Generalized WarpDriver: Unified Collision Avoidance for Multi-Robot Systems in Arbitrarily Complex Environments

David Wolinski* and Ming C. Lin*†
*Department of Computer Science, UNC Chapel Hill, NC, USA
{dwolinsk, lin}@cs.unc.edu
†Department of Computer Science, University of Maryland at College Park, MD, USA
lin@cs.umd.edu

***Abstract*—In this paper we present a unified collision-avoidance algorithm for the navigation of arbitrary agents, from pedestrians to various types of robots, including vehicles. This approach significantly extends the WarpDriver algorithm [27] specialized for disc-like agents (e.g. crowds) to a wide array of robots in the following ways: (1) the new algorithm is more robust by unifiying the original set of *Warp Operators* for different non-linear extrapolations of motion into a single, general operator; (2) the algorithm is generalized to support agent dynamics and additional shapes beyond just circles; and (3) with addition of few, simple soft constraints, the algorithm can be used to simulate vehicle traffic. Thanks to the generality of the unified algorithm without special case handling, the new capabilities are tighly integrated at the level of collision avoidance, rather than as added layers of multiple heuristics on top of various collision-avoidance schemes designed independently for pedestrians vs. different types of robots and vehicles.**

I. Introduction

Collision avoidance is a fundamental component of multi-agent navigation, and its study has received much attention due to interest in many fields such as robotics, urban planning, AI, collective motions in biology, computer animation, etc. As a result, there exists a large collection of different approaches, many of which are adaptations for specific applications.

A common simplification for most current approaches is that collision avoidance between **holonomic**, **disc-shaped** agents (e.g. humans) is assumed to conserve **constant velocity** for the purpose of collision prediction. In robotics for instance, the velocity-obstacle [8, 23, 12, 24], and inevitable-collision-state [9, 7, 10] approaches are very popular due to their assurance of collision-free motions inside their corresponding time horizons. Methods presenting these three same characteristics have also been widely investigated in graphics and pedestrian dynamics [21, 18, 19, 14, 13, 20, 15]. However, for many practical applications, these three assumptions could lead to some limitations in their applicability and/or higher complexity of the final solution. While a human could for instance be considered holonomic at sufficiently low traveling speeds, it is not the case of most robots or vehicles. While a human again could be considered disc-shaped at low densities, this too is not the general case (e.g. vehicles). Finally, while an agent's motion is likely to be linear in an open, empty environment, this assumption is also not applicable with the introduction of interactions with other agents and in particular environment layouts. Among these, the dynamic constraints of agents' motion have been most studied, almost exclusively in robotics, where this issue most needs to be addressed. Hence the velocity-obstacle approach has seen multiplie adaptations to address this issue [17, 26, 25, 16, 2, 4, 3].

Since an agent's dynamics is one source of non-linear motions, this aspect also has received some attention [22] (also an adaptation of velocity obstacles). However, agents' dynamics are not the only source of non-linear motions. With the recent interest in robots interacting with humans (aka "social" robots), it may become increasingly difficult to further adapt most of the existing collision avoidance approaches to account for the many assumptions on linear-motion behaviors.

Finally, the shapes of agents have received relatively little attention as well (with two recent examples of adaptions of velocity obstacles [11, 5]). This neglect is likely due to the fact that most mobile robots can be reasonably bounded by a disc in 2D or a cylinder in 3D. However, with increasing interest in autonomous vehicles, such assumptions can become problematic or introduce inefficiency. In fact, the issue of interfacing different collision-avoidance algorithms for specialized classes of objects (e.g. pedestrians vs. vehicles), and transitioning between them (e.g. vehicles on roads to vehicles in open environments, such as exposition grounds) can introduce a new level of complexity to the problem.

Given these challenges, including dynamics, non-linear motions, and arbitrarily shaped geometry, addressing all these issues simultaneously poses an even greater challenge. In order to solve these problems, we build on the concept of "WarpDriver" [27] and extend it into a unified algorithm with the following combined characteristics: (1) built-in capability to handle arbitrary sensing error (a base feature of WarpDriver), (2) support for any sources of non-linear motion and agent dynamics, (3) general applicability for a variety of agent shapes, (4) ability to cope with any environment constraints, and (5) a relatively low combined computational cost for the aforementioned features (20% overhead).

To sum up, our contributions are as follow:

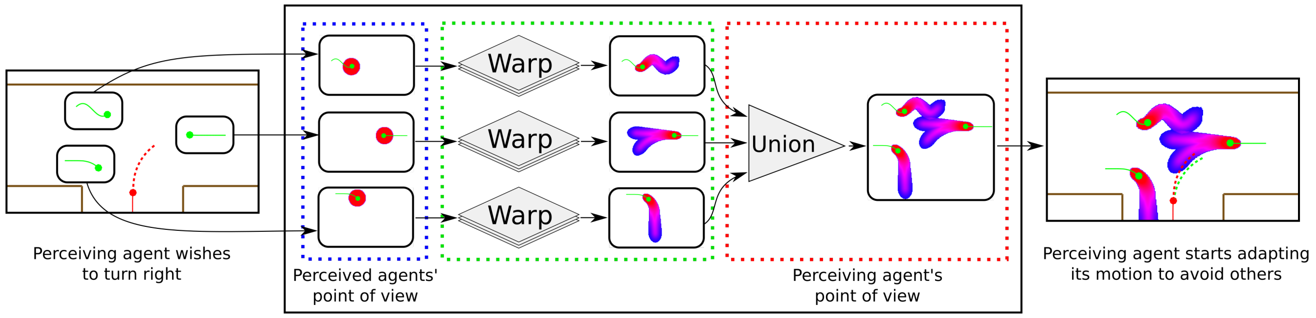
Perceiving agent wishes to turn right
Warp
Warp
Warp
Union
Perceived agents' point of view
Perceiving agent's point of view
Perceiving agent starts adapting its motion to avoid others

Fig. 1. Overview of the algorithmic framework of WarpDriver.

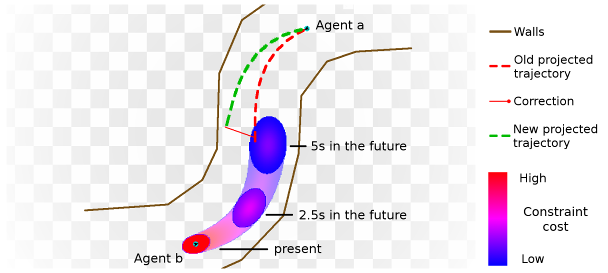

Fig. 2. Collision avoidance example between two agents a and b. The color gradient represents the constraint induced by b over a. The red dotted line is a's initial trajectory. The green dotted line is a's corrected trajectory (exaggerated). The solid red line between the dotted ones is the correction agent a will perform (exaggerated).

- Unifying a number of ad-hoc WarpDriver's *Warp Operators* related to non-linear motion into a single *Warp Operator* based on a function modeling probable future motion, of which we offer a sample implementation incorporating agent dynamics and fast path-planning.
- Addition of support for various shapes based on fast Minkowski sums of convex polygons with even numbers of sides and where each pair of opposite sides is parallel and of equal length (lines, parallelograms, parallel hexagons, parallel octagons, etc.)
- Introduction of simple modeling of path-constrained motion.

II. The Method

In this section, we briefly summarize the original WarpDriver algorithm [27], as well as the changes we make.

A. WarpDriver Algorithm

The WarpDriver algorithm introduced a modular and computationally tractable way of avoiding collisions between agents based on their collision probabilities. At its core, at each timestep, it tries to find a trajectory for each agent that has a lower chance of collision along it. During this process, the navigated agent (*perceiving* agent) is reduced to a point and all static and dynamic obstacles (*perceived* agents) are represented through their probability of colliding with it.

In order to achieve this collision avoidance in real time between thousands of agents and while keeping a clear implementation, the algorithm adopts a modular pipeline. At a glance, there are three main parts:

- The first individually constructs the collision probabilities between the *perceiving* agent and every other *perceived* agent (blue and green dotted rectangles in Figure 1).
- The second part combines the collision probabilites of all *perceived* agents such that the *perceiving* agent can tell at any point in space and time what its chances of colliding with anything are (red dotted rectangle in Figure 1).
- The third and final part is a solver that uses these collision probabilities to compute a new trajectory with a lower overall collision probability (right side of Figure 1, and Figure 2).

In order to keep implementation details as independent from each other as possible, the algorithm adopts the following conventions:

- The solver operates under the assumption that the *perceiving* agent's trajectory is linear, which means that all non-linear factors in the agents' motions are confined to the collision probability construction step.
- All *properties* (e.g. size, speed...) of the agents are dealt with separately to keep the design modular and simplify extending the algorithm:
  - Collision probabilities between a pair of agents are computed cumulatively,
  - using the lowest common factor between agents (their co-existence in space and time, called *Intrinsic Field*, blue dotted rectangle in Figure 1),
  - which is further warped by *Warp Operators* for each agent *property* to be modeled (green dotted rectangle in Figure 1).

With such an approach, a common, constant brick is used for all interactions (*Intrinsic Field*) and is cumulatively "sculpted" (by *Warp Operators*) to form the collision probabilities between any pair of *perceiving-perceived* agents. Then, all these interactions are combined into one using classical unions of probabilities. Finally, the solver computes a new velocity resulting in a trajectory with a lower overall collision probability.

In this work, we mainly focus on the step where collision probabilities are constructed for a *perceiving-perceived* pair of agents, leaving the union of probabilities and solver unchanged.

B. Overview

In the following text, normal-face, lower-case fonts represent scalar values, and bold-face, lower-case fonts represent vectors (e.g. in $\mathbb{R}^3$).

In practice, at timestep k and for a set of agents $\mathcal{A}$, at any point in 2D space and time $\mathbf{s} \in \mathbb{R}^3$, the solver has access to the collision probability $p_{a \to \mathcal{A} \backslash a, k}(\mathbf{s})$ between a *perceiving* agent a and all its neighbors $\mathcal{A} \backslash a$, as well as the associated collision probability gradient $(\nabla p_{a \to \mathcal{A} \backslash a, k})(\mathbf{s})$. They are both computed from pair-wise interactions through unions of probabilities, where a pair-wise interaction between a *perceiving* agent a and a *perceived* neighbor b yields a collision probability $p_{a \to b, k}(\mathbf{s})$ and a probability gradient $(\nabla p_{a \to b, k})(\mathbf{s})$, and is computed as follows.

Let the scalar field $I \,|\, \forall \mathbf{s} \in \mathbb{R}^3, I(\mathbf{s}) \in [0,\, 1]$ be the *Intrinsic Field*, $\mathbf{W} = W_n \circ ... \circ W_1$ the composition of given *Warp Operators* $\{W_1, ..., W_n\}$ ($\circ$ denotes the composition operator), and $\mathbf{W}^{-1} = W_1^{-1} \circ ... \circ W_n^{-1}$ the inverse of $\mathbf{W}$, where for any *Warp Operator* $W \in \{W_1, ..., W_n\}$, W^{-1} is the inverse of W. Note that the wording "inverse" is slightly abusive here and is only meant to imply that each W^{-1} undoes the corresponding W in a geometrical sense.

Then, the pair-wise collision probabilities and their gradients between a *perceiving* agent a and a *perceived* agent b are:

$$p_{a \to b, k}(\mathbf{s}) = (\mathbf{W}^{-1} \circ I \circ \mathbf{W})(\mathbf{s}) \tag{1}$$

$$(\nabla p_{a \to b, k})(\mathbf{s}) = (\mathbf{W}^{-1} \circ (\nabla I) \circ \mathbf{W})(\mathbf{s}) \tag{2}$$

In this formulation, agents' *properties* are implemented as *Warp Operators*, independently of other mechanisms, making it straightforward to extend the algorithm with new capabilities. Hence, the present work focuses on the set of used *Warp Operators*.

Originally, WarpDriver was proposed along with an initial set of various *Warp Operators* as follows:

Position and Orientation : the W_{po} operator handles the agents' positions and orientations.

Velocity : the W_v operator handles the agents' instantaneous velocities.

Environment Layout : if used, the W_{el} operator models the layout of the environment (e.g. intersections, curved paths...).

Obstacle Interaction : if used, the W_{oi} operator models how static obstacles affect agents (e.g. stopping at a wall, circumventing an obstacle...).

Observed Behaviors : if used, the W_{ob} operator tries to predict the trajectory of an agent based on its past trajectory (independently of speed).

Time Horizon : the time horizon operator W_{th} determines how far into the future agents avoid collisions.

Radius : the W_r operator sets the radii of agents.

Time Uncertainty : The W_{tu} operator increases uncertainty on other agents' states as we look further into the future.

Velocity Uncertainty : Due to an agent's speed, certain motion adaptations are easier (more likely) than others (e.g. at higher speeds, it is easier to accelerate/decelerate than turn); denoted W_{vu}.

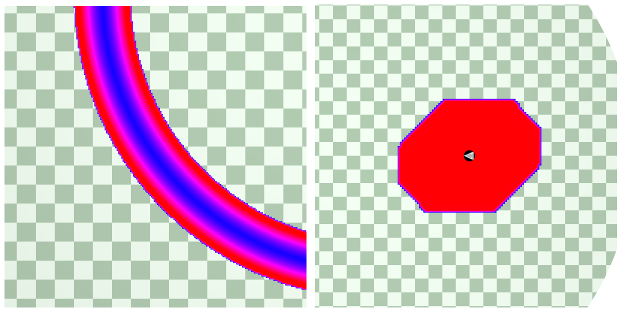
Fig. 3. Left: biarc-induced constraint. Right: sum of two 2-by-4 rectangles at a 45° angle.

In this work, we:

- Remove the "position and orientation", "environment layout", "obstacle interaction", "observed behaviors" and "velocity" *Warp Operators* which were used interchangeably in an ad-hoc fashion, replace them with two *Warp Operators* W_{l1}, W_{l2} (split to allow other *Warp Operators* requiring access to world referential to be inserted) into which any motion prediction function can be plugged (in the present work we use a fast, pre-computed path-planner), and add support for agents' dynamic constraints.
- Add two *Warp Operators* W_{sh} and W_o to handle agents' shapes (and shape offsets).
- Add a *Warp Operator* W_a to handle road boundaries as soft constraints (which can be broken if the car needs to steer off the road to avoid a collision as a last resort) for traffic-like simulation.

III. Generalized WarpDriver

In order to develop a more unified WarpDriver algorithm, we have made a number of changes and introduced new features that are summarized in this section: generalization of non-linear motions, generalization of agents, and generalization of environment (applied to traffic).

As previously mentioned, we implement the proposed work as *Warp Operators* to be included in the WarpDriver pipeline. In Equation 1, I is a scalar field and thus takes a point in space and time and outputs a probability: $\forall \mathbf{s} \in \mathbb{R}^3, I(\mathbf{s}) \in \mathbb{R}$. In Equation 2, ∇I is a gradient over a scalar field and thus takes a point in space and time and outputs a probability gradient: $\forall \mathbf{s} \in \mathbb{R}^3, \nabla I(\mathbf{s}) \in \mathbb{R}^3$. Thus, in order to fully define a *Warp Operator*, one needs to define three operations:

- Every *Warp Operator* W composing the chain $\mathbf{W}$ on the right of I and ∇I in Equations 1 and 2 takes and outputs coordinates in space and time $W : \mathbb{R}^3 \to \mathbb{R}^3$.
- Every inverse *Warp Operator* W^{-1} composing the inverse chain $\mathbf{W}^{-1}$ on the left of I in Equation 1 takes and outputs a probability $W^{-1} : [0,\, 1] \to [0,\, 1]$.
- Every inverse *Warp Operator* W^{-1} composing the inverse chain $\mathbf{W}^{-1}$ on the left of ∇I in Equation 2 takes and outputs a probability gradient $W^{-1} : \mathbb{R}^3 \to \mathbb{R}^3$.

Therefore, for each *Warp Operator* W described in this section, we give $W(\mathbf{s}) \; \forall \mathbf{s} \in \mathbb{R}^3$, $W^{-1}(p) \; \forall p \in [0,\, 1]$, and $W^{-1}(\mathbf{g}) \; \forall \mathbf{g} \in \mathbb{R}^3$. A the end of this section, we summarize the changes by giving the finalized chain of *Warp Operators*.

A. Additional Notations and Terms

Here is a list of notations and terms used in the rest of the exposition:

- $\cdot$, $\times$, $\star$ and $*$ respectively denote the dot product, cross product, component-wise multiplication and convolution.
- Three commonly used vectors are $\mathbf{x} = (1, 0, 0)$, $\mathbf{y} = (0, 1, 0)$, and $\mathbf{t} = (0, 0, 1)$, respectively representing the axes for 2-dimentional positions and time in the current referential. They also make it simple to access the three components $\mathbf{s} \cdot \mathbf{x}$, $\mathbf{s} \cdot \mathbf{y}$, and $\mathbf{s} \cdot \mathbf{t}$ of a vector $\mathbf{s} \in \mathbb{R}$.
- An agent's referential refers to a coordinate system in $\mathbb{R}^3$ where the origin is at the agent's intended center of rotation, the agent "faces" along the local $\mathbf{x}$ vector, $\mathbf{y}$ is to the agent's "left", and $\mathbf{t}$ remains the "up" vector.
- $\forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^3, angle(\mathbf{u}, \mathbf{v}) \in \mathbf{R}$ is the direct angle between vectors $\mathbf{u}$ and $\mathbf{v}$.
- $\forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^3, \forall \theta \in \mathbb{R}, rotate_{\mathbf{v}}(\mathbf{u}, \theta) \in \mathbb{R}^3$ rotates vector $\mathbf{u}$ around vector $\mathbf{v}$ by angle θ.

B. Generalization of Non-linear Motions

The original WarpDriver algorithm had a number of *Warp Operators* that dealt specifically with trajectories. The "position and orientation" and "velocity" operators allowed to make linear motion prediction, while the "environment layout", "obstacle interaction", and "observed behaviors" operators (respectively for environment-related curved paths, changes of motion due to walls, and extrapolation of pattern-like motions) were substituted for "position and orientation" when needed.

This on-the-fly substitution, external to the collision-avoidance mechanism, means that these operators cannot be properly combined (e.g. interacting with obstacles while on a curved path).

Thus, we replace these alternating operators with a single mechanism, which constructs an agent's future likely trajectory by using a function v which returns the velocity $v(\mathbf{s})$, $\forall \mathbf{s} \in \mathbb{R}^3$ (also $v(\mathbf{s}) \in \mathbb{R}^3$, and $v(\mathbf{s}) \cdot \mathbf{t} = 1$) that agent a would follow in the absence of constraints at position and time $\mathbf{s}$. The likely trajectory is a collection of n positions and times $L_{a,k} = \{\mathbf{s}_1, \ldots, \mathbf{s}_n\} \in (\mathbb{R}^3)^n$ where $\mathbf{s}_1$ is the position of agent a at timestep k in world referential, and $\forall i \in [\![1, n-1]\!], \mathbf{s}_{i+1} = \mathbf{s}_i + u\, v(\mathbf{s}_i)$ where u is the duration between consecutive $\mathbf{s}_i$.

Then, we define two operators W_{l1} and W_{l2}, which respectively warp a point from the *perceiving* agent's referential into world referential, and from world referential into the *perceived* agent's referential.

With two interpolation functions $pos_a(t) \in \mathbb{R}^3$ and $ori_a(t) \in \mathbb{R}$, which given the likely trajectory $L_{a,k}$, respectively return the *perceiving* agent's position along $L_{a,k}$ at time t, and its corresponding orientation at that time:

$$W_{l1}(\mathbf{s}) = (pos_a(t) \cdot \mathbf{x},\ pos_a(t) \cdot \mathbf{y},\ t),$$
$$W_{l1}^{-1}(p) = p,$$
$$W_{l1}^{-1}(\mathbf{g}) = rotate_{\mathbf{t}}(\mathbf{g}, -ori_a(t)),$$
$$\text{with } t = \mathbf{s} \cdot \mathbf{t}.$$

Here, $W_{l1}(s)$ needs no rotation, since in practice it is only ever evaluated on the *perceiving* agent's center of rotation.

Similarly, with interpolating functions $pos_b(t) \in \mathbb{R}^3$ and $ori_b(t) \in \mathbb{R}$ which respectively return the *perceived* agent's position and orientation in world referential at time $t \in \mathbf{R}$ from $L_{b,k}$:

$$W_{l2}(\mathbf{s}) = rotate_{\mathbf{t}}(s - pos_b(t), -ori_b(t)),$$
$$W_{l2}^{-1}(p) = p,$$
$$W_{l2}^{-1}(\mathbf{g}) = rotate_{\mathbf{t}}(\mathbf{g}, ori_b(t)),$$

Obviously, function v benefits from being fast to compute. This function is meant to be a genral interface for any motion prediction algorithm (which might be highly application-specific), and in this work we connected it to our path-planner which returns the direction of the shortest path between a current position and a destination; as this path-planner makes as many computations as possible ahead of time, in our test cases v is computed on average in less than $0.3\mu s$. The implementation details for this precomputed path-planner can be found in the Appendix.

C. Generalization of Agents

In order to apply WarpDriver to agents other than pedestrians, we need to be able to deal with shapes other than circles as well as the agents' dynamics.

1) Shapes: The formal way to deal with shapes is through the Minkowski sum of the *Intrinsic Field*. For memory, the *Intrinsic Field* was defined in WarpDriver as the lowest common denominator between agents, that is their co-existence in space and time. In practice, this scalar field results from a convolution between two functions. The first one is a sensing-error function, which so far is the bivariate normal density function (and *Warp Operators* can model additional uncertainty): $\forall x, y \in \mathbb{R},\ f(x, y) = \frac{1}{2\pi} exp(-(\frac{x^2+y^2}{2}))$. The second function is the Minkowski sum of two circular agents, simply a disc-shaped step function of normalized radius 1:

$$\forall x, y \in \mathbb{R},\ g(x, y) = \begin{cases} 1, & \text{if } \sqrt{x^2 + y^2} \leq 1 \\ 0, & \text{otherwise} \end{cases}$$

The *Intrinsic Field* is then defined as:

$$\forall \mathbf{s} \in \mathbb{R}^3,\ I(\mathbf{s}) = \begin{cases} (f * g)(\mathbf{s} \cdot \mathbf{x}, \mathbf{s} \cdot \mathbf{y}), & \text{if } \mathbf{s} \cdot \mathbf{t} \in [0,\ 1] \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

This computation, involves a convolution with a possibly non-trivial error function, and the Minkowski sum in general depends on the agents' relative orientation. Setting agents' shapes to discs here makes the *Intrinsic Field* constant throughout the simulation and allows its precomputation (WarpDriver's original approach). We wish to leave it precomputed, thus motivating our decision to implement shapes through a *Warp Operator*.

It can be noted that since the *Intrinsic Field* as well as the scalar fields resulting from warping it are volumetric objects, and circles/lines/rectangles have simple symmetries, Minkowski sums between them can be implemented in a *Warp*

Operator W_{sh} by successively "cutting" and "stretching" the scalar fields.

Assuming a circle-like (in the $(\mathbf{x}, \mathbf{y})$ plane) volumetric object that is not reduced to a point and centered at the origin $\mathbf{o}$ (i.e. the *Intrinsic Field*), one cut-and-stretch operation will make it into a line (with a non-null thickness), another such operation into a parallelogram (either an agent's own shape or the sum of two line agents), yet another into a parallel hexagon (the sum of a parallelogram and a line), and a final one into a parallel octagon (sum of two parallelogram agents). In fact, this could be continued to model the Minkowski sum of any number of agents whose shape is a convex polygon with an even number of sides where opposite sides are parallel and of equal length.

We start by assuming that each agent a is characterized by a set of cut vectors $\mathcal{C}^c_a = \{\mathbf{c}^c_{a,1}, \ldots, \mathbf{c}^c_{a,n}\} \in (\mathbb{R}^3)^n$, a set of stretch vectors $\mathcal{C}^s_a = \{\mathbf{c}^s_{a,1}, \ldots, \mathbf{c}^s_{a,n}\} \in (\mathbb{R}^3)^n$, and a set of stretch distances $\mathcal{C}^d_a = \{c^d_{a,1}, \ldots, c^d_{a,n}\} \in \mathbb{R}^n$ along the corresponding stretch vectors. For example, for a wall a of total width $3m$ facing along the local $\mathbf{x}$ axis, $\mathcal{C}^c_a = \{(1,0,0)\}$, $\mathcal{C}^s_a = \{(0,1,0)\}$, and $\mathcal{C}^d_a = \{1.5\}$. For a car a of width $1.8m$ and length $4.6m$ facing along the local $\mathbf{x}$ axis, $\mathcal{C}^c_a = \{(1,0,0),(0,1,0)\}$, $\mathcal{C}^s_a = \{(0,1,0),(1,0,0)\}$, and $\mathcal{C}^d_a = \{0.9, 2.3\}$.

Then at time $t \in \mathbb{R}$, with $\theta = angle(ori_b(t), ori_a(t))$, for a *perceiving* agent a and a *perceived* agent b, we note the combined set of cut vectors $\mathcal{C}^c_{a,b} = \{\mathbf{c}^c_1, \ldots, \mathbf{c}^c_n\} = \{rotate_{\mathbf{t}}(\mathbf{c}^c, \theta) \mid \mathbf{c}^c \in \mathcal{C}^c_a\} \bigcup \mathcal{C}^c_b$, the combined set of stretch vectors $\mathcal{C}^s_{a,b} = \{\mathbf{c}^s_1, \ldots, \mathbf{c}^s_n\} = \{rotate_{\mathbf{t}}(\mathbf{c}^s, \theta) \mid \mathbf{c}^s \in \mathcal{C}^s_a\} \bigcup \mathcal{C}^s_b$, and the combined set of stretch distances $\mathcal{C}^d_{a,b} = \{\mathbf{c}^d_1, \ldots, \mathbf{c}^d_n\} = \mathcal{C}^d_a \bigcup \mathcal{C}^d_b$. The rotations used during the construction of the combined sets of cut and stretch vectors deal with the relative orientation between both agents. In order to account for the combinatorial interactions between the cut vectors, we define $\hat{\mathcal{C}}^c_{a,b} = \{\hat{\mathbf{c}}^c_1, \ldots, \hat{\mathbf{c}}^c_n\}$ using a helper sequence H where $H_1 = \{c^d_1 \mathbf{c}^s_1,\ -c^d_1 \mathbf{c}^s_1\}$, and $\forall i \in [\![2, n]\!]$, $H_i = \{\mathbf{c}^s + c^d_i \mathbf{c}^s_i,\ \mathbf{c}^s - c^d_i \mathbf{c}^s_i \mid \mathbf{c}^s \in H_{i-1}\}$. In fact, H_i contains the vertices of the polygon constructed by the first i cut-and-stretch operations. Then, $\hat{\mathbf{c}}^c_1 = \mathbf{c}^c_1$, and $\forall i \in [\![2, n]\!], \hat{\mathbf{c}}^c_i = \frac{\mathbf{c}}{\|\mathbf{c}\|}$, where $\mathbf{c} = center(\operatorname{argmax}_{\mathbf{c}^s \in H_{i-1}} \|\mathbf{c}^c_i \cdot \mathbf{c}^s\|)$). Note that we use a function $center$ which computes the centroid of its input vectors, since argmax returns a set of either one or two elements, the latter case happens when we try to cut through (and perpendicularly to) the middle of a face of the so-far-computed polygon.

With the previous definitions, W_{sh} is then computed as follows:

$$\begin{aligned} W_{sh}(\mathbf{s}) &= \mathbf{s}_0, \\ W^{-1}_{sh}(p) &= p, \\ W^{-1}_{sh}(\mathbf{g}) &= \mathbf{g}_n, \end{aligned}$$

where:

$$\begin{gathered} \mathbf{s}_n = \mathbf{s} \text{ and } \mathbf{s}_{i-1} = (0,\ 0,\ \mathbf{s}_i \cdot \mathbf{t}) + \alpha \hat{\mathbf{c}}^c_i + \gamma \mathbf{c}^s_i, \\ \alpha = \frac{(\mathbf{s}_i \times \mathbf{c}^s_i) \cdot \mathbf{t}}{(\hat{\mathbf{c}}^c_i \times \mathbf{c}^s_i) \cdot \mathbf{t}},\ \beta = \frac{(\mathbf{s}_i \times \hat{\mathbf{c}}^c_i) \cdot \mathbf{t}}{(\mathbf{c}^s_i \times \hat{\mathbf{c}}^c_i) \cdot \mathbf{t}}, \\ \gamma = \begin{cases} \beta - c^d_i \frac{\beta}{|\beta|}, & \text{if } |\beta| > c^d_i + \epsilon \\ \epsilon \frac{\beta}{|\beta|}, & \text{otherwise} \end{cases} \\ \mathbf{g}_0 = \mathbf{g} \text{ and } \mathbf{g}_i = rotate_{\mathbf{t}}(\mathbf{g}_{i-1}, -angle(\mathbf{c}^c_i, \hat{\mathbf{c}}^c_i)). \end{gathered}$$

Since the cut-and-stretch operations are defined through their cut and stretch direction vectors, this *Warp Operator* works in the case where the center of rotation of the agent is at the center of its circle/line/rectangle shape. To handle cases where this is not true (e.g. the local center of rotation of a typical car is at the middle of its rear axle), we add an offset *Warp Operator* W_o.

Denoting the *perceiving* (resp. *perceived*) agent a's (resp. b's) shape centroid in that agent's referential by $\mathbf{o}_a$ (resp. $\mathbf{o}_b$), W_o is computed as follows:

$$\begin{gathered} W_o(\mathbf{s}) = \mathbf{s} + rotate_{\mathbf{t}}(\mathbf{o}_a, \theta) - \mathbf{o}_b, \\ W^{-1}_o(p) = p, \\ W^{-1}_o(\mathbf{g}) = \mathbf{g}, \\ \text{with } t = \mathbf{s} \cdot \mathbf{t} \text{ and } \theta = angle(ori_b(t), ori_a(t)), \end{gathered}$$

An example sum of two 2-by-4 rectangles at a 45 angle can be seen on the right of Figure 3.

2) Dynamics: In order to enforce agents' dynamic properties, we define a function d which, given the current speed and acceleration, bounds a target acceleration (e.g. resulting from collision avoidance) to the agent's capabilities. This function is then used in two places.

The first place is after the algorithm's solver: at timestep k, given an agent a's previous velocity $\mathbf{v}_{a,k-1} \in \mathbb{R}^2$, and the acceleration computed by the solver $\Delta \mathbf{v}_{a,k} \in \mathbb{R}^2$, the agent's new velocity is $\mathbf{v}_{a,k} = \mathbf{v}_{a,k-1} + d(\Delta \mathbf{v}_{a,k})$. This part enforces the dynamic properties of the *perceiving* agent on its own motion.

The second place is inside function v (from Section III-B) which constructs agents' likely future trajectories, thereby making sure the *perceived* agents' constraints over the *perceiving* one also comply with their dynamics.

In this paper, d is implemented based on "openxc-vehicle-simulator " from the OpenXC Platform [1].

D. Environment Constraints

Environment constraints other than layout (already dealt with in a general way in Section III-B) are rather application-specific. However, there still is one type which is fairly common, which is a constraint forcing agents to stay on a given path.

For this purpose, we model paths as biarc curves[6], that is a series of connected curves, each composed of two circle arcs. Each arc is defined by its center $\mathbf{c}_{arc}$, radius r_{arc}, half-width w_{arc}, and start and end angles $\theta_{start}, \theta_{end}$. This representation allows to easily construct paths (each section has four control points similarly to cubic Bezier curves), has C^1 smoothness, and most quantities that might be needed in further computations (e.g. distance to curve or distance along curve) are straight-forward and fast to compute. Consequently, for each arc in such constructed paths, we create a corresponding agent with the following *Warp Operator* W_a:

$$W_a(\mathbf{s}) = \begin{cases} \alpha - w_{arc}\frac{\alpha}{|\alpha|}, & \text{if } \theta_{\mathbf{s}} \in [\theta_{start}, \theta_{end}] \\ 0, & \text{otherwise} \end{cases},$$

$$W_a^{-1}(p) = \gamma p,$$

$$W_a^{-1}(\mathbf{g}) = \gamma rotate_{\mathbf{t}}(\mathbf{g}, \theta_{\mathbf{s}}),$$

where $\gamma \in [0,\ 1]$ is a tuning parameter, and:

$$\alpha = \|\boldsymbol{\beta}\| - r_{arc},$$

$$\theta_{\mathbf{s}} = angle(\mathbf{x}, \boldsymbol{\beta}),$$

$$\boldsymbol{\beta} = \mathbf{s} - (\mathbf{c}_{arc} \cdot \mathbf{x},\ \mathbf{c}_{arc} \cdot \mathbf{y},\ \mathbf{s} \cdot \mathbf{t}).$$

Using γ, it is possible to tune how much importance we allow the path. With $\gamma = 1$, the path will have as much of an impact on collision avoidance as other agents. With a lower value, a car that couldn't avoid a collision with another agent by braking could exit the lane as an alternative strategy, for instance. An example biarc-induced constraint can be seen on the left of Figure 3.

E. Finalized Chain of Warp Operators

With the changes and additions made in this section, the chain of *Warp Operators* $\mathbf{W}$ used for Equations 1 and 2 becomes:

$$\mathbf{W} = W_{th} \circ W_{tu} \circ W_{vu} \circ W_r \circ W_{sh} \circ W_o \circ W_{l2} \circ W_a \circ W_{l1}$$

IV. Results

In this section, we present some experimental results on the prototype implementation of the algorithm as described in this paper, with both simulation results and a discussion on computational costs vs. benefits.

A. Benchmarks

Since our method is meant to be general, with no distinctions between environment layout constraints, agent shapes, nor dynamics, we have designed a set of situations fused into one continuous track-like setting, where a car-like agent drives along the track and transitions between the following four cases:

Pedestrian crossing : A group of 10 pedestrians crossing the road after a right-angle turn at an intersection (Figure 4, left).

Sharp turn : A sharp right-angle turn after having reached maximum speed (Figure 4, right).

Roundabout : A roundabout in a crowd of 300 pedestrians walking with no regard for the road, and with randomly assigned destinations (Figure 5).

Open-space : An open space with two cars and two buses in a cricle (waiting for the main car to complete the circle) trying to reach antipodal positions while in a crowd of 200 pedestrians with randomly assigned destinations (Figure 6).

All cars and buses are subject to the previously mentioned dynamics [1], and have a preferred speed of 10m/s (subject to slowdowns based on interactions with other agents and environment). Pedestrians are not subject to dynamics, and have a preferred speed of 1.4m/s.

For qualitative comparison purposes, we also show some results using the publically available ORCA algorithm [24]. It is a velocity-obstacle based method, operating on holonomic, disc-shaped agents with linear motions, and we conservatively set the radius of the car and bus agents so that it contains these vehicles.

With this algorithm, the size of the car agent disc is greater than the width of the road, and therefore saturates the algorithm's solution space. Thus, we only show simulation results for ORCA on the "Open-space" part of the track (however, the first three parts of the simulation greatly rely on non-linear motions, for which the benefits of WarpDriver have already been shown in [27]).

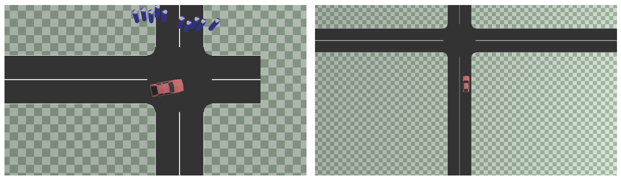

Fig. 4. Left: car stops in advance for pedestrians to cross the road. Right: car slows down before sharp turn to the right.

1) Pedestrian Crossing: In this simple situation (Figure 4, left), the car decelerates to around 2m/s when engaging on the intersection. This results from the combination of two factors. The first is the car's ability to perceive the pedestrians after the turn thanks to the non-linear knowledge of its own future path (turning and following the road), the result of which is the car's reacting to the pedestrians, although it could either turn around them or slow down. The second is the presence of path constraints related to the road the car is following, which keep the car from leaving the road, thereby slowing it down instead.

2) Sharp Turn: In this second situation (Figure 4, right), after having reached its desird speed, the car slows down to 4.5m/s before the turn. This is due to the interplay between non-linear future motion, road constraints, and the car's dynamics. The car's non-linear future motion is based on both the curve of the road and its dynamics, and at its preferred speed this motion would take it out of the road. In order to fit into the road constraints, the car slows down until its future motion coincides with the road.

3) Roundabout: In this next situation (Figure 5), the car is made to mostly disregard the pedestrians by artificially lowering the collision probabilities they impose (similar to the tuning parameter γ in Section III-D), such that most of the collision avoidance effort falls on the pedestrians. As a result, as the car approaches the roundabout, the pedestrians start moving out of the road to let the car pass. This behavior is

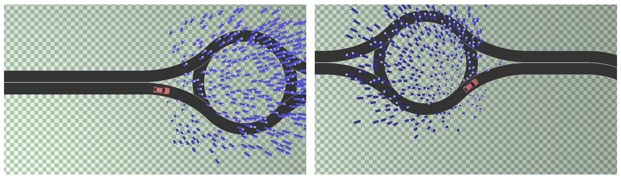
Fig. 5. Car arriving at roundabout, artificially made to mostly disregard pedestrians, which as a result clear the road ahead.

mostly due to the non-linear nature of the car's future motion, allowing the pedestrians to know where the car will pass.

4) Open Space: In this complex final situation (Figure 6), a few main observations can be made. The first is that despite the size and dynamics-related limitations of the vehicles, and the number of pedestrians, all agents stay collision-free. The pedestrians specifically, get out of the way of the vehicles and sometimes just stay on their side while they pass by, taking advantage of their shape. The second is that due to their size and lower ability to turn, the buses cross paths in the center of the circle, while cars either circumvent them or pass aferwards. Finally, the classical advantages of non-linear motion prediction also apply, with no observable "sweeping" of agents on front of the vehicles; in the case of linear prediction, agents would try avoiding collisions by steering towards the inside of the vehicles' turning maneuver even had they been safe to begin with (although note that it is difficult to observe this phenomenon in the corresponding simulation with ORCA since, having no dynamics, vehicles under ORCA change directions instantaneously).

In the ORCA simulation of this part of the scenario however (Figure 7), only one of the cars and neither of the buses manages to reach the center of the crowd, likely due to the crowd's density and the size of the car/bus agents' radii. There is also the artifact of the vehicles' spinning, though this is a relatively minor issue as bounding the acceleration using d from Section III-C2 would likely partially solve this (partially because an agent could enforce dynamic constraints on itself but not on the motion prediction applied to its neighbors).

B. Computational Cost and Analysis

The complexity of the full algorithm is $0(n^2)$ with n the number of agents, assuming each agent interacts with every other agent. A "Sweep and Prune"-like N-body culling scheme on a grid, proposed along the original WarpDriver algorithm allowed to reduce this complexity. In this scheme, each agent estimates a "hull": a conservative region of space where the collision probabilities it imposes on neighbors are lkely to be non-null (and certainly null outside of this region). Then on a 2-dimensional grid, each agent registers itself in each cell intersecting its estimated hull. As a result, agents can sample collision probabilities only for agents registered in the corresponding cells. This scheme in effect imposes a linear-time upper bound on the algorithm: as a crowd's density has an upper bound, there is an upper bound to the number of agents simultaneously registered in each cell. Furthermore, due to the conservative estimation of the hulls, simulations with and without this culling scheme give the same results. Then, with a target of 15-20 fps framerates, the algorithm could simulate upwards of 5,000 agents.

We can easily compute the overhead of the systems introduced in this paper, by looking at the differences in the computation time of agent interactions with the new *Warp Operators* enabled/disabled while using the same neghbor-selection scheme. In WarpDriver, interactions are solved in the following order: (1) commonly used values are precomputed (e.g. the $L_{a,k}$ from Section III-B), (2) constraint hulls are imprinted onto the grid, (3) the actual interactions are computed (i.e. *Warp Operators*, union of fields, solver), and finally (4) the constraint-hull grid is cleared. Table I shows this overhead as $\frac{d_i}{d_t}$, where d_i is the computation time of the considered phase of the system, and d_t is the total simulation time with all *Warp Operators* enabled. These values are then given for the four phases with the newly introduced *Warp Operators* disabled (i.e. $\mathbf{W} = W_{th} \circ W_{tu} \circ W_{vu} \circ W_r \circ W_v \circ W_{po}$, i.e. only circular agents and linear motions supported, no path constraints) and then enabled. As can be seen in this table: (1) the operations related to the constraint-hull grid have equivalent duration (expected since the same culling was performed), (2) the precomputation is noticeably longer with our complete system ($\sim$19x, expected, since this is where the $L_{a,k}$ are computed) but this duration remains negligible compared to the interaction computation, and (3) the computation of the new *Warp Operators* is only $\sim$17% more expensive. Overall, there is about 20% computational overhead over a simple, base WarpDriver algorithm supporting only disc-like agents with linear motions.

TABLE I
RELATIVE COMPUTATION TIMES ($\frac{operation\ duration}{maximum\ duration}$).

Method	Precomp.	Make hulls	Interact	Clear hulls	Total
Disabled	0.0041	0.1420	0.6117	0.0726	0.8304
Enabled	0.0785	0.1374	0.7147	0.0694	1

C. Discussion

In this work, we have addressed several key aspects central to collision avoidance: non-linear motions, agent dynamics, and agent shapes. Each of these factors has received attention in the past individually. Extensions to well-known algorithms for handling these cases, mostly based on velocity obstacles, usually come with a significant computational and implementation overhead.

For instance, even in the seemingly simple case of interactions between circular, line-like, and rectangle-shaped agents, nine shape-pair cases need to be constructed, if using an approach similar to [5]. Similarly, for algorithms that take into account agent dynamics or other sources of non-linear motion prediction, a noticeable overhead is introduced by the discretization of the velocity obstacles. Furthermore, the complexity of each resulting algorithm also makes it difficult to combine them (with algorithms or other properties, such as support for sensing error). Finally, for specific applications, the addition of corresponding features (e.g. safety distances for road traffic) or the interfacing of different algorithms also subject the resulting systems and implementations to a

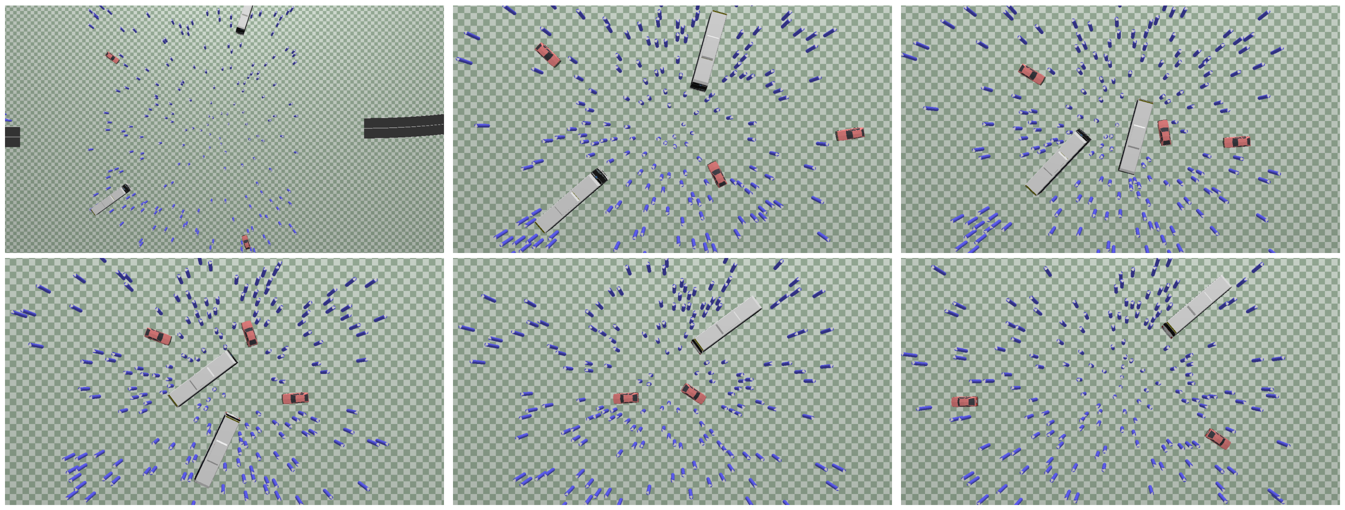

Fig. 6. Six consecutive stills from circle of buses and cars moving to antipodal positions while inside a crowd of pedestrians with random destinations. Top-left: cars and buses waiting for the main car to arrive and complete the circle.

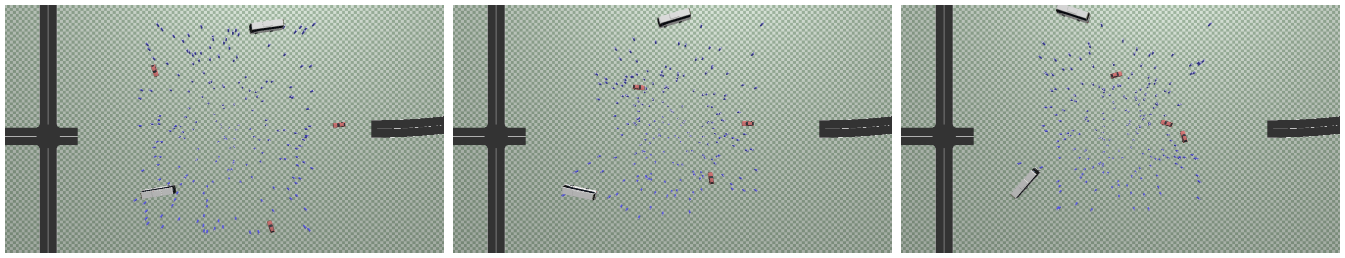

Fig. 7. Three consecutive stills at 10s intervals from the Open-space part of the simulation scenario using ORCA. Only one of the cars (the one starting from the left) manages to reach the center of the crowd, while none of the buses do.

variety of race conditions, leading to potential instability of the systems.

The Generalized WarpDriver addresses all these issues with minimal computational overhead by introducing a unified formulation applicable to all agent shapes, dynamics constraints, and non-linear motion, as shown in the benchmark examples (see the supplementary video).

V. Conclusion

In this paper, we have presented a generalization of WarpDriver, by: (1) presenting one single (though split in two for modularity), unified non-linear *Warp Operator*, instead of multiple specialized implementations and their combinatorial interaction, (2) introducing support for agent dynamics and common shapes, and (3) adding support for path constraints. Most importantly, not only do we offer support for these features individually, but also in a unified formulation. They can each be implemented in separate *Warp Operators*, but ultimately when each agent computes its next state, all these operators contribute to the construction of a single scalar field, thereby removing race conditions due to cascading of pair-wise interactions. Finally, these features can be implemented at a low computational cost, with only 20% overhead compared to a base WarpDriver implementation.

The current implementation of this work that can benefit from more optimization is the culling of neighboring agents, as the current scheme is inefficient with regard to interactions involving various shapes (difficulty of quickly constructing hulls). It would also be interesting to further compose multiple shapes into highly complex geometry, for example possibly with *Warp Operators* that branch into hierarchies of operator chains); this may help handling much more complex agent interactions (e.g. small robot moving between the wheels of a plane, while a medium robot would be limited by the fuselage and engines).

VI. Acknowledgment

This research is supported in part by the Army Research Office and Defense Advanced Research Projects Agency.

References

[1] The OpenXC Platform. http://openxcplatform.com/, https://github.com/openxc/openxc-vehicle-simulator.

[2] Javier Alonso-Mora, Andreas Breitenmoser, Paul Beardsley, and Roland Siegwart. Reciprocal collision avoidance for multiple car-like robots. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 360–366. IEEE, 2012.

[3] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Distributed Autonomous Robotic Systems*, pages 203–216. Springer, 2013.

[4] Daman Bareiss and Jur Van den Berg. Reciprocal collision avoidance for robots with linear dynamics using lqr-obstacles. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3847–3853. IEEE, 2013.

[5] Andrew Best, Sahil Narang, and Dinesh Manocha. Real-time reciprocal collision avoidance with elliptical agents. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 298–305. IEEE, 2016.

[6] KM Bolton. Biarc curves. *Computer-Aided Design*, 7 (2):89–92, 1975.

[7] Bruno Damas and José Santos-Victor. Avoiding moving obstacles: the forbidden velocity map. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4393–4398. IEEE, 2009.

[8] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.

[9] Thierry Fraichard and Hajime Asama. Inevitable collision statesa step towards safer robots? *Advanced Robotics*, 18(10):1001–1024, 2004.

[10] Oren Gal, Zvi Shiller, and Elon Rimon. Efficient and safe on-line motion planning in dynamic environments. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 88–93. IEEE, 2009.

[11] Andrew Giese, Daniel Latypov, and Nancy M Amato. Reciprocally-rotating velocity obstacles. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3234–3241. IEEE, 2014.

[12] Stephen. J. Guy, Jatin Chhugani, Changkyu Kim, Nadathur Satish, Ming Lin, Dinesh Manocha, and Pradeep Dubey. Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 177–187, New York, NY, USA, 2009. ISBN 978-1-60558-610-6. doi: 10.1145/1599470.1599494. URL http://doi.acm.org/10.1145/1599470.1599494.

[13] Mubbasir Kapadia, Shawn Singh, William Hewlett, and Petros Faloutsos. Egocentric affordance fields in pedestrian steering. pages 215–223, 2009. doi: 10.1145/1507149.1507185. URL http://doi.acm.org/10.1145/1507149.1507185.

[14] Ioannis Karamouzas, Peter Heil, Pascal Beek, and Mark H. Overmars. A predictive collision avoidance model for pedestrian simulation. In *Proceedings of the 2nd International Workshop on Motion in Games*, pages 41–52, Berlin, Heidelberg, 2009. Springer-Verlag. ISBN 978-3-642-10346-9. doi: 10.1007/978-3-642-10347-6_4. URL http://dx.doi.org/10.1007/978-3-642-10347-6_4.

[15] Ioannis Karamouzas, Brian Skinner, and Stephen J. Guy. Universal power law governing pedestrian interactions. *Phys. Rev. Lett.*, 113:238701, Dec 2014. doi: 10.1103/PhysRevLett.113.238701. URL http://link.aps.org/doi/10.1103/PhysRevLett.113.238701.

[16] Emmett Lalish and Kristi A Morgansen. Distributed reactive collision avoidance. *Autonomous Robots*, 32(3): 207–226, 2012.

[17] Eduardo Owen and Luis Montano. Motion planning in dynamic environments using the velocity space. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 2833–2838. IEEE, 2005.

[18] Sbastien Paris, Julien Pettr, and Stphane Donikian. Pedestrian reactive navigation for crowd simulation: a predictive approach. *Computer Graphics Forum*, 26 (3):665–674, 2007. ISSN 1467-8659. doi: 10.1111/j.1467-8659.2007.01090.x. URL http://dx.doi.org/10.1111/j.1467-8659.2007.01090.x.

[19] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 261–268, Sept 2009. doi: 10.1109/ICCV.2009.5459260.

[20] Julien Pettré, Jan Ondřej, Anne-Hélène Olivier, Armel Cretual, and Stéphane Donikian. Experiment-based modeling, simulation and validation of interactions between virtual walkers. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 189–198, NY, USA, 2009. ACM. ISBN 978-1-60558-610-6. doi: 10.1145/1599470.1599495. URL http://doi.acm.org/10.1145/1599470.1599495.

[21] C.W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference 1999*, pages 763–782, 1999.

[22] Zvi Shiller, Frederic Large, and Sepanta Sekhavat. Motion planning in dynamic environments: Obstacles moving along arbitrary trajectories. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 4, pages 3716–3721. IEEE, 2001.

[23] J. van den Berg, Ming Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *IEEE International Conference on Robotics and Automation*, May 2008. doi: 10.1109/ROBOT.2008.4543489.

[24] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*, pages 3–19. Springer, 2011.

[25] Jur Van Den Berg, Jamie Snape, Stephen J Guy, and Dinesh Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3475–3482. IEEE, 2011.

[26] David Wilkie, Jur Van Den Berg, and Dinesh Manocha. Generalized velocity obstacles. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 5573–5578. IEEE, 2009.

[27] David Wolinski, Ming C Lin, and Julien Pettré. Warpdriver: context-aware probabilistic motion prediction for crowd simulation. *ACM Transactions on Graphics (TOG)*, 35(6):164, 2016.