RECIPROCALLY-ROTATING VELOCITY OBSTACLES

A Thesis

by

ANDREW WEYAND GIESE

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Nancy M. Amato
Committee Members,	Suman Chakravorty
	Dylan Shell
Head of Department,	Nancy M. Amato

May 2014

Major Subject: Computer Science

ABSTRACT

Modern multi-agent systems frequently use high-level planners to extract basic paths for agents, and then rely on local collision avoidance to ensure that the agents reach their destinations without colliding with one another or dynamic obstacles. One state-of-the-art local collision avoidance technique is Optimal Reciprocal Collision Avoidance (ORCA). Despite being fast and efficient for circular-shaped agents, ORCA may deadlock when polygonal shapes are used. To address this shortcoming, we introduce Reciprocally-Rotating Velocity Obstacles (RRVO). RRVO extends ORCA by introducing a notion of rotation. This extension permits more realistic motion than ORCA for polygonally-shaped agents and does not suffer from as much deadlock. In this thesis, we present the theory of RRVO and show empirically that it does not suffer from the deadlock issue ORCA has, that it permits agents to reach goals faster, and that it has a comparable collision rate at the cost of some performance overhead.

DEDICATION

To my parents, for everything

ACKNOWLEDGEMENTS

I would like to thank my advisor, Nancy M. Amato, for mentoring me through my research here at Texas A&M. During my time under her, I've learned an incredible amount and grown significantly as a technical person. I'm humbled by the patience she displayed as I searched far and wide for a thesis topic, and beholden to her reassuring words when things didn't work as well as I'd hoped.

I am grateful to the GAMMA group at UNC Chapel Hill for helping me understand and use their open-source RVO2 library, which RRVO was built off of. Specifically, thanks to Stephen J. Guy (now at University of Minnesota), Jur van den Berg (now at University of Utah), and Ioannis Karamouzas (University of Minnesota). The RVO2 library can be found at http://gamma.cs.unc.edu/RVO2

Thank you to my co-author, Daniel Latypov, without whose help I might never have run any experiments.

I would like to thank my friends and colleagues in the Parasol lab, who tolerated my overly-excited impromptu whiteboard presentations on my research. I'd like to specifically thank Jory Denny, Aditya Mahadevan, Ali-akbar Agha-Mohammadi, and Adam Fidel, who not only acted as mentors and sounding boards in turn, but also made me feel like a friend and equal.

I would like to thank Bob Lind, my high school Computer Science teacher for showing me that Computer Science might be hard, but is also fun. Also, Jennifer Seitzer, my first AI teacher and eventual co-author, for inspiring me to go to graduate school. Finally, I'd like to thank my committee members, Dylan Shell and Suman Chakravorty, for their kind words, support, and willingness to work with me under a deadline.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

1. INTRODUCTION

Collision-free path planning is a central part of any multi-agent system and is a longstanding problem in robotics and animation. Planning a collision-free path for even a single agent in a continuous environment was found to be NP-Hard [4], and any centralized approach to planning collision-free paths for multiple agents is also PSPACE-hard [19]. Despite these theoretical hurdles, fast and efficient solutions have been designed using potential fields [25], priority-based decoupling [7], and sampling-based methods [24] [20] [27].

Crowd simulations allow us to study the behavior of crowds ranging from a few individuals to those numbering in the tens of thousands. They enable us to observe how agents interact with each other given constraints on cooperation and competition in a wide variety of realistic scenarios. Multi-agent systems have found numerous application in architectural design [35], emergency training [28], entertainment [34], urban planning [1], and more.

To satisfy interactivity requirements for crowd simulation, it is common to separate planning into high-level and low-level phases. The high-level planner usually preprocesses the environment to construct a static navigation graph (e.g., a navigation mesh [26]) that can quickly solve path queries using A* or Dijkstra's shortest path algorithms. After a desired path is extracted by the high-level planner, control is handed off to a low-level planner that is responsible for navigation decisions on a per-timestep basis. The low-level planner's role can be considered online *local collision avoidance* (LCA). LCA is responsible for deforming a trajectory generated by a high-level planner in order to avoid collision with unforeseen obstacles. LCA is usually performed in each *sense-plan-act* cycle, whereas high level planning is performed

periodically.

Recently, decoupled methods that anticipate the positions of obstacles over a small time window have gained traction [8]. These *Velocity Obstacle* (VO) variations are able to efficiently simulate agent movement for up to thousands of agents, and are amenable to parallelization. Most Velocity Obstacle techniques assume disc-shaped robots translating in a plane. This representation may be unsuitable for some agents, e.g., a bus, or one may wish to use a larger variety of shapes to model finer interactions between agents. Another important weakness of VO methods is that they restrict motion to translation, which is sufficient when circular agents are used because a circle is rotation invariant. However, using the bounding circle of an agent or restricting it to translation alone may cause some problems to become unsolvable, as in the example shown in Figure 1.1.

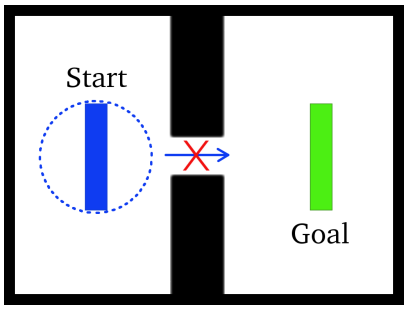


Figure 1.1: A long skinny agent (blue rectangle) cannot reach its goal (green), if represented as its bounding circle (blue dotted circle) or if only translational movement is permitted.

In this work, propose *Reciprocally-Rotating Velocity Obstacles* (RRVO) to address the inadequacies of using translating discs to represent agents for local collision avoidance. RRVO represents agents as convex rotating polygons, and in each *sense-plan-act* cycle, an agent chooses a new translational velocity *and* orientation. Agents

avoid colliding with each other while rotating by assuming their neighbors may rotate as much as themselves. By choosing high clearance rotations, RRVO often breaks the symmetries that cause deadlock in ORCA.

In ORCA, an agent's neighbors induce constraints on collision-free velocities for the next time step. These constraints form the basis of a linear program that is solved to find a collision-free velocity that is optimized to be nearest to some preferred velocity. In RRVO, each collision-free orientation is associated with a linear program. Collision-free orientations are discovered by discretizing the interval through which an agent may rotate in the next time frame, and then applying a notion of reciprocity where we account for potential orientations of neighboring agents. The naïve RRVO algorithm solves all linear programs and arbitrates over the resulting collision-avoiding velocities to choose a preferred one. In our implementation, we discard most linear programs by minimizing the distance from an agent's current velocity to the feasible region. By utilizing the space more efficiently via rotation, we empirically show that RRVO agents deadlock less frequently than ORCA ones. The cost of accounting for ones own orientations, as well as the orientations of each neighbor, however, incurs a theoretical performance penalty that is quadratic in the granularity at which we discretize these sets of orientations.

Our specific contributions include:

- Reciprocally-Rotating Velocity Obstacles (RRVO) theory,
- Analysis of time complexity, and
- An empirical study that shows how RRVO results in less deadlock, faster completion, and comparable collision rates to ORCA in exchange for some computational overhead.

A version of this work [11] with preliminary results has been accepted to appear at the *IEEE International Conference on Robotics and Automation (ICRA)* in May/June 2014.

1.1 Thesis Organization

This thesis is organized as follows. Section 2 describes related work for local collision avoidance in multi-agent systems. Section 3 more formally defines the multi-agent collision avoidance problem as it pertains to polygonal agents. Section 4 introduces the theory of Reciprocally-Rotating Velocity Obstacles and reviews its foundations. In Section 5 we detail our experimental approach of RRVO, and compare it to ORCA in terms of overhead and deadlock resolution. Finally, we summarize and conclude the work in Section 6.

2. RELATED WORK

Local collision avoidance has been studied extensively. In this section, we separate different approaches to LCA into four different models: reactive, predictive, cellular automata, and cognitive (rule-based).

2.1 Reactive Models

Local collision avoidance was popularized largely by Reynolds' seminal work where agents in a flock were attracted to either a global or local flock center, but repulsed from nearby neighbors [32]. The flock as a whole was given a goal, called a "migratory urge", and individual members attempted to match velocities with their neighbors. The result was realistic bird-like flocking that didn't need to be explicitly scripted by an animator. Avoidance of static obstacles was performed geometrically by finding the nearest point on each obstacle and assigning a repulsive force from the surface normal.

Helbing expanded Reynold's ideas to include general social forces acting as attractive and repulsive impulses [17]. Personal space was represented with a radius about an agent's center, and a repulsive force was assigned to any other agent that entered this sphere. Agents grouped together as families or friends remained coherent via an attractive force assigned they mutually assigned each other.

Reif and Wang independently developed a social forces model in [31]. Although their work shares many similarities to Helbing's, it was actually inspired by Khatib's seminal work [25] on potential fields. Reif and Wang make an argument for spring laws to allow flocks to assemble into predefined formations, analyze the stability and convergence of their fields, and design a number of hierarchical potential laws to achieve specific desired behaviors other than generic flocking or single-point queries.

Collision avoidance is still handled through locally assigning repulsive forces to the centers of nearby agents and obstacles.

The same year as Reif and Wang's paper, Reynolds presented work that argued for a library of primitive steering behaviors that could be combined to create realistic pursuit-evasion, wander, path following, leader following, and cohesion behaviors [33]. Combining the higher level behaviors in a priority-based architecture would allow one to create realistic agents capable of a variety of tasks. Obstacle avoidance is handled via its own primitive steering behavior where a cylinder extends outward from an agent's forward axis as a sort of probe to test for future collisions. When the cylinder intersects an obstacle or other agent, a repulsive force is assigned to the point of intersection. This cylinder grows and shrinks dynamically with the agent's speed.

In [10], Gayle et al. used ideas from both [18] and [31] to allow for collision-free translational and rotational motion between arbitrary polyhedra. This was achieved by sampling points across the surfaces of the polyhedra, and then combining social forces on those individual points to provide acceleration and torque. All forces acting on the agents were incorporated as physical constraints for a physics-based motion planner described in [9]. Our work also allows for rotational motion, but does so without using potential fields to impart torque; instead, we use a geometric approach to create linear programs which agents explicitly use to select desirable orientations.

2.2 Predictive Models

Predictive models anticipate future collisions so that agents can take steps to avoid them. Often this means linearly extrapolating neighbor velocities, and then avoiding the future locations of those neighbors.

Our work most closely follows that of Reciprocal Velocity Obstacles (RVO), first

presented by van den Berg *et al.* in [43] and then renamed to Optimal Reciprocal n-Body Collision Avoidance (ORCA) in [41]. RVO and ORCA predict the set of collision-causing velocities for each agent by assuming linear trajectories, and then choose a velocity outside that set. The main difference between RVO and ORCA is that ORCA solves a linear program whereas RVO searches via sampling.

RVO has been the focus of much research, and there are many variations and optimizations in the literature. None of them allow agents to rotate, and instead concern themselves almost exclusively with choosing a better collision-free translational velocity.

In [36], the authors were interested in modelling arrival and departure behaviors around shared resources. Departing agents use RVO normally, but arriving agents use a slightly modified version to change their preferred velocities so that they defer to the departing agents. In [12], the authors assign a confidence level to predicted future positions that drops off linearly with time. Agents prioritize avoiding positions they are more confident about. He and van den Berg developed an ORCA-variant that considers groups of agents as single entities for meso-scale collision avoidance [16]. In [13], RVO is used to compute the set of collision-free velocities, from which agents choose those that minimize biomechanical energy expenditure using the principle of least effort. In [14], the authors sought to parallelize RVO computations on SIMD, shared-memory, multi-core machines. Yeh *et al.* developed a system where agents signal intent by placing proxy agents at locations they wished others to avoid [44]. Finally, in [15], the authors used a psychological model to vary the parameters of RVO to create more realistic simulations.

In [23], the authors used Helbing's social forces model with a twist. Instead of agents simply reacting to the presence of other agents inside their personal space, agents integrate their own trajectories and linearly extrapolate the trajectories of

other agents to predict future collisions. For each future collision, an evasive force is applied that drops off exponentially with distance. Evasive force is thresholded to avoid jerky motion. The result is that agents take paths that expend less energy than agents using a pure social forces approach.

In [39], the authors employed a mixture of long, mid, and short-term planning coupled with a collision prediction model as well as reactive collision avoidance to achieve realistic motion for thousands of agents in real-time. The authors experimented with adapting the frequencies at which the passive perception, collision prediction, and reactive behaviors would run, enabling them to use fewer resources while achieving the same effect. The difference between our work and theirs is that their reactive collision avoidance model involved casting three rays out from the agent to detect nearby obstacles and agents, and using a rule-based system to determine an action based on the object's relative position.

Sometimes it may be preferable to not make any assumptions about an agent following a predictable trajectory. Egocentric Affordance Fields rates areas around other agents as threatening based on their proximity and relative velocity [22]. Threatening areas create repulsive potential fields about the agent. Other potential fields are constructed in concentric rings about the agent, and the resulting motion is a result of the summed forces acting upon it at any point in time.

2.3 Cellular Automata

When crowd density becomes exceptionally high, modelling agents as particles or incompressible fluids has become an attractive notion. Largely based on the work of Hughes [21] and Chenney [5], these approaches discretize the environment into a grid of varying coarseness, and assign velocity fields that direct agent motion in individual cells. Due to the nature of cellular decomposition, agents cannot theoretically collide

and thus local collision avoidance is rarely needed.

Treuille *et al.*'s Continuum Crowds maps agents to grid cells and generates potential fields based on how much weight an agent assigns path length, time, and congestion [40]. Computing an agent's path is done by integrating the sum of potentials acting on it. Agent speed varies depending on crowd density and terrain slope. Because the grid may be relatively coarse, agents in the same grid cell may occasionally intersect, so the authors iterate over all pairs of agents and enforce a minimum separation distance. While such an approach works well in practice it is not guaranteed to satisfy the minimum distance constraint, as moving one agent affects its distance to all others.

Narain *et al.* took a similar approach, but added a notion of incompressibility where agents in areas of especially high density are more affected by the overall average crowd velocity of that area [29]. Collision avoidance is again handled by enforcing a minimum pairwise distance between all agents.

While cellular automata approaches are extensively used for their ability to simulate extremely large crowds, they make many simplifying assumptions about individual agents. In particular, they usually do not account for agent-specific physical or mental properties such as visual occlusion or variable environmental knowledge. Instead, they are applicable for simulating large groups of homogeneous agents.

2.4 Cognitive Models

Cognitive models attempt to incorporate human psychology into overall crowd behavior. The variables representing each agent's mental state affect the goals the agents intend to reach, and indirectly the routes the agents will traverse.

Shao and Terzopoulos constructed a system where agents sense ground height and static and dynamic obstacles [38]. Agents are equipped with a small library of

reactive rule-driven behaviors to avoid collision. By using a hierarchy of variably-detailed maps to solve planning and pathing queries, the authors were able to achieve real-time performance for many agents in a large environment. The maps range from a simple topological graph indicating connections between rooms to a fine grid decomposition to be used for reactive collision avoidance.

In [30], the authors combined psychological, social forces, and geometrical rules into a system meant for more realistic simulation of very large crowds that would not have the downsides of a cellular automata approach. Collision avoidance is handled with a mixture of social forces and geometric rule-based behaviors. The authors successfully addressed the "shaking" problem that can appear in simulations where agents appear to vibrate in high density situations due to repulsive forces on all sides. The resulting system successfully recreates a variety of phenomena seen in large crowds, such as panic propagation, bi-directional flow, pushing, and queuing.

Cognitive models generally result in more realistic motion among agents, but one weakness is that they typically rely on rule-driven collision avoidance behaviors, which may require a great deal of hand-tuning.

3. PROBLEM DEFINITION

In this section , we describe the collision avoidance problem for multi-agent systems that is addressed in this thesis. For the following definitions, we will restrict ourselves to a two-dimensional environment and assume there are no nonholonomic constraints on movement.

Let there be a set of n agents A such that each agent $a_i \in A$ can be represented with a position $a_{i_{pos}}$ and an orientation $a_{i_\theta} \in (-\pi, \pi]$. The agent's geometry, a_{i_P}, is a convex polygon consisting of the vertices $\{a_{i_{P_1}}, a_{i_{P_2}}, \cdots, a_{i_{P_m}}\}$ centered about $a_{i_{pos}}$ and rotated about the vertical axis by a_{i_θ}. Agent a_i additionally has a velocity a_{i_v}, a maximum translational speed $a_{i_{s_{max}}}$, and a maximum angular velocity $a_{i_{\omega_{max}}}$. Finally, the agent has a preferred translational velocity $a_{i_{v_{pref}}}$ and orientation $a_{i_{\theta_{pref}}}$.

Problem 1. *(Collision Avoidance) Each agent $a_i \in A$ must choose a new collision-free velocity and orientation at each timestep of the simulation such that is valid for a time interval of length $\geq \tau$.*

Ideally, the new velocity $a_{i_{v_{new}}}$ and orientation $a_{i_{\theta_{new}}}$ are as close as possible to the agent's preferred velocity and orientation, respectively. LCA techniques such as ORCA and RRVO are not concerned with choosing $a_{i_{v_{pref}}}$ or $a_{i_{\theta_{pref}}}$ and assume that both are provided by the high-level planner. For example, $a_{i_{v_{pref}}} = \left(a_{i_{goal}} - a_{i_{pos}}\right)$ and $a_{i_{\theta_{pref}}} = \text{ATAN2}(a_{i_{v_{pref}}})$, where ATAN2 is the signed arctangent function. We assume agents do not explicitly coordinate to select new orientations and velocities.

4. RECIPROCALLY-ROTATING VELOCITY OBSTACLES

In this section we introduce Reciprocally-Rotating Velocity Obstacles (RRVO), our solution to the multi-agent collision avoidance problem where non-circular agents assume that their neighbors are equally capable of rotating as much as they are each timestep. First we provide some background on velocity obstacles, including their construction and the notion of reciprocity between agents. Next, we introduce the idea of reciprocal rotation. Finally, we address the theoretical complexity of RRVO before discussing how RRVO handles obstacles and resolves collisions.

4.1 Reciprocal Velocity Obstacles

A velocity obstacle [8] is defined as the set of all robot velocities that will cause a collision. In its original formulation, agents assume others continue moving along a linear trajectory. In [41], this assumption is different in that agents assume others will bear half the responsibility of avoiding a collision (reciprocity).

The high-level view of the algorithm for Optimal Reciprocal Collision Avoidance (ORCA) is displayed in Algorithm 1. In ORCA, each agent transforms the positions and velocities of its nearest neighbors into linear constraints on its own chosen velocity. Solving the resulting linear program yields a collision-free (or nearly so) new velocity. The resulting linear program may be infeasible, in which case the constraints are relaxed until exactly one velocity is feasible. A feasible linear program is guaranteed to be collision-free provided that other agents employ the same algorithm. In an infeasible linear program, the chosen velocity is that which violates constraints the least, and is thus the most likely collision-free velocity attainable.

Algorithm 1 Compute new velocity for agent a_i

Input: $neighbors_{a_i} \subset A$
Output: $a_{i_{v_{new}}}$

1: **for all** $neighbor_j \in neighbors_{a_i}$ **do**
2: Add a linear constraint on $a_{i_{v_{new}}}$
3: Solve linear program to find $a_{i_{v_{new}}}$

4.1.1 Construction

Algorithm 1 solves a linear program to discover a new collision-avoiding velocity for a_i. At the core of the algorithm is the creation of each linear constraint based on the positions and velocities of a_i's neighbors. This involves first building a Velocity Obstacle, which is a geometric region in velocity space denoting the set of agent velocities that are *not* guaranteed to be collision-free.

Given two agents a_1 and a_2, a_1 will create a velocity obstacle representing a_2 (and vice-versa) such that a_1 wishes to choose a guaranteed collision-free velocity for the time interval τ. We denote this velocity obstacle representing a_2 as $VO^{\tau}_{a_1|a_2}$. The computation of $VO^{\tau}_{a_1|a_2}$ is shown in Algorithm 2.

Geometrically, $VO^{\tau}_{a_1|a_2}$ is an unbounded polygon such that:

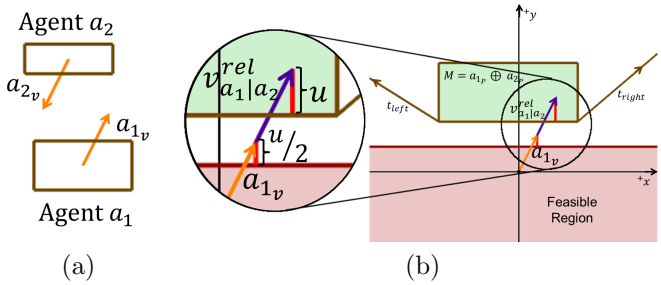
- It contains the Minkowski sum of a_1 and a_2's geometry, $M = a_{1_P} \oplus a_{2_P}$, where $\oplus$ is the Minkowski sum operator,
- it is bounded by at least one line segment on M,
- it is bounded on two sides by the tangent lines on M through the origin, and
- it is otherwise unbounded.

Figures 4.1(a) and (b) show an example of this construction.

Algorithm 2 Compute velocity obstacle induced by a_2 on a_1

Input: Agents a_1 and a_2, time horizon τ
Output: $VO^{\tau}_{a_1|a_2}$

1: $M \leftarrow a_{1_P} \oplus a_{2_P}$ //Minkowski Sum
2: TRANSLATE$\left(M, \frac{a_{2_{pos}}(1-\tau) - a_{1_{pos}}(1+\tau)}{\tau}\right)$
3: SCALE$(M, \frac{1}{\tau})$
4: $(t_{left}, t_{right}) \leftarrow$ COMPUTETANGENTS$(M, 0)$
5: **for all** $m_i \in M$ **do**
6: **if** $((t_{right} - t_{left}) \times (m_i - t_{left})) \leq 0$ **then**
7: $VO^{\tau}_{a_1|a_2} = VO^{\tau}_{a_1|a_2} \cup \{m_i\}$
8: //Represent unbounded sides with tangent vectors
9: $VO^{\tau}_{a_1|a_2}.left\_leg \leftarrow 2t_{left}$
10: $VO^{\tau}_{a_1|a_2}.right\_leg \leftarrow 2t_{right}$



(a) (b)

Figure 4.1: **(a)** Two rectangular robots a_1 and a_2 on a collision course. **(b)** Construction of $VO^{\tau}_{a_1|a_2}$ for the scenario shown in (a). A linear constraint (pink region) on a_{1_v} is derived from the velocity obstacle, and it can be seen that a_{1_v} does not lie in the feasible region.

The translation applied to M on line 2 of Algorithm 2 is actually the combination of three different translations. Computing the Minkowski sum M of a_1 and a_2 and translating it to $a_{2_{pos}}$ allows us to discard a_1's geometry and only consider it as a point robot defined by $a_{1_{pos}}$. To get an absolute frame of reference, we consider a_1 at the origin (egocentric coordinates), so we translate M by $-a_{1_{pos}}$. Furthermore, we only require that a_1 chooses a velocity that is valid for a given time interval τ, so we scale M and its position by $\frac{1}{\tau}$ (line 3). This has the effect of 'dragging' the

Minkowski sum nearer to the origin to simulate future timesteps while maintaining the same tangent lines.

ComputeTangents() computes t_{left} and t_{right}, which are the tangent points on M relative to the origin. They act as endpoints to rays in the direction of the tangent lines that help bound $VO^{\tau}_{a_1|a_2}$. Lines 5-7 use these endpoints to compute the line segment(s) on M that bound $VO^{\tau}_{a_1|a_2}$.

4.1.2 Geometric Linear Programming

Note that $VO^{\tau}_{a_1|a_2}$ is a constraint on the *relative* velocity of a_1 and a_2, defined as:

Definition 1. *(Relative Velocity)* $v^{rel}_{a_1|a_2} = a_{1_v} - a_{2_v}$

In this section we will transform $VO^{\tau}_{a_1|a_2}$ from a constraint on $v^{rel}_{a_1|a_2}$ into a linear constraint on a_{1_v}, which agent a_1 can use to choose its new velocity v_{new_a} for the next timestep.

Any relative velocity $v^{rel}_{a_1|a_2}$ inside $VO^{\tau}_{a_1|a_2}$ will violate our guarantee of collision-free movement for time τ. Finding p_{near}, the nearest point on $VO^{\tau}_{a_1|a_2}$ to $v^{rel}_{a_1|a_2}$, allows us to compute the minimum amount that $v^{rel}_{a_1|a_2}$ must change (u):

Definition 2. *(Minimal Velocity Change)* $u = p_{near} - v^{rel}_{a_1|a_2}$

When testing obstacle-agent collisions, full responsibility is on the agent to affect $v^{rel}_{a_1|a_2}$, so a_{1_v} must change by at least u to avoid collisions. However, for agent-agent collisions, each agent a_1 and a_2 assumes half the responsibility in affecting $v^{rel}_{a_1|a_2}$, so the minimum amount that a_{1_v} and a_{2_v} must change is $\frac{u}{2}$. In [41], they prove this formulation still results in collision-free motion.

$a_{1_v} + u$ is a vector facing in the direction that a_{1_v} must change for collision avoidance. We can represent the entire set of admissible velocities as a geometric half plane bounded by the line perpendicular to $a_{1_v} + u$. This half-plane is a linear

constraint on the agent's next chosen velocity where the feasible region lies in the direction u from a point on the bounding line. An example linear constraint is shown in Figure 4.1(b).

Each velocity obstacle for agent a_1 induces a new linear constraint on a_1's chosen velocity. Consequently, solving for v_{new_a} involves solving a two-dimensional linear program, which can be done in $O(n)$ randomized expected time where n is the number of constraints [37].

When crowd density becomes high, it is possible that a solution to the linear programming problem does not exist, which will cause the solver to fail. Because the agent still needs to decide upon a velocity, the two-dimensional linear program is transformed into a three-dimensional one where the infeasible velocity that minimizes the distance to its nearest half-plane is chosen. This velocity can be thought of as the velocity that violates constraints the least [41].

4.2 Reciprocal Rotation

Using velocity obstacles to derive collision-free velocities works well when agents are represented as discs; when agents become near, so do the tangent points they compute on the dilated disc representing the Minkowski sum between their geometries. That is, $v^{rel}_{a_1|a_2}$ almost always eventually projects onto a tangent line, allowing agents to move around each other, as in Figure 4.2(a).

When agents are polygonal, though, this construction has a serious flaw. When two polygonal agents interact, there is no guarantee that the tangent points on $a_{1_P} \oplus a_{2_P}$ grow nearer as the agents do, which can cause deadlock, as shown in Figure 4.2(b). In Reciprocally-Rotating Velocity Obstacles, we reduce the possibility of deadlock between two agents by explicitly disallowing this scenario. When we also allow agents to rotate, we can greatly reduce the amount of deadlock.

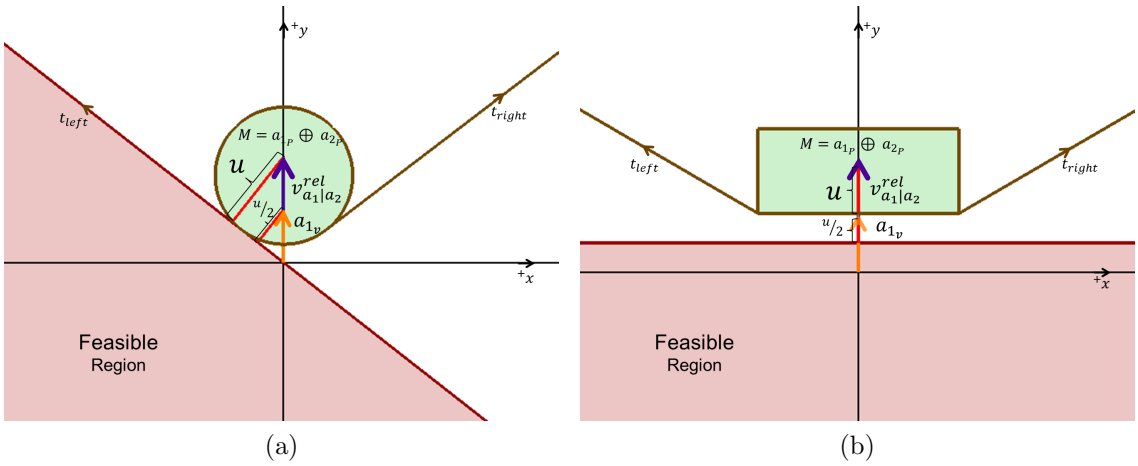


Figure 4.2: **(a)** When two circular agents with opposing velocities meet, eventually they are instructed to choose lateral velocities. **(b)** When two rectangular agents with opposing velocities meet, they may never choose a lateral velocity.

When we permit polygonal agents a_1 and a_2 to rotate, the shape of the velocity obstacles they induce on each other will change. As shown in Figure 4.3, as agents actively rotate, they can more easily utilize available space to move around one another.

In Reciprocally-Rotating Velocity Obstacles, agents assume that others will rotate *reciprocally*. That is, in RRVO, agents assume that others may rotate equally (or equally opposite). When all agents make this assumption, they can intelligently choose collision-free orientations. RRVO easily handles rotating agents, and considers convex obstacles as special cases of (convex) agents. Therefore, we consider RRVO to be an extension and generalization of ORCA.

4.2.1 Method

In this subsection, we present Reciprocally-Rotating Velocity Obstacle theory, from deciding which neighboring agents must be considered to how we use the notion of reciprocal rotation to choose a collision-free orientation.

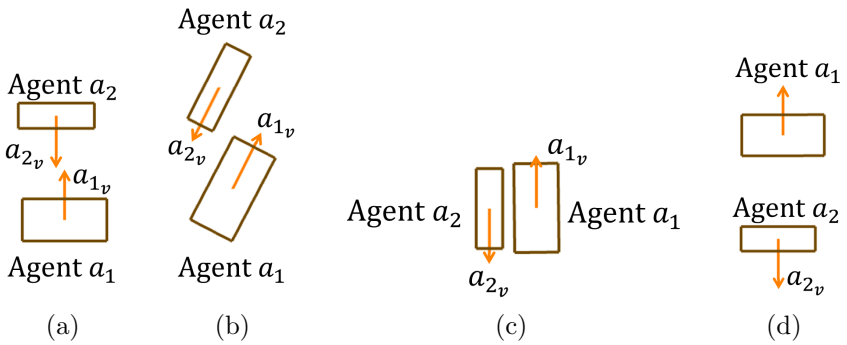


Figure 4.3: When two agents encounter a potential deadlocking situation, they may rotate to better maneuver about one another.

The idea behind RRVO is to assume a maximum amount of rotation by nearby neighbors, and then compute approximated swept areas they may rotate through. From these swept areas, we may create Velocity Obstacles, the boundaries of which can be transformed into linear constraints on velocity. An overview of the method is presented in Algorithm 3. The approximation of the swept area and the creation of the linear constraints are handled simultaneously by rotating each (convex) neighbor by a small amount and using the methodology of ORCA to create a linear constraint for that orientation. Later in this section we will more fully explain what we mean by "reachable orientations".

Algorithm 3 Compute new velocity and orientation for agent a_i

Input: $neighbors_{a_i} \subset A$, $a_{i_{\omega_{max}}}$
Output: $a_{i_{v_{new}}}, a_{i_{\theta_{new}}}$

$LP \leftarrow$ a set of linear programs
for all Orientations reachable by a_i **do**
  for all $neighbor_j \in neighbors_{a_i}$ **do**
    for all Orientations reachable by $neighbor_j$ **do**
      $LP[i] = LP[i] \cup$ linear constraint on $a_{i_{v_{new}}}$
Solve linear programs in LP and choose desired $\left(a_{i_{v_{new}}}, a_{i_{\theta_{new}}}\right)$

Not every agent in the environment needs to be considered as a reciprocally-rotating neighbor. In fact, if we could observe the maximum speeds of other agents, we could compute the set of neighbors that must be considered when rotating to guarantee collision-free rotation by using their bounding radii.

Definition 3. *(Bounding Radius) The bounding radius of agent a_i, $a_{i_{rad}}$, is the maximal Euclidean distance from $a_{i_{pos}}$ to some point $a_{i_{P_j}} \in a_{i_P}$.*

Definition 4. *(Rotation Neighbors) The rotation neighbors for agent $a_i \in A$ for time interval τ are defined by the set $N_{rot_{a_i}} = \{\forall a_i \in A \mid a_j \neq a_i, ||a_j - a_i|| - \tau(a_{i_{s_{max}}} + a_{j_{s_{max}}}) < (a_{i_{rad}} + a_{j_{rad}})\}$*

Rotation neighbors for polygonal agents can be visualized by returning to the notion of disc-shaped agents, such that every agent's disc has radius equal to the distance from the agent's center to the farthest point on its polygonal boundary. For any agent, its rotation neighbors consist of those whose discs (could) overlap its own within τ.

Given a time interval of duration τ, every agent $a_i \in A$ has a set of reachable orientations $a_{i_{\theta_{reach}}} \subseteq (-\pi, \pi]$ from which it will choose $a_{i_{\theta_{new}}}$.

Definition 5. *(Reachable Orientations)* $a_{i_{\theta_{reach}}} = \{a_{i_\theta} - \left(\tau a_{i_{\omega_{max}}}\right), a_{i_\theta} + \left(\tau a_{i_{\omega_{max}}}\right)\} \setminus \{\forall \theta \mid \exists a_j \in N_{rot_{a_i}}, a_i \neq a_j, a_{i_{pos}} \in a_{i_P} \oplus a_{j_P}\}$

Implicit in Definition 5 is that $\forall a_j \in A$ $(a_i \neq a_j)$, a_j is rotated by the same $\Delta\theta$ (or $-\Delta\theta$) as a_i. That is, a_i cannot choose a change in orientation if an equal (or equally opposite) change in a_{j_θ} would cause a collision. Also not stated is that the Minkowski sum between a_i and a_j's geometries, $a_{i_P} \oplus a_{j_P}$, is centered at $a_{j_{pos}}$.

RRVO approximates the bounds on $a_{i_{\theta_{reach}}}$ by discretizing the set of rotations by the constant δ. For each orientation of a_i, we check for collision against δ neighbor

orientations, for all neighbors. If there is a collision, then $a_{i_{\theta_{reach}}}$ is approximately bounded in that direction (clockwise or counter-clockwise).

After $a_{i_{\theta_{reach}}}$ is computed, a desired orientation must be selected from the interval. If the chosen $a_{i_{\theta_{new}}}$ is not reachable in the next frame, then a_j simply rotates at its maximum speed for that timestep (this can happen if τ is on the order of a few seconds).

Given the above information, Algorithm 3 can then be summarized as follows:

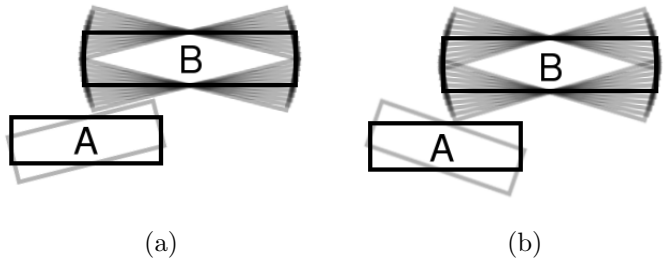
- Agents assume that the reachable change in their own orientation is the same reachable change by any neighbor.
- If a rotation causes collision with any neighbors' set of potential orientations, no more linear programs are created because the set of reachable orientations has been bounded (see Figure 4.4); linear programs are only created for (approximately) collision-free orientations.
- To choose a new orientation and velocity, all linear programs are solved and some function is applied on the resulting choices to choose an optimal one.

In our implementation, we further optimized the method by discarding most linear programs. We only kept those where we considered the feasible region to be maximized. To understand the specifics of how this is performed, recall that each constraint of a linear program is created by finding u (Definition 2), the vector representing the distance from $v^{rel}_{a_1|a_2}$ to $VO^{\tau}_{a_1|a_2}$. Geometrically, we developed the notion of a *signed magnitude* for the vector u. We consider u's magnitude as negative if $VO^{\tau}_{a_1|a_2}$ is contained within $VO^{\tau}_{a_1|a_2}$, and positive otherwise. More formally, u's magnitude is negative if the angle between it and a vector pointing toward the center of $VO^{\tau}_{a_1|a_2}$ is greater than $\frac{\pi}{2}$. In any linear program, there will be one or more

constraint that has minimal magnitude of u. We associate this minimum value of u with each linear program and only keep those linear programs that maximize it.

After solving the remaining linear programs, we ranked the potential new (orientation, velocity) pairs by the following criteria (in order of precedence):

1. Linear program feasibility (favor linear programs that don't relax constraints)
2. Minimization of $|\text{ATAN2}(a_{i_v})| - |a_{i_\theta}|$ (favor those that allow agents to face their velocity vectors)
3. Minimization of $|a_{i_v} - a_{i_{v_{pref}}}|$ (favor those that allow agents to follow their preferred velocities)



(a) (b)

Figure 4.4: **(a)** Agent A, represented as a black rectangle, bounds its maximum counter-clockwise rotation by assuming another agent, B may rotate at most as much as A. A discovers that a rotation of about 14° (light gray) is the maximum it can rotate such that it doesn't intersect the swept volume of B through ±14° (gray). **(b)** Agent A is able to rotate farther in a clockwise direction before an identical rotation from agent B would cause a collision.

Choosing a new velocity that satisfies all constraints should lead to collision-free rotation and translation. However, because we have discretized the set of orientations instead of computing an actual swept area, our approach is an approximation that improves as δ increases. The approximation may be overly optimistic for low values

of δ, which could lead to a choice of velocity and orientation that may cause a collision in the transition to $a_{i_{\theta_{new}}}$. As we show in our results though, even low values of δ (< 10) lead to very few collisions. Another caveat of our algorithm is that we are forced to approximate the set of rotation neighbors by using a conservative nearest neighbor check because we do not allow agents to deduce maximum speeds of other agents.

4.2.2 Obstacles

RRVO can be extended to work with static obstacles. Assuming that obstacles are also represented using convex polygons, agents may treat them as motionless agents. Accounting for obstacles, therefore, is simpler and more efficient than accounting for other agents. However, RRVO is only concerned with collision avoidance, and is like ORCA in that it is not guaranteed to direct the agent around static obstacles; a high-level planner should take that responsibility.

4.2.3 Collisions

A feasible solution to the linear program does not always exist. In this case, the agent may find itself in collision at the next timestep. When an agent is in collision, a velocity obstacle cannot be constructed because no tangent lines exist. In this case, we instead choose to construct a linear constraint based on the distance from $a_{1_{pos}}$ to the nearest point on M scaled by the timestep duration. Such a constraint encourages agents to choose velocities that escape collisions. When an agent is in collision, RRVO allows them to rotate through the entire set of orientations without regard to collision in an additional effort to resolve collision via rotation.

4.2.4 Time Complexity

In ORCA, finding one's k-nearest neighbors is performed in $O(k \log n)$ time on average when using a k-d tree, which can be constructed in $O(n \log n)$ time [2]. Velocity obstacle creation for each agent is a constant time operation with circles, and solving the geometric linear program involves satisfying k constraints, one for each neighbor. As mentioned in Section 4.1.1, solving such a two or three-dimensional linear program can be done in $O(k)$ time. Every agent must perform nearest neighbor search and solve a linear program, so the total time complexity of ORCA is $O(nk \log n + nk) = O(nk \log n)$.

Reciprocally-Rotating Velocity Obstacles performs the same operations at each step except for the computation of velocity obstacles. Computing a velocity obstacle for two convex polygonal agents a_1 and a_2 requires the Minkowski sum of a_{1_P} and a_{2_P}, as well as computation of tangents. Assuming a suitable representation of a polygon (e.g., a vertex list topologically sorted counter-clockwise about the polygon's centroid), computing the boundary of the Minkowski sum can be done in $O(||a_{1_P}|| + ||a_{2_P}||)$ using the convolution method, and finding the tangents is at worst $O(n)$ in the number vertices of the Minkowski sum [6] (by construction of the Minkowski sum, we retain the counter-clockwise sorting of the vertices). Additionally, for each Δa_{i_θ} for which we create a linear program, each neighbor adds δ constraints to the linear program due to of our discretization of the rotation interval. Therefore we must compute $O(\delta^2)$ velocity obstacles for each neighbor. The theoretical complexity RRVO incurs is therefore $O(\delta^2 \textsc{argmax}(||a_{i_P}||))$. Realistically, a comparable implementation of ORCA to handle polygons must also take on the additional complexity involved in computing Minkowski sums, so we can consider RRVO's additional complexity to be on the order of δ^2.

5. EXPERIMENTAL RESULTS

In this section, we experimentally evaluate Reciprocally-Rotating Velocity Obstacles. We compare it against ORCA in terms of computational performance, deadlock resolution, and collision frequency. We show that RRVO suffers from less deadlock, scales according to our analysis of time complexity, and experiences a comparable collision rate as ORCA. As the shapes used to approximate agents approach a bounding circle, though, the ORCA and RRVO behave the same.

5.1 Metrics

We measure deadlock in RRVO versus in ORCA by first tracking the percentage of agents at their goal at simulation's end. If there is less deadlock, then more agents will reach their goals. Some agents will be totally deadlocked, and therefore will not complete. We track only those agents at their goal positions at simulation end instead of the number of agents that reached their goals once because it is very common for an agent to get pushed off its goal position, and then subsequently deadlock with another agent and never reach its goal again.

Next, we measured the number of frames it took each agent to arrive at its goal. In this case, we track frames for agents that reach their goal once, and then only for the number of frames it took for them to reach it that first time. That is, if an agent arrives at its goal, is pushed off it, and then arrives again, we only consider the number of frames from beginning to the first goal arrival.

We computed the average frame rate (FPS) normalized by the frame rate of ORCA to show the overhead that RRVO introduces. This measurement is intended to show how RRVO scales with higher values for δ, and to validate our analysis of complexity.

Finally, we tracked the mean number of collisions agents experienced throughout the simulation. Because RRVO is only approximately collision-free in scenarios where ORCA is guaranteed to be collision-free, this measurement is intended to show the reader how much more collisions to expect when using RRVO.

We only tracked the number of frames and the average collisions for those agents that actually reached their goals because otherwise the results would be unfairly skewed in RRVO's favor due to the fact that it experiences less deadlock than ORCA. Deadlocking agents tend to collide much more than non-deadlocking agents and skew the average number of frames it takes for agents to reach their goals.

5.2 Experimental Setup

We implemented RRVO in C++ by adapting the RVO2 library [42]. The library has existing support for coarse OpenMP parallelization which we retained, although it by no means represents the limits of RRVO's scalability. Timing experiments were run on a quad-core Intel i5-2520M system with 4 GB of memory running Ubuntu 12.04, and we used Performance Application Programming Interface (PAPI) high performance timers.

For our first round of experiments, we explicitly compare RRVO against ORCA for all metrics. Agents are modelled as rectangles with a length-to-width ratio of 1.83, which is intended to model the shoulder-to-chest depth ratio of the average human [3]. We vary the value of δ to observe how finer searching of rotations would affect our metrics.

For our second round of experiments, we are interested in the effect of shape on deadlock for both RRVO and ORCA. To that end, we vary the number of sides used to approximate a bounding circle about an agent. The circle sizes are equivalent to the bounding circles of the agents from the first round of experiments.

For all 33 trials, we used 50 agents. To account for the fact that deadlocking agents will sometimes never reach their goals, we capped each trial at 20,000 frames. In each graph we show a 95% confidence interval around each measurement. We employed Welch's t test to test for statistical significance. Except where otherwise stated, $\alpha = 0.001$.

We experiment with RRVO in two scenarios that have high potential for deadlock:

Lines Five parallel lines of five agents move opposite another group of five parallel lines of five agents. Agents attempt to reach their horizontally-symmetric positions, which causes a great deal of congestion. An example simulation of this scenario is shown in Figure 5.1.

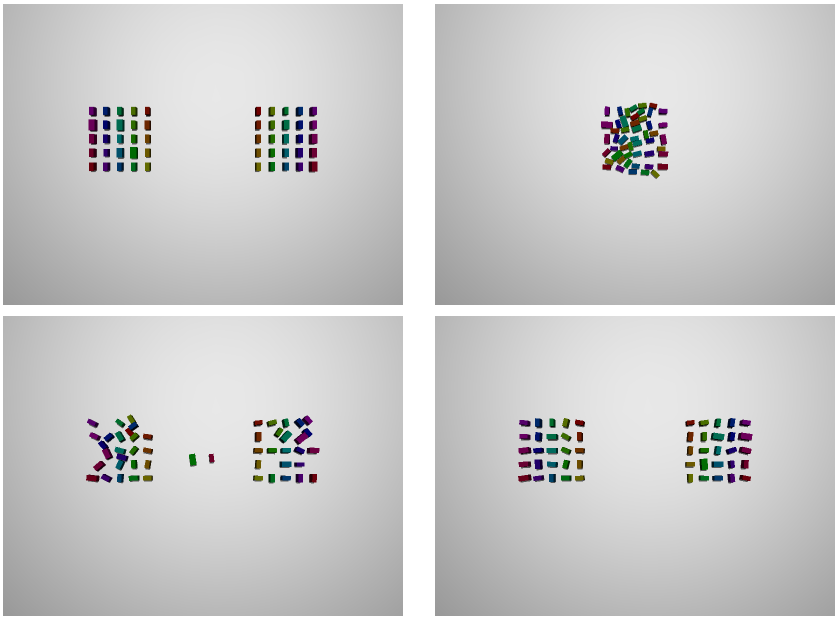


Figure 5.1: Progression of the *Lines* scenario for 50 agents. Agents are positioned in opposing groups of parallel lines of five, and instructed to reach their horizontally-symmetric positions. This scenario requires agents to navigate around many stopped agents with very small gaps between them.

Circle 50 agents are evenly distributed about a circle, and then instructed to reach the location directly opposite themselves. This scenario causes extreme crowd density near the center. Figure 5.2 shows an example simulation of this scenario.

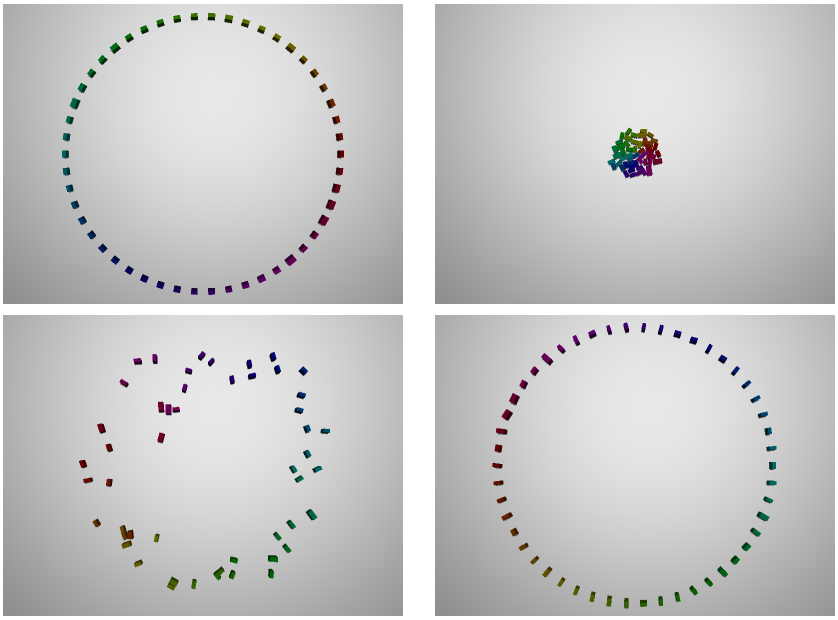


Figure 5.2: Progression of the *Circle* scenario for 50 agents. Agents are positioned evenly around a circle and directed to reach their antipodal position. Congestion forms in the middle of the circle, but is eventually resolved, and the agents reach their goals.

5.3 Results (Rectangles)

In this section we present the relative performance of RRVO and ORCA when rectangular agents (approximating humans) are used.

5.3.1 Frame rate

In Figure 5.3, we demonstrate the frame rates at which the *Lines* and *Circle* scenarios progressed, excluding render time. This figure is intended to show that even with average computing hardware, RRVO is able to run interactively for δ

values up to 10.

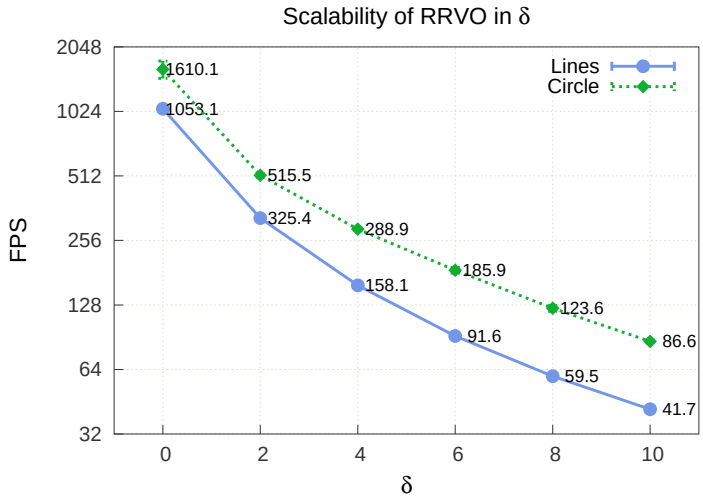


Figure 5.3: Frame rate for RRVO as δ increases in the *Lines* and *Circle* scenarios. RRVO remains interactive up through a δ value of 10.

Figure 5.4 shows RRVO's frame rate normalized to ORCA for both scenarios. The plot shows that RRVO slows down on the order of δ^2 albeit slightly better due to the optimizations mentioned in Section 4.2.1. While having worse performance than ORCA is a drawback of our method, it may be preferable when the alternative is deadlock.

5.3.2 Completion Rate

Our results show that RRVO enables more agents to reach their goals than ORCA. RRVO agents seem to rarely to mutually block each other. Instead, they tend to either rotate about each other, or use rotation to more efficiently form lanes. Conversely, ORCA agents often find themselves in local minima where only a large lateral velocity would resolve the deadlock.

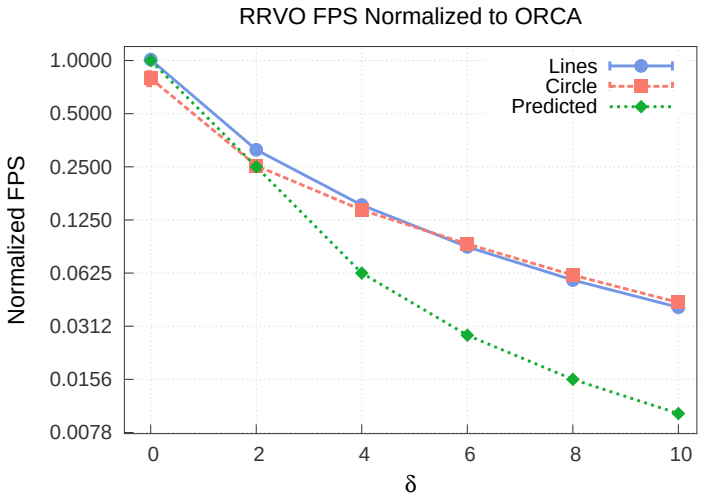


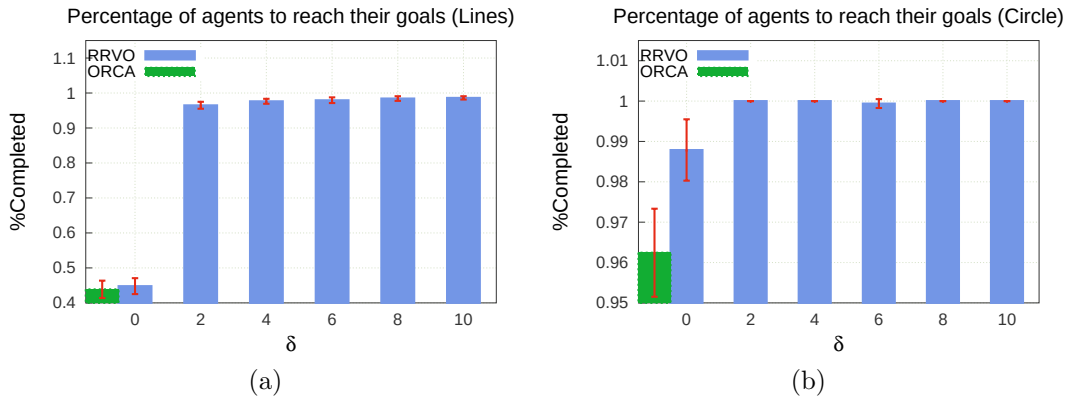
Figure 5.4: Normalized frame rate of RRVO to ORCA. RRVO's performance worsens relative to ORCA on the order of δ^2.

Figures 5.5(a) and (b) show the percentage of agents that reach their goals in the *Lines* and *Circle* scenarios, respectively. Even RRVO with $\delta = 2$ has an average completion percentage of about 96% while ORCA's completion rate is only 44% in the *Lines* scenario. In the Circle scenario, despite ORCA allowing 96% of agents to reach their goals on average, even RRVO with $\delta = 0$ has a significantly higher completion percentage at 98%.

RRVO with values of $\delta > 0$ have statistically higher completion percentages ($p_0 < 0.005$ for *Lines*, $p_0 < 0.001$ for *Circle*) against $\delta = 0$.

5.3.3 Completion Time

For those agents that do reach their goals, RRVO allows them to do so sooner than ORCA. Figures 5.6(a) and 5.6(b) demonstrate that even RRVO with $\delta = 0$, agents reach their goals quicker ($p_0 < 0.001$ except for $\delta = 0$ in the *Circle* scenario, where $p_0 < 0.005$). This indicates that agents using ORCA experience more interference from other agents and choose fewer deadlock-avoiding velocities than in RRVO.



(a) (b)

Figure 5.5: Percentages of agents to reach their goals in the **(a)** *Lines* and **(b)** *Circle* scenarios for RRVO and ORCA. RRVO enables more agents to reach their goals than ORCA.

5.3.4 Collisions

Figure 5.7(a) depicts the number of collisions for completing agents in the *Lines* scenario. These results show that the number of collisions that RRVO agents encounter may actually be far fewer than ORCA agents, even though RRVO approximates collision-avoiding rotation whereas ORCA uses an exact geometric solution (for translational motion only). The high number of collisions for ORCA is because ORCA agents continually bump against each other as they attempt to find feasible velocities.

The *Circle* scenario is perhaps a more accurate comparison of collision count, as the completion rates between ORCA and RRVO are more comparable. In this scenario, while the $\delta = 2$ case creates fewer collisions on average for RRVO than ORCA, the $\delta = 8$ and $\delta = 10$ cases saw significantly more collisions than ORCA ($p_0 < 0.005$ and $p_0 < 0.001$, respectively). Otherwise there is no significant difference in collision counts. The collision results for the *Circle* scenario are shown in Figure 5.7(b).

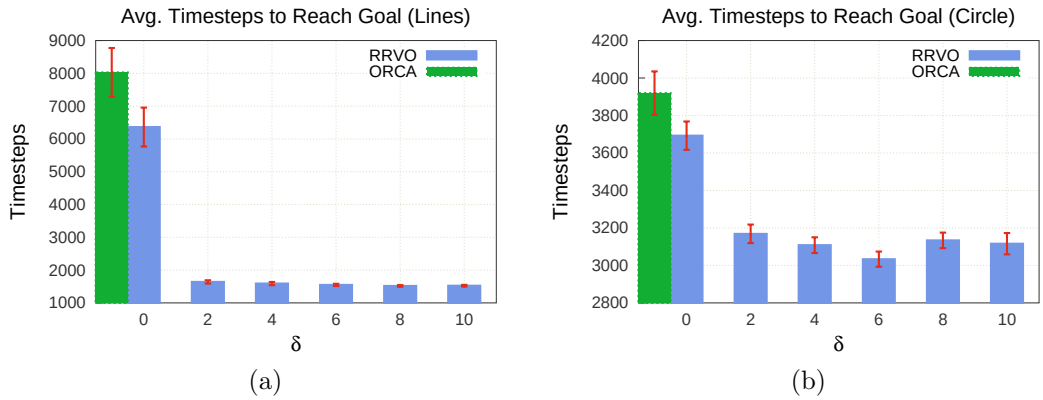


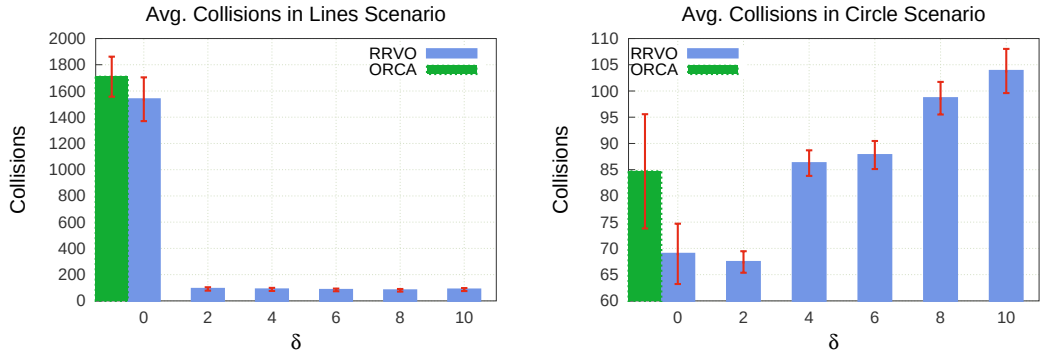
Figure 5.6: The number of frames, or timesteps, it takes for agents to reach their goals for the **(a)** *Lines* and **(b)** *Circle* scenarios. RRVO outperforms ORCA for all values of δ in that agents reach their goals quicker (However, $p_0 < 0.005$ for $\delta = 0$ in *Lines* instead of the usual $p_0 < 0.001$).

5.4 Effect of Shape

In Section 4.2, we showed how ORCA can have a deadlocking problem when polygonal agents are used. In this section we are interested in seeing the benefit RRVO has over ORCA as the shape of an agent approaches a circle. Bear in mind that the radii of the circles for these shapes remain invariant; we only increase the number of edges (sides). This has the effect of increasing the total area occupied by an agent, which could potentially make scenarios more difficult. However, we will show that the benefits of shorter sides outweigh the cost of occupying more space.

5.4.1 Completion Rate

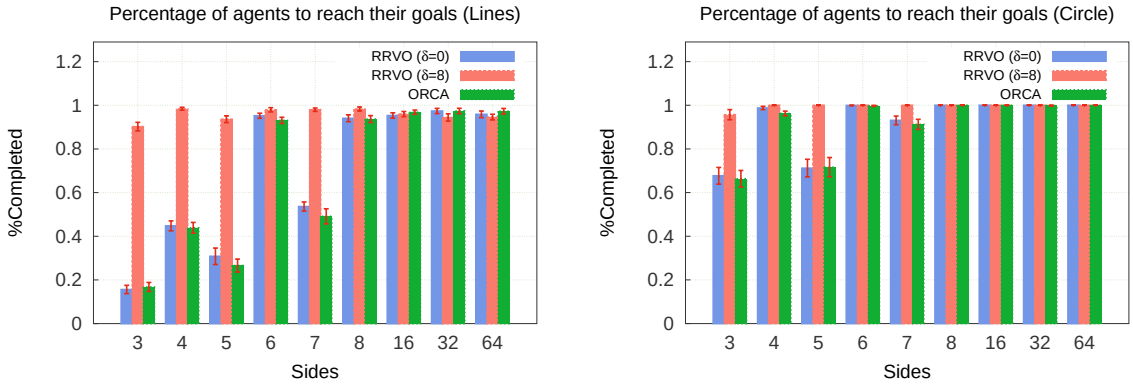
Figures 5.8(a) and (b) show the percentage of agents that reach their goals in the *Lines* and *Circle* scenarios, respectively. The figures demonstrate a positive correlation with edge count and completion rate. Meanwhile, RRVO with $\delta = 8$ consistently allows all agents to finish, achieving at 90% completion rate for triangles in the *Lines* scenario whereas ORCA and RRVO with $\delta = 0$ only permit 17% and



(a) (b)

Figure 5.7: The number collisions experienced, on average, by those agents who reached their goals in the **(a)** *Lines* and **(b)** *Circle* scenarios. Agents using RRVO generally do not collide more often on average than if they were to use ORCA, despite RRVO's relaxed collision-avoidance guarantees. In fact, in the *Lines* scenario they collide significantly less than in ORCA due to deadlocked agents colliding with completed ones.

16% respectively.

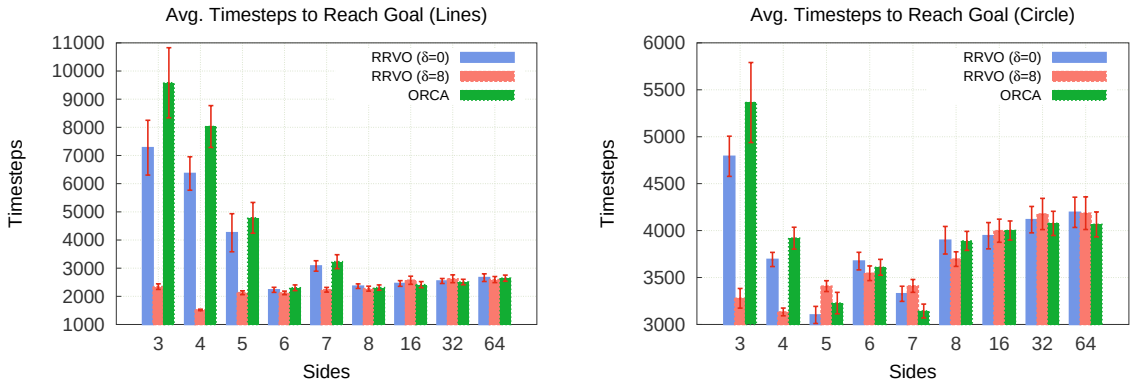


(a) (b)

Figure 5.8: Percentages of agents to reach their goals in the **(a)** *Lines* and **(b)** *Circle* scenarios for RRVO and ORCA. As shapes approach circular, more agents reach their goals. However, permitting rotation in general provides consistently high completion rates.

5.4.2 Completion Time

Figures 5.9(a) and (b) show the average number of frames it takes agents to reach their goals for the *Lines* and *Circle* scenario, respectively, as the number of sides is increased. What is interesting about these figures is that the completion rate steeply drops at first as the number of sides is increased, but then climbs afterwards before it stabilizes. This is likely the interplay between shorter average sides and larger overall area occupied. It seems that the ideal shape for an agent is the hexagon, which gives both high completion rates and low frame counts. Again, while ORCA's performance improved relative to RRVO as the shape approached a circle, RRVO ($\delta = 8$) agents consistently complete in the fewest frames. This result lends more credence to the notion that choosing high clearance rotations will result in less overall interference, and makes problems easier to solve.



(a) (b)

Figure 5.9: The number of frames, or timesteps, it takes for agents to reach their goals for the **(a)** *Lines* and **(b)** *Circle* scenarios as the number of sides in agent polygons is increased

6. CONCLUSION

In this work, we introduce Reciprocally-Rotating Velocity Obstacles, an extension and generalization of Optimal Reciprocal Collision Avoidance (ORCA) for local collision avoidance. RRVO enables finer modeling of agents that also allows for rotation and helps mitigate potential deadlock scenarios that arise when using polygonal agents. Our results show that even a little bit of rotation results in much less deadlock, and that the performance overhead RRVO incurs is quadratic in the granularity δ at which agents search for feasible rotations. Despite this drawback, small values of δ still result in an interactive frame rate, a low number of collisions, and more direct paths towards goals than ORCA. Finally, RRVO has plenty of room for additional parallelism, and we plan to explore this avenue in future work.

REFERENCES

[1] Itzhak Benenson. Multi-agent simulations of residential dynamics in the city. *Computers, Environment and Urban Systems*, 22(1):25–42, 1998.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[3] Corky Bingelli. *Interior Graphic Standards*. Wiley, New York, 2nd student edition edition, 2011.

[4] John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *Foundations of Computer Science, 1987., 28th Annual Symposium on*, pages 49–60. IEEE, 1987.

[5] Stephen Chenney. Flow tiles. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242, New York, NY, USA, 2004. ACM Press.

[6] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. Computational geometry (2nd revised ed.). pages 267–290, 2000.

[7] M. Erdmann and T. Lozano-Perez. On multiple moving objects. In *Robotics and Automation. Proceedings. 1986 IEEE International Conference on*, volume 3, pages 1419–1424, 1986.

[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] Maxim Garber and Ming C. Lin. Constraint-based motion planning using Voronoi diagrams. In *Algorithmic Foundations of Robotics V*, pages 541–558.

Springer, Berlin/Heidelberg, 2003. book contains the proceedings of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Nice, France, 2002.

[10] Russell Gayle, William Moss, Ming C. Lin, and Dinesh Manocha. Multi-robot coordination using generalized social potential fields. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 106–113, 2009.

[11] Andew Giese, Daniel Latypov, and Nancy M. Amato. Reciprocally-rotating velocity obstacles. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2014. to appear.

[12] Abhinav Golas, Rahul Narain, and Ming Lin. Hybrid long-range collision avoidance for crowd simulation. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pages 29–36. ACM, 2013.

[13] Stephen J. Guy, Jatin Chhugani, Sean Curtis, Pradeep Dubey, Ming Lin, and Dinesh Manocha. Pledestrians: a least-effort approach to crowd simulation. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '10, pages 119–128, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[14] 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*, SCA '09, pages 177–187, New York, NY, USA, 2009. ACM.

[15] Stephen J Guy, Sujeong Kim, Ming C Lin, and Dinesh Manocha. Simulating heterogeneous crowd behaviors using personality trait theory. In *Proceedings of*

the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation, pages 43–55. ACM, 2011.

[16] Liang He and Jur van den Berg. Meso-scale planning for multi-agent navigation. In *(ICRA), 2013*, 2013.

[17] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51:4282–4286, May 1995.

[18] Dirk Helbing and Peter Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[19] John E Hopcroft, Jacob T Schwartz, and Micha Sharir. On the complexity of motion planning for multiple independent objects; pspace-hardness of the "warehouseman's problem". *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[20] D. Hsu, J-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 2719–2726, 1997.

[21] Roger L. Hughes. A continuum theory for the flow of pedestrians. *Transportation Research Part B: Methodological*, 36(6):507 – 535, 2002.

[22] Mubbasir Kapadia, Shawn Singh, William Hewlett, and Petros Faloutsos. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 215–223. ACM, 2009.

[23] Ioannis Karamouzas, Peter Heil, Pascal van Beek, and Mark H Overmars. A predictive collision avoidance model for pedestrian simulation. In *Motion in Games*, pages 41–52. Springer, 2009.

[24] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robot. Automat.*, 12(4):566–580, August 1996.

[25] O. Khatib. Real–time obstacle avoidance for manipulators and mobile robots. *Int. J. Robot. Res.*, 5(1):90–98, 1986.

[26] David Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[27] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[28] Daniel Massaguer, Vidhya Balasubramanian, Sharad Mehrotra, and Nalini Venkatasubramanian. Multi-agent simulation of disaster response. In *ATDM workshop in AAMAS*, volume 2006. Citeseer, 2006.

[29] Rahul Narain, Abhinav Golas, Sean Curtis, and Ming C. Lin. Aggregate dynamics for dense crowd simulation. In *ACM SIGGRAPH Asia 2009 papers*, SIGGRAPH Asia '09, pages 122:1–122:8, New York, NY, USA, 2009. ACM.

[30] N. Pelechano, J. Allbeck, and N. Badler. Controlling individual agents in high-density crowd simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2007.

[31] J. H. Reif and H. Wang. Social potential fields: A distributed behavioral control for autonomous robots. *J. Robot. and Autonom. Sys.*, pages 171–194, 1999.

[32] C. W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics*, pages 25–34, 1987.

[33] C. W. Reynolds. Steering behaviors for autonomous characters. In *Game Developers Conference*, 1999.

[34] Mark Riedl, Cesare J Saretto, and R Michael Young. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 741–748. ACM, 2003.

[35] S. Rodriguez, A. Giese, N. M. Amato, S. Zarrinmehr, F. Al-Douri, and M. Clayton. Environmental effect on egress simulation. In *Proc. of the 5th Intern. Conf. on Motion in Games, 2012, Lecture Notes in Computer Scien ce (LNCS)*, pages 7–18, 2012.

[36] Seyed Abbas Sadat and Richard T Vaughan. Bravo: Biased reciprocal velocity obstacles break symmetry in dense robot populations. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 441–447. IEEE, 2012.

[37] Raimund Seidel. Linear programming and convex hulls made easy. In *Proceedings of the sixth annual symposium on Computational geometry*, SCG '90, pages 211–215, New York, NY, USA, 1990. ACM.

[38] Wei Shao and Demetri Terzopoulos. Autonomous pedestrians. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28, New York, NY, USA, 2005. ACM Press.

[39] S. Singh, M. Kapadia, B. Hewlett, G. Reinman, and P. Faloutsos. A modular framework for adaptive agent-based steering. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, pages 141–150, New York, NY, USA, 2011. ACM.

[40] Adrien Treuille, Seth Cooper, and Zoran Popovi. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[41] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In Cdric Pradalier, Roland Siegwart, and Gerhard

Hirzinger, editors, *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer Berlin Heidelberg, 2011.

[42] Jur van den Berg, Stephen J Guy, Jamie Snape, Ming C Lin, and Dinesh Manocha. Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation, 2011.

[43] Jur van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.

[44] H. Yeh, S. Curtis, S. Patil, J. van den Berg, D. Manocha, and M. Lin. Composite agents. In *Proceedings of the 2008 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '08, pages 39–47, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.