(\text{path}_\text{phys}\) is in
the physical visibility polygon (which represents \(\text{Free}_\text{phys}\). Our
redirection controller (which executes \(\text{RD}(\cdot)\)) is not predictive, so it
does not have access to the user’s future virtual path \(\text{path}_\text{virt}\). This
makes it effectively impossible to compute the optimal physical path
\(\text{path}_\text{phys}\) as described in Sect. 3.2 since we cannot directly transform
\(\text{path}_\text{phys}\) to make it lie within \(\text{Free}_\text{phys}\). To resolve this, we reframe
the redirection problem slightly and use alignment in our redirection
function \(\text{RD}(\cdot)\). Instead of superimposing \(\text{path}_\text{virt}\) onto \(\text{E}_\text{phys}\), we
superimpose \(\text{Free}_\text{virt}\) onto \(\text{Free}_\text{phys}\), centered on the user (both \(\text{Free}_\text{virt}\)
and \(\text{Free}_\text{phys}\) are represented by visibility polygons). When the user
walks in \(\text{E}_\text{virt}\), their path in the superimposed \(\text{Free}_\text{virt}\) will correspond to
some path within \(\text{E}_\text{phys}\), and hopefully within \(\text{Free}_\text{phys}\). Our controller’s
goal is now to apply redirection such that \(\text{Free}_\text{phys}\) is transformed to
\(\text{Free}_\text{virt}\) as closely as possible. The intuition here is that the
more similar \(\text{Free}_\text{phys}\) is to \(\text{Free}_\text{virt}\), the higher the chance that the
user’s next virtual configuration \(\text{q}_n^{t+1}\) will correspond to a valid physical
configuration \(\text{q}_n^{t+1} \in \text{Free}_\text{phys}\). This process of superimposing \(\text{Free}_\text{phys}\)
and \(\text{Free}_\text{virt}\) is shown in Fig. 3.

We guide this free space matching process using our alignment met-
ic, which we define as the area of the free space in front of the user (see
Sect. 3.3.3). Note that while we framed the problem as transforming
\(\text{Free}_\text{phys}\) in implementation, we simply apply \(\text{RD}\) gains to change
the user’s physical configuration such that their new \(\text{Free}_\text{phys}\) is more
similar to \(\text{Free}_\text{virt}\). The pseudocode that our redirection controller exe-
cutes on every frame is shown in Algorithm 1. We now provide details
of each step of the algorithm.

3.3.1 ComputeVisibilityPolygon\((E, p, \theta)\)
Given an environment \(E\) and a position \(p\) and heading \(\theta\) in that environ-
ment, we compute the visibility polygon using the algorithm described
by Suri et al. \([40]\) (note that \(\theta\) is not used in this initial computation). The
visibility polygon \(P\) is defined by a kernel \(k\) and a set of vertices
\(\{\text{v}_0, \text{v}_1, \ldots, \text{v}_{n-1}\}\). Here, \(k\) is the position of the observer (i.e., the
position \(p\) in the environment \(E\)). The set of vertices defines the edges of \(P\),
where consecutive vertices \(\text{v}_i\) and \(\text{v}_{i+1}\) form an edge.
Fig. 4: An overview of our redirection controller based on visibility polygons. (A) We compute the visibility polygon corresponding to the user’s position in both the physical (blue) and virtual (red) environments. After the visibility polygons are computed, they are divided into regions called “slices” which we use later in our approach to measure the similarity of the two polygons. (B) The “active slice” in the virtual environment is computed. This is the slice of the virtual visibility polygon that the user is walking towards (shown in yellow). (C) The corresponding slice in the physical environment that is most similar to the active slice is computed. Similarity is measured using slice area. (D) Redirected walking gains are applied according to the user’s heading to steer them in the direction of the most similar physical slice that was computed in step (C).

Algorithm 1: Redirection Gain Computation

Result: Redirection gains \( g_r, g_s, g_e \) to apply on the current frame.

\[
P_{\text{phys}} = \text{ComputeVisibilityPolygon}(E_{\text{phys}}, P_{\text{phys}}, \theta_{\text{phys}}); \]
\[
P_{\text{virt}} = \text{ComputeVisibilityPolygon}(E_{\text{virt}}, P_{\text{virt}}, \theta_{\text{virt}}); \]
\[
s^\text{phys} = \text{GetMostSimilarPhysicalSlice}(P_{\text{phys}}, s^\text{virt}); \]
\[
s^\text{virt} = \text{GetActiveVirtualSlice}(P_{\text{virt}}, \theta_{\text{virt}}); \]
\[
g_r, g_s, g_e = \text{GetRedirectionGains}(s^\text{virt}, s^\text{phys}, \theta_{\text{phys}}); \]

• Average length (\( s_l \)): The average length of the two line segments connecting \( s[1] \) and \( s[2] \) to \( s[0] \):

\[
s_l = \frac{|s[0] - s[1]| + |s[0] - s[2]|}{2}. \tag{2}\]

• Angle offset (\( s_\theta \)): The angular distance between the user’s heading and the slice bisector:

\[
s_\theta = |\theta - s_\phi|. \tag{3}\]

• Slice area (\( s_A \)): The area of the triangular slice \( s \).

The composition of a slice is shown in Fig. 5b. The polygon \( P \), and all of its slices, are returned at the end of ComputeVisibilityPolygon(\( E, P, \theta \)).

3.3.2 GetActiveVirtualSlice(\( P_{\text{virt}}, \theta_{\text{virt}} \))

We define the “active slice” as the slice that the user is walking towards in the VE. For this work, we assume that the user walks in the direction \( \theta_{\text{virt}} \) that they are facing. Thus, the active slice \( s^\text{virt} \) is defined as:

\[
s^\text{virt} = \arg\min_{s \in P_{\text{virt}}} |s_\beta - \theta_{\text{virt}}|. \tag{4}\]

We return \( s^\text{virt} \) at the end of GetActiveVirtualSlice(\( P_{\text{virt}}, \theta_{\text{virt}} \)).

3.3.3 GetMostSimilarPhysicalSlice(\( P_{\text{phys}}, s^\text{virt} \))

This function computes the slice \( s^\text{phys} \in P_{\text{phys}} \) that is most similar to \( s^\text{virt} \). Here, we leverage our similarity metric to measure the alignment of slices in \( P_{\text{phys}} \) to the active virtual slice \( s^\text{virt} \). Our slices are triangles, so we wish to compute a similarity metric that accurately measures how similar two triangles are. Furthermore, the slices represent regions of free space that the user can walk in, so an ideal similarity metric would be able to compare two slices according to their similarity with regards to both shape and navigability. Measuring shape similarity is a very well-studied problem in geometric computing [46]. Likewise, the relationship between an environment’s structure and navigation in that environment is well-studied [9, 16, 27]. Our goal is to design a metric that can compute the similarity of shapes with respect to the geometric structure as well as navigability. Therefore, we use a geometric measure of similarity (slice area), but we constrain the set of physical slices that we consider according to perceptual heuristics that tend to guide human locomotion (we only consider slices in the user’s field of view). We chose slice area over other shape similarity measures since we are primarily concerned with the user’s proximity to \( \text{Obs}_{\text{phys}} \), which is described by the slice’s total area.

Given the physical visibility polygon \( P_{\text{phys}} \), we first compute the set of eligible slices, which we will compare against \( s^\text{virt} \). This set, denoted
We conducted four experiments to evaluate the performance of our algorithm. For each experiment, we compared our visibility-based steering algorithm against the alignment-based redirection controller (ARC) presented by Williams et al. [48], the artificial potential field (APF) controller by Thomas et al. [42], and the implementation of steer-to-center (S2C) by Hodgson et al. [17]. Our algorithm’s reset function is the same as the one used by ARC [48]. Both APF and S2C use the modified reset-to-center algorithm introduced by Thomas et al. [42].

ARC is currently the best-performing reactive controller, while controllers based on potential-fields also perform fairly well. We compare against S2C since it is a very common benchmark to compare against in the RDW literature; however, we note that S2C is not expected to do well in any of our experiments due to the obstacles present in the PE. The first three experiments involve only static scenes with no dynamic obstacles. Our fourth experiment investigates controller performance in dynamic scenes.

It is known that the structure of the environment affects a redirection controller’s performance [25], so it is important to consider the environments’ layouts when assessing a controller’s efficacy. There is currently no standard suite of test environments with which we can evaluate our algorithm, so we opted to test in environments used before to move towards a more standard set of environments. Furthermore, we wish to test our algorithm in cluttered environments with many obstacles and narrow passageways, which are challenging scenarios where avoiding obstacles is non-trivial. Thus, we evaluated our algorithm using two of the environment pairs from Williams et al. [48]. When choosing our physical environments, we wished to test environments that did not represent traditional physical tracked spaces which often have very few obstacles. Our goal is to test the viability of redirection in unstructured, irregular environments that may be commonly encountered in the real world, such as an office building with cubicles or a living room with tables and couches.

The performance metric we use is the number of resets incurred over the entire duration of the walked path since it is a fairly standard performance metric in the RDW community. Another common metric is the average virtual distance walked between resets, but this metric is dependent on the number of resets, so it is slightly redundant to include both metrics in our evaluation. Before conducting our experiments, we developed three hypotheses:

H1 Our visibility-based steering algorithm will result in fewer collisions than the current state-of-the-art controllers in static scenes.

H2 Our visibility-based steering algorithm will result in fewer collisions than the current state-of-the-art controllers in dynamic scenes.

H3 Redirection controllers that use alignment will perform better in physical-virtual environment pairs that have more local similarity than they will in environment pairs that have less local similarity.

4.1 Environment Pairs

Each experiment had a different pair of physical and virtual environments. The environments are shown in Figure 6. Experiment 1 includes a 12m × 12m physical and 17m × 12m VE consisting of narrow corridors. Experiment 2 has a 10m × 10m PE with three rectangular obstacles and a 20m × 20m VE with many convex and non-convex obstacles. Using these two environments makes it easier to draw comparisons between our paper and that of Williams et al. [48]. Experiment 3 uses the PE from Experiment 1 and the VE from Experiment 2. We noticed that the pairs of environments used in Environment B and C in [48] (Experiments 1 and 2 here) appear to have a high degree of local similarity. That is, the user’s proximity to obstacles will be roughly similar between the physical and virtual environments. Environment B, both physical and virtual environments have irregularity in the sizes and shapes of obstacles in both environments. Therefore, for our third experiment, we opted to evaluate the controllers on a mixture between the environments from Experiments 1 and 2 since this would lower the degree of local similarity between the physical and virtual environments and allow us to test H3.

Finally, in Experiment 4, we used the physical environment from Experiment 2 and an empty 10m × 10m environment with four dynamic, circular obstacles as the virtual environment. We chose to use a VE with no static obstacles so that we could more easily study the influence of dynamic obstacles on the controller’s performance. That is, in an environment with both static and dynamic obstacles, it may be difficult to determine to what degree either type of obstacle influences the controller’s performance.
4.2 Simulated Environment

To evaluate the effectiveness of our algorithm, we conducted extensive experiments with a simulated user walking in virtual reality. Simulation has become a popular method of evaluation for redirection controllers since it allows researchers to quickly iterate on their algorithms and run large-scale experiments in a variety of environments [4, 6, 11, 22, 26, 38, 41, 43, 48]. Our simulated user is represented as a circle with radius 0.5m, and a reset is incurred whenever they came within 0.2m of any obstacle in the PE. The user walked with a speed of 1m/s and turned with a speed of 90°/s. Our simulation timestep size was 0.05.

In our simulation, the model for generating the paths of the user is slightly different for static and dynamic scenes. For static scenes, we used the motion model that was introduced by Azmandian et al. [2] and has been used by others [41, 49]. For dynamic scenes, we used ORCA [45] to generate trajectories for the user and the dynamic obstacles since ORCA generates smooth, collision-free paths for multiple objects in the same environment.

4.3 Experiment Design

For each of the static environment experiments, we generated 100 paths using the path model developed by Azmandian et al. [2]. The average path length in these experiments was roughly 350m. We ran our simulation on these 100 paths once with each of the redirection controllers we evaluated (our visibility-based controller, ARC [48], APF [42], and S2C [17]). For each path, the user starts in a random location in the physical and virtual environments. The user also has a random heading in both environments. It is important to clarify that these random starting configurations were different between the 100 paths but were the same each time we simulated a particular path.

For the dynamic scene, we generated 100 collision-free paths for the user and the four dynamic obstacles in the VE using the ORCA [45]. These paths had an average length of 136m. These paths were shorter than those used in static environments because we generated the dynamic paths to take roughly the same amount of timesteps to complete as in the static experiments, before considering time taken for resets. As we did in the static scenes, all redirection controllers were evaluated on the same 100 paths that we generated with ORCA.

5 Results

We compared the number of resets across all 100 paths for the four algorithms that we tested in our experiments. Some of our data violated assumptions of homoscedasticity or normality. To account for these violated assumptions, we compared the controllers’ performance with a robust one-way repeated measures ANOVA with 20% trimmed means. We used the WRS2 package in R to conduct our analyses [25]. For all of the results presented in this section, we include the test statistic (F) and the significance level (p-value). We also include the results of the post-hoc comparisons in Table 1, which are computed using linear contrasts. For the post-hoc tests, we report the difference between the means (Ψ), the upper and lower confidence intervals ([CI lower, CI upper]), and the significance level (p-value). Note that we use confidence intervals as a measure of effect size [23]. The main focus of this work is to study the efficacy of our visibility-based algorithm, so we do not include the results of the post-hoc tests for any comparisons that do not include our visibility-based steering controller in the main text (e.g. ARC compared to APF). Results comparing the other controllers can be found in the supplementary materials.

5.1 Experiment 1

The robust ANOVA indicated a significant difference in the number of resets incurred when exploring with the different redirection controllers $F(1.88, 111.03) = 46.92, p < .0001$. We see in Fig. 7 that our visibility-based algorithm achieves a median of 121 resets, which is 32 lower than the median of 153 achieved by ARC. The median number of resets achieved by APF and S2C is slightly higher (188.5 and 217.5), and these two controllers have a much larger variance in the number of resets across all 100 paths.
### Table 1: The results of post-hoc pairwise comparisons between our visibility-based algorithm (denoted Vis. Poly.) and the other controllers (ARC, APF, S2C).

| Redirection Controller | Experiment 1 | Experiment 2 | Experiment 3 | Experiment 4 |
|------------------------|--------------|--------------|--------------|--------------|
| **Ψ**                  | **CI**       | **p**        | **Ψ**        | **CI**       | **p**       | **Ψ**        | **CI**       | **p**       |
| Vis. Poly. vs ARC [48]  | -32.7        | [28.9, 36.5] | < .0001      | -5.68        | [-2.73, -8.64] | < .0001      | -24.8        | [-20.6, -29.0] | < .0001      |
| Vis. Poly. vs APF [42]  | -76.4        | [-101, -51.7] | < .0001      | -105         | [-117, -92.1]  | < .0001      | -49.6        | [-71.1, -28.0]  | < .0001      |
| Vis. Poly. vs S2C [17]  | -114         | [-143, -85.1] | < .0001      | -342         | [-362, -321]   | < .0001      | -56.9        | [-71.7, -42.1]  | < .0001      |

The **Ψ** value indicates that our algorithm has a lower average number of resets across all 100 paths. The CI columns present the lower and upper bounds of the confidence interval, while the **p** column presents the significance level of the difference between the algorithms. The **Ψ** and CI values are rounded to three significant figures.

---

**6 Discussion**

#### 6.1 Static Scenes

Our first three experiments were conducted in static scenes. The results showed that our novel algorithm based on visibility polygons resulted in significantly fewer resets with obstacles in the environments we tested. Overall, the results in the static scenes supported our first hypothesis (H1) that our visibility-based controller will outperform existing redirection controllers in static scenes. In the first experiment, the physical and virtual environments had a considerable similarity since both environments featured regularly spaced narrow corridors, though the VE was larger than the physical one. The simulated user incurred more resets in this PE than in the one used for Experiment 2. This is likely because the narrow corridors make it difficult for the user to consistently walk without resets.

Experiment 3 used the PE from Experiment 1 and the virtual environment from Experiment 2. We chose this combination in order to study the steering algorithms' performance in environment pairs that have little similarity on a local scale. The PE is very regular and has only 90° turns. However, the VE contains many differently-shaped obstacles, creating irregular turns and regions of differing amounts of free space. We found that all redirection controllers except APF performed worse in this experiment than they did in the others, which we believe is due to the more significant mismatch between the physical and virtual environments. It suggests that controllers based on alignment will experience more difficulty avoiding resets in environment pairs with low local similarity than they will in pairs with high local similarity. Note that APF, which does not steer the user with alignment, was able to perform better in Experiment 3 than in Experiment 1. The result of
7 Conclusion, Limitations, and Future Work

In this paper, we presented a novel formulation of the redirected walking problem and developed a visibility-based redirection controller that takes advantage of this formulation to yield fewer resets during walking. Our formal description of the redirection problem frames it in terms of optimizing the redirection gains such that the user follows a collision-free path in the physical free space while simultaneously traveling on a collision-free path in the virtual free space. We hope that this new formulation will allow researchers to further improve redirection controllers by leveraging motion planning techniques that have been well-studied by the robotics community.

The novel redirection controller that we also presented in this work is based on the key insight that the visibility polygon provides a reliable representation of the user’s free space at their location. By dividing the physical and virtual visibility polygons into “slices” and steering the user towards a physical slice that is similar to the current virtual slice, we were able to avoid significantly more resets than the state-of-the-art controllers were able to. As evidenced by our simulation-based results, our algorithm proved to be more effective in both static and dynamic scenes where the physical and virtual environments had locally dissimilar layouts.

Although our results were positive, it is important to discuss the limitations of this work, too. In our experiments, we only evaluated the algorithms in simulated environments. Simulation can be effective for quickly getting an understanding of a RDW algorithm’s strengths and weaknesses, but full user studies should be conducted to gain a more complete evaluation of a controller’s efficacy and to highlight shortcomings that may not arise in simulations. Another limitation of this work is that we only tested scenarios involving one user in the PE and VE. Many use-cases for VR involve multiple users, either in the same PE or different PEs. However, collecting layout data of the PE is considerably more difficult, since this involves object detection and tracking in real time. This problem can be sidestepped by providing the RDW controller with a map of the PE if it is known beforehand, but this cannot be guaranteed in a commodity VR setting where the application will be used in a variety of PEs. In addition to collecting environment data, it is important to consider the run-time of the algorithm. Visibility computations are very well-studied problems in computer graphics and robotics, so a considerable amount of effort has been invested into developing efficient and robust algorithms for computing visibility. Nevertheless, since many other computations need to be done on each timestep in VR applications, computing visibility polygons in real-time may be difficult if the environment contains a large number of obstacles.

Future work should investigate different ways to use the visibility polygon for more sophisticated steering, such as more accurate measures of shape similarity. Our results showed that for some environments, the performance improvement afforded by visibility polygons was not large (though it was statistically significant). It may be the case that our algorithm does not use visibility polygons to their full potential. There is not a lot of research studying the relationship between the shape of the virtual path and a controller’s performance, so the impact of the path models on a controller’s performance is an open question that warrants more research. Another interesting area for future work is to extend our visibility-based controller to real-world scenarios where the exact geometry of the environments is not known. Our experiments were conducted using reliable simulations to show that our algorithm is effective, but calculating visibility polygons in real PEs will likely introduce new challenges. Finally, extensive in-person user studies should be conducted now that we know that our visibility-based controller can yield significant benefits.
[49] R. Yu, Z. Duer, T. Ogle, D. A. Bowman, T. Tucker, D. Hicks, D. Choi, Z. Bush, H. Ngo, P. Nguyen, et al. Experiencing an invisible world war I battlefield through narrative-driven redirected walking in virtual reality. In 2018 IEEE Conference on Virtual Reality and 3D User Interfaces (VR), pp. 313–319. IEEE, 2018.

[50] M. Zank and A. Kunz. Optimized graph extraction and locomotion prediction for redirected walking. In 2017 IEEE Symposium on 3D User Interfaces (3DUI), pp. 120–129. IEEE, 2017.

[51] 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.
## A Supplementary Materials

### A.1 Environment Layouts

#### A.1.1 Experiment 1

**Experiment 1 (physical)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 | Obstacle 4 |
|--------------|------------|------------|------------|------------|
| \([-6, -6]\), \((6, -6), (6, 6), (-6, 6)\) | \([-4, -4], (-1, -4), (-1, -1), (-4, -1)\) | \([1, -4], (4, -4), (4, -1), (1, -1)\) | \([1, 1], (4, 1), (4, 4), (1, 4)\) | \([-4, 1], (-1, 1), (-1, -4), (-4, -4)\) |

**Experiment 1 (virtual)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 | Obstacle 4 |
|--------------|------------|------------|------------|------------|
| \([-11, -6]\), \((6, -6), (6, 6), (-6, 6)\) | \([-4, -4], (-1, -4), (-1, -1), (-4, -1)\) | \([1, -4], (4, -4), (4, -1), (1, -1)\) | \([1, 1], (4, 1), (4, 4), (1, 4)\) | \([-4, 1], (-1, 1), (-1, -4), (-4, -4)\) |

#### A.1.2 Experiment 2

**Experiment 2 (physical)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 |
|--------------|------------|------------|------------|
| \([-5, -5]\), \((5, -5), (5, 5), (-5, 5)\) | \([-4.5, -4.5], (-2.5, -4.5), (-2.5, -4.5)\) | \([-2.1, 2.1], (-2.1, 2.1)\) | \([-2.4, 2.4], (2.5, -2.5)\) |

**Experiment 2 (virtual)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 |
|--------------|------------|------------|------------|
| \([-10, -10]\), \((10, 10), (-10, 10), (10, 10)\) | \([-4.5, -4.5], (-2.5, -4.5), (-2.5, -2.5)\) | \([-2.1, 2.1], (-2.1, 2.1)\) | \([-2.4, 2.4], (2.5, -2.5)\) |

#### A.1.3 Experiment 3

**Experiment 3 (physical)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 | Obstacle 4 |
|--------------|------------|------------|------------|------------|
| \([-6, -6], (6, -6), (6, 6), (-6, 6)\) | \([-4, -4], (-1, -4), (-1, -1), (-4, -1)\) | \([1, -4], (4, -4), (4, -1), (1, -1)\) | \([1, 1], (4, 1), (4, 4), (1, 4)\) | \([-4, 1], (-1, 1), (-1, -4), (-4, -4)\) |

**Experiment 3 (virtual)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 | Obstacle 4 |
|--------------|------------|------------|------------|------------|
| \([-10, -10], (10, 10), (-10, 10), (10, 10)\) | \([-4.5, -4.5], (-2.5, -4.5), (-2.5, -2.5)\) | \([-2.1, 2.1], (-2.1, 2.1)\) | \([-2.4, 2.4], (2.5, -2.5)\) | \([-2.1, 2.1], (-2.1, 2.1)\) |

#### A.1.4 Experiment 4

**Experiment 4 (physical)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 |
|--------------|------------|------------|------------|
| \([-5, -5], (5, -5), (5, 5), (-5, 5)\) | \([-4.5, -4.5], (-2.5, -4.5), (-2.5, -2.5)\) | \([-2.1, 2.1], (-2.1, 2.1)\) | \([-2.4, 2.4], (2.5, -2.5)\) |

**Experiment 4 (virtual)**

| Boundary     | Obstacle 1 | Obstacle 2 | Obstacle 3 |
|--------------|------------|------------|------------|
| \([-11, -6], (6, -6), (6, 6), (-11, 6)\) | \([-4.5, -4.5], (-2.5, -4.5), (-2.5, -2.5)\) | \([-2.1, 2.1], (-2.1, 2.1)\) | \([-2.4, 2.4], (2.5, -2.5)\) |
### A.2 Additional Results

#### A.2.1 Post-hoc Comparisons For Number Of Resets

Here we show the results of post-hoc comparisons for total number of resets between the other controllers we tested against in our experiments.

| Redirection Controller | Experiment 1 | | | Experiment 2 | | | Experiment 3 | | | Experiment 4 | |
|-------------------------|-------------|---|---|-------------|---|---|-------------|---|---|-------------|---|
|                         | Number of Resets | | |                   | | |                   | | |                   | |
|                         | Experiment 1 | Experiment 2 | Experiment 3 | Experiment 4 | |
| ARC [48] vs APF [42]    | -42.3 [-66.6, -18.1] | < .0001 | -99.0 [-112, -86.4] | < .0001 | -32.0 [-47.6, -16.3] | < .0001 | -59.1 [-77.4, -40.7] | < .0001 |
| ARC [48] vs S2C [17]    | -82.4 [-114, -51.3] | < .0001 | -33.5 [-356, -313] | < .0001 | -32.0 [-47.6, -16.3] | < .0001 | -177 [-197, -157] | < .0001 |
| APF [42] vs S2C [17]    | -35.2 [-71.3, 0.882] | < .01 | -233 [-254, -212] | < .0001 | -8.02 [-31.3, 15.2] | = .350 | -112 [-133, -91.5] | < .0001 |

#### A.2.2 Post-hoc Comparisons For Resets Per Meter

The table below shows the results of post-hoc comparisons between controllers according to the number of resets incurred per meter walked in the virtual environment.

| Redirection Controller | Experiment 1 | | | Experiment 2 | | | Experiment 3 | | | Experiment 4 | |
|-------------------------|-------------|---|---|-------------|---|---|-------------|---|---|-------------|---|
|                         | Resets per Meter | | |                   | | |                   | | |                   | |
|                         | Experiment 1 | Experiment 2 | Experiment 3 | Experiment 4 | |
|                         |             |             |             |             | |
| Vis. Poly. vs ARC [48]  | -.0929 [-.0811, -.105] | < .0001 | -.0157 [-.00746, -.0239] | < .0001 | -.0688 [-.0570, -.0806] | < .0001 | -.0428 [-.0278, -.0577] | < .0001 |
| Vis. Poly. vs APF [42]  | -.220 [-.291, -.149] | < .0001 | -.299 [-.324, -.255] | < .0001 | -.137 [-.198, -.0767] | < .0001 | -.471 [-.598, -.345] | < .0001 |
| Vis. Poly. vs S2C [17]  | -.327 [-.411, -.242] | < .0001 | -.940 [-.995, -.885] | < .0001 | -.158 [-.200, -.116] | < .0001 | -.134 [-.147, -.121] | < .0001 |
| ARC [48] vs APF [42]    | -.123 [-.195, -.0515] | < .0001 | -.274 [-.309, -.239] | < .0001 | -.0697 [-.131, -.00858] | < .01 | -.431 [-.556, -.306] | < .0001 |
| ARC [48] vs S2C [17]    | -.237 [-.325, -.149] | < .0001 | -.920 [-.975, -.864] | < .0001 | -.0886 [-.132, -.0456] | < .0001 | -.130 [-.143, -.117] | < .0001 |
| APF [42] vs S2C [17]    | -.101 [-.207, .00495] | < .05 | -.642 [-.700, -.587] | < .0001 | -.0220 [-.0851, .0412] | = .346 | -.829 [-.975, -.683] | < .0001 |