2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

Reciprocally-Rotating Velocity Obstacles

Andrew Giese, Daniel Latypov, and Nancy M. Amato

***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 generalizes ORCA by introducing a notion of rotation for polygonally-shaped agents. This generalization permits more realistic motion than ORCA and does not suffer from as much deadlock. In this paper, we present the theory of RRVO and show empirically that it does not suffer from the deadlock issue ORCA has, permits agents to reach goals faster, and has a comparable collision rate at the cost of performance overhead quadratic in the (typically small) user-defined parameter δ.**

I. INTRODUCTION

Collision-free path planning is a central part of any multi-agent system and is a longstanding problem in robotics in general. Planning a collision-free path for even a single agent in a continuous environment was found to be NP-Hard [3], and any centralized approach to planning collision-free paths for multiple agents is also PSPACE-hard [16]. Despite these theoretical hurdles, fast and efficient solutions have been designed using potential fields [22], priority-based decoupling [6], and sampling-based methods [21] [17] [24].

To satisfy performance 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 [23]) that can quickly solve path queries using A* or Dijkstra's algorithm. 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 [7]. These *Velocity Obstacle* (VO) variations are able to efficiently simulate agent movement for up to thousands of agents in real-time, and are amenable to parallelization. Most VO techniques assume disc-shaped robots translating in a plane, which may be an unsuitable representation for some agents (e.g., a bus). In some cases, designers may simply want to model finer interactions between agents, and bounding circles are simply insufficient. Another weakness of VO methods is that they restrict motion to translation, which is adequate if agents are represented as circles, but such an approach may create unsolvable scenarios like the example in Figure 1.

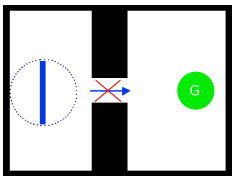


Fig. 1. A long skinny agent (blue rectangle) cannot reach its goal, G (green), if represented as its bounding circle (blue dotted circle) or if only translational movement is permitted.

We 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 breaks the symmetries that cause deadlock in ORCA. Our specific contributions include:

- Reciprocally-Rotating Velocity Obstacles (RRVO),
- A complexity analysis of RRVO, and
- An empirical study that shows how RRVO results in less deadlock, faster completion times, and comparable collision rates to ORCA in exchange for additional processing time quadratic in a (typically small) user-defined parameter δ.

II. RELATED WORK

In this section we discuss approaches to local collision avoidance found in the literature, and separate them into three categories: Reactive Models, Predictive Models, and Cellular Automata.

Reactive Models Local collision avoidance was influenced by Reynolds' seminal work where agents in a flock were attracted to the flock center and repulsed from nearby neighbors [25]. Helbing expanded Reynold's ideas to include

This research supported in part by NSF awards CNS-0551685, CCF-0833199, CCF-0830753, IIS-0916053, IIS-0917266, EFRI-1240483, RI-1217991, by NIH NCI R25 CA090301-11, by Chevron, IBM, Intel, Oracle/Sun and by Award KUS-C1-016-04, made by King Abdullah University of Science and Technology (KAUST).

All authors are with the Dept. of Computer Science and Engineering, Texas A&M University, College Station, TX 77843
{awg4619, dlatypov, amato}@cse.tamu.edu

the idea of personal space in crowds, becoming the first of many "social forces" models [15].

In [9], Gayle et al. simulated collision-free translational and rotational motion between arbitrary polyhedra by applying social forces to sampled points on polyhedra surfaces. Forces were incorporated into the physics-based motion planner described in [8] as constraints. 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 use to consciously select desirable orientations.

Predictive Models Predictive models anticipate future collisions so that agents can take steps to avoid them. Often this means linearly extrapolating neighbor trajectories.

Our work most closely follows that of Reciprocal Velocity Obstacles (RVO), first presented by van den Berg *et al.* in [31] and then renamed to Optimal Reciprocal n-Body Collision Avoidance (ORCA) in [29]. RVO/ORCA involves predicting the set of collision-causing velocities for each agent by assuming linear velocities, and then choosing a velocity not in that set. The main difference between RVO and ORCA is that ORCA solves a linear program whereas RVO searches for via sampling.

RVO and ORCA have been the focus of much research, and there are many variations and optimizations in the literature [10] [11] [12] [13] [14] [26] [32]. Our work is unique in that it adds another layer to RVO-based collision avoidance and could thus incorporate nearly all these variations.

In [20], the authors used a modified social forces model that predicted future collisions and applied evasive forces from those locations. In [19], instead of assuming a predictable trajectory by neighbors, Egocentric Affordance Fields simply assigns repulsive potential fields based on their proximities and relative velocities.

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 [18] and Chenney [4], these approaches discretize the environment into a grid of varying coarseness and assign velocity fields to 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 generates potential fields based on how much weight agents assign path length, time, or congestion [28]. Agents in the same grid cell may occasionally intersect, so all pairs are iterated over once to enforce a minimum separation distance. Such a policy works well in practice, but was not guaranteed to satisfy a minimum distance constraint.

Cellular automata approaches are used extensively for their ability to simulate extremely large crowds, but 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.

III. PROBLEM DEFINITION

In this section, we describe the collision avoidance problem for multi-agent systems. 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 (robots) R such that each agent $a \in R$ can be represented with a position p_a and an orientation $\theta_a \in (-\pi, \pi]$. The agent's geometry, G_a, is a convex polygon consisting of the vertices $\{g_{a_1}, g_{a_2}, \cdots, g_{a_m}\}$ centered about p_a and rotated about the vertical axis by θ_a. Agent a additionally has a velocity v_a, a maximum translational speed s_{max_a}, and a maximum angular velocity ω_{max_a}. Finally, the agent has a preferred translational velocity v_{pref_a} and orientation θ_{pref_a}.

Problem 1. *(Collision Avoidance) Each agent $a \in R$ must choose a new velocity and orientation at each timestep of the simulation such that a is guaranteed not to collide with any other agent for a time interval of length $\geq \tau$.*

Preferably, the new velocity v_{new_a} and orientation θ_{new_a} 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 v_{pref_a} or θ_{pref_a} and assume that both are provided by the high-level planner. For example, $v_{pref_a} = (Goal_a - p_a)$ and $\theta_{pref_a} = \text{ATAN2}(v_{new_a})$. We assume agents do not explicitly coordinate to select new orientations and velocities.

IV. 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 each timestep. First we provide some background, and then we introduce the idea of reciprocal rotation. Finally, we address the theoretical complexity of RRVO, obstacle avoidance and collision resolution.

A. Reciprocal Velocity Obstacles

A velocity obstacle [7] is defined as the set of all robot velocities that will cause a collision with an obstacle at some future time. In its original formulation, agents assume others continue moving along a linear trajectory. In [29], 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

Input: $neighbors_a \subset R$
Output: v_{new_a}
1: **for all** $neighbor_i \in neighbors_a$ **do**
2: Add a linear constraint on v_{new_a}
3: Solve linear program to find v_{new_a}

1) Construction: At the core of Algorithm 1 is assigning a linear constraint to an agent'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 and b, a will create a velocity obstacle representing b (and vice-versa) such that a wishes to choose a guaranteed collision-free velocity for the time interval τ. We denote this velocity obstacle representing b as $VO^{\tau}_{a|b}$. The computation of $VO^{\tau}_{a|b}$ is shown in Algorithm 2.

Algorithm 2 Compute velocity obstacle induced by b on a

Input: Agents a and b, time horizon τ
Output: $VO^{\tau}_{a|b}$
1: $M \leftarrow G_a \oplus G_b$ //Minkowski Sum
2: TRANSLATE$\left(M, \frac{p_b(1-\tau)-p_a(1+\tau)}{\tau}\right)$
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: $VO^{\tau}_{a|b} = VO^{\tau}_{a|b} \cup \{m_i\}$
8: //Represent unbounded sides with tangent vectors
9: $VO^{\tau}_{a|b}.left\_leg \leftarrow 2t_{left}$
10: $VO^{\tau}_{a|b}.right\_leg \leftarrow 2t_{right}$

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

- It contains the Minkowski sum of a and b's geometry, $M = G_a \oplus G_b$, 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 2(a)- (b) show an example of this construction.

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 and b and translating it to p_B allows us to discard a's geometry and only consider it as a point robot defined by p_a. To get an absolute frame of reference, we consider a at the origin (egocentric coordinates), so we translate M by $-p_a$. Furthermore, we only require that a 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 $VO^{\tau}_{a|b}$. Lines 5-7 use these endpoints to compute the line segment(s) on M that bound $VO^{\tau}_{a|b}$.

2) Geometric Linear Programming: Note that $VO^{\tau}_{a|b}$ is a constraint on the *relative* velocity of a and b, defined as:

Definition 1. *(Relative Velocity)* $v^{rel}_{a|b} = v_a - v_b$

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

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

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

When testing obstacle-agent collisions, full responsibility is on the agent to affect $v^{rel}_{a|b}$, so v_a must change by at least u to avoid collisions. However, for agent-agent collisions, each agent a and b assumes half the responsibility in affecting $v^{rel}_{a|b}$, so the minimum amount that v_a and v_b must change is $\frac{u}{2}$. In [29], they prove this formulation still results in collision-free motion.

$v_a + u$ is a vector facing in the direction that v_a 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 $v_a + u$. 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 2(b).

Each velocity obstacle for agent a induces a new linear constraint on a'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 [27].

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 [29].

B. 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, $v^{rel}_{a|b}$ almost always eventually projects onto a tangent line, allowing agents to move around each other, as in Figure 3(a).

When agents are polygonal, though, this construction has a serious flaw. When two polygonal agents interact,

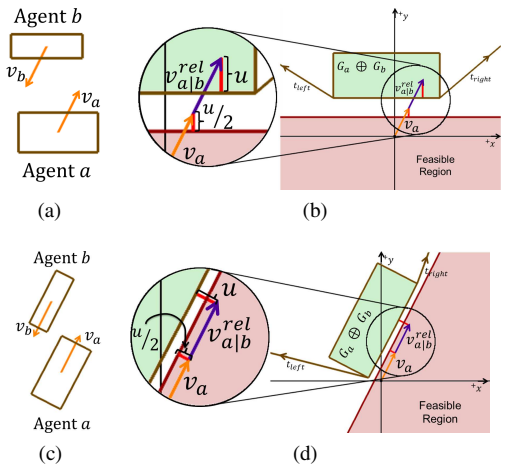


Fig. 2. **(a)** Two rectangular robots a and b on a collision course. **(b)** Construction of $VO^{\tau}_{a|b}$ for the scenario shown in (a). A linear constraint (pink region) on v_a is derived from the velocity obstacle, and it can be seen that v_a does not lie in the feasible region. **(c)** The same scenario as before, except a and b are rotated. **(d)** This rotation creates a scenario where v_a is already a feasible velocity for the next timestep.

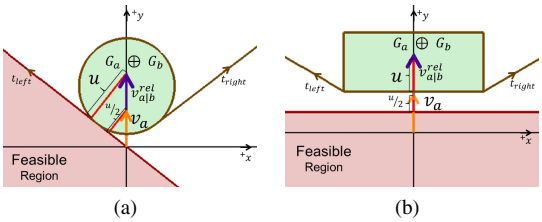


Fig. 3. **(a)** When two circular agents with opposing velocities meet, eventually they are instructed to choose lateral velocities. **(b)** When two rectangular agents with opposing velocities meet, they may never choose a lateral velocity.

there is no guarantee that the tangent points on $G_a \oplus G_b$ grow nearer as the agents do, which can cause deadlock, as shown in Figure 3(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.

When we permit polygonal agents a and b to rotate, the shape of the velocity obstacles they induce on each other will change. As shown in Figure 2(c) and Figure 2(d), as agents actively rotate, they can reduce the probability of collision without needing to modify their translational velocities.

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 a generalization of ORCA.

1) Method: In this subsection we present the 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.

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

Input: $neighbors_a \subset R$, ω_{max_a}
Output: $v_{new_a}, \theta_{new_a}$

```
LP ← a set of linear programs
for all Orientations reachable by a do
    for all neighbor_i ∈ neighbors_a do
        for all Orientations reachable by neighbor_i do
            LP[i] = LP[i] ∪ linear constraint on v_new_a
Solve linear programs in LP and choose desired (v_new_a, θ_new_a)
```

Not every agent in the environment needs to be considered as a reciprocally-rotating neighbor. In fact, if we could observe other agents' maximum speeds, 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, r_a, is the maximal Euclidean distance from p_a to some $g_{a_i} \in G_a$.*

Definition 4. *(Rotation Neighbors) The rotation neighbors N_r for agent $a \in R$, N_{r_a}, for time interval τ, are the set* $\{\forall b \in R \mid b \neq a, ||p_b - p_a|| - \tau(s_{max_a} + s_{max_b}) < (r_a + r_b)\}$

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 $a \in R$ has a set of reachable orientations $\theta_{reach_a} \subseteq (-\pi, \pi]$ from which it will choose θ_{new_a}.

Definition 5. *(Reachable Orientations)* $\theta_{reach_a} = \{\theta_a - (\tau\omega_{max_a}), \theta_a + (\tau\omega_{max_a})\}$
$\setminus \{\forall \theta \mid \exists b \in N_{r_a}, a \neq b, p_a \in G_a \oplus G_b\}$

Implicit in Definition 5 is that $\forall b \in R\ (a \neq b)$, b is rotated by the same $\Delta\theta$ (or $-\Delta\theta$) as a. That is, a cannot choose a change in orientation if an equal (or equally opposite) change

in θ_b would cause a collision. Also not stated is that the Minkowski sum between a and b's geometries, $G_a \oplus G_b$, is centered at p_b.

RRVO approximates the bounds on θ_{reach_a} by discretizing the set of rotations by the constant δ. For each orientation of a, we check for collision against δ neighbor orientations, for each neighbor. If there is a collision, then θ_{reach_a} is approximately bounded in that direction (clockwise or counter-clockwise).

After θ_{reach_a} is computed, a desired orientation must be selected from the interval. If the chosen θ_{new_a} is not reachable in the next frame, then a 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); 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 $v^{rel}_{a|b}$to $VO^{\tau}_{a|b}$. Geometrically, we developed the notion of a *signed magnitude* for the vector u. We consider u's magnitude as negative if $VO^{\tau}_{a|b}$ is contained within $VO^{\tau}_{a|b}$, and positive otherwise. More formally, u's magnitude is negative if the angle between it and a vector pointing toward the center of $VO^{\tau}_{a|b}$ 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 $|atan2(v)|-|\theta|$ (favor those that allow agents to face their velocity vectors)
3) Minimization of $|v - v_{pref}|$ (favor those that allow agents to follow their preferred velocities)

Choosing a new velocity that satisfies all constraints should lead to collision-free rotation and translation. However, because we've discretized the set of orientations instead of computing an actual swept volume, 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 θ_{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 neighbors check because we do not allow agents to deduce others' maximum speeds,

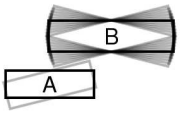


Fig. 4. Agent A, represented as a black rectangle, bounds its maximum counter-clockwise rotation by assuming another agent, B may rotate at most as much as A. A discovers that a rotation of about 14° (light gray) is the maximum it can rotate such that it doesn't intersect the swept volume of B through $\pm 14°$ (gray).

2) Obstacles: RRVO can be easily 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.

In the case that obstacles may also rotate, agents cannot make the reciprocity assumption like they do with other agents, and would therefore need to consider obstacles as their bounding circles, or apply some other heuristic or approximation technique but we do not consider any such approaches in this paper.

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. We choose to construct a linear constraint based on the distance from p_a to the nearest point on M scaled by the timestep duration instead. 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) 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 [1]. 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 IV-A.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 and b requires the Minkowski sum of G_a and G_b, 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 $O(||G_a|| + ||G_b||)$ using the convolution method, and finding the tangents is at worst $O(n)$ in the number vertices of the Minkowski sum [5] (by construction of the Minkowski sum, we retain the counter-clockwise sorting of the vertices). Additionally, for each $\Delta\theta_a$ that we create a linear program for, each neighbor adds δ constraints to the linear program on account 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 $O(\delta^2 \textsc{Max}(||G_i||))$. 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.

V. EXPERIMENTAL RESULTS

In this section, we show how RRVO performs against a polygonally-based ORCA implementation.

A. Metrics

Our metrics included average frame rate (FPS) normalized by the frame rate of ORCA, percentage of agents at their goals at simulation end, average number of frames it took each agent to arrive at its goal, and the mean number of collisions agents experienced throughout the simulation. 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. Clearly, deadlocking agents will also skew the average number of frames it takes for agents to reach their goals.

We varied the value of δ to observe its effect on our metrics. To account for the fact that deadlocking agents will sometimes never reach their goals, we capped each trial at 20,000 frames. Each experiment was repeated for 33 trials. 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, $\alpha = 0.001$.

B. Experimental Setup

We implemented RRVO in C++ by adapting the RVO2 library [30]. The library had 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.

We experimented with RRVO in two scenarios that had 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.

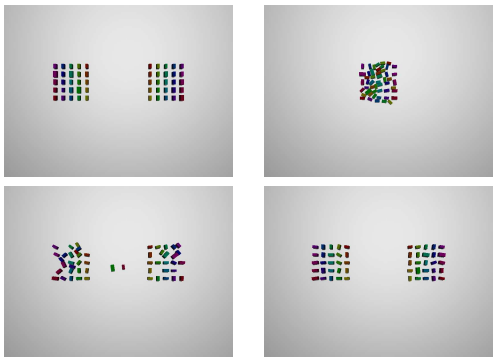


Fig. 5. Progression of the *Lines* scenario for 50 agents. Agents are positioned in opposing groups of parallel lines of five, and instructed to reach their horizontally-symmetric positions. This scenario requires agents to navigate around many stopped agents with very small gaps between them.

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 6 shows an example simulation of this scenario.

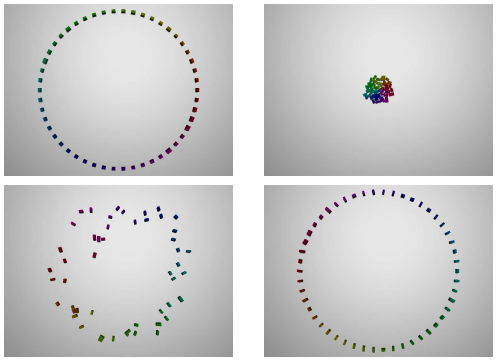


Fig. 6. Progression of the *Circle* scenario for 50 agents. Agents are positioned evenly around a circle and directed to reach their antipodal position. Congestion forms in the middle of the circle, but is eventually resolved, and the agents reach their goals.

For all trials, we used 50 rectangular agents. The length-to-width ratio of the rectangles was approximately 1.83, which is intended to model the shoulder-to-chest depth ratio of the average human [2].

C. Results

1) Frame rate: In Figure 7, 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.

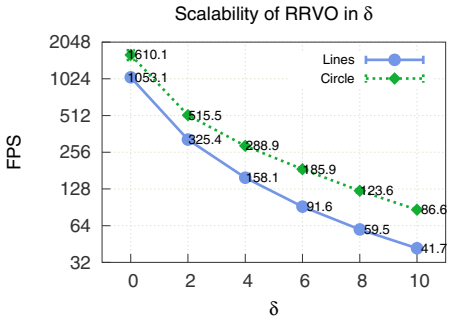


Fig. 7. Frame rate for RRVO as δ increases in the *Lines* and *Circle* scenarios. RRVO remains interactive up through a δ value of 10.

Figure 8 shows RRVO's frame rate normalized to ORCA for both scenarios. The plot shows that, while RRVO slows down on the order of δ^2, our implementation outperforms the predicted slowdown, likely due to the optimizations mentioned in Section IV-B.1. While having worse performance than ORCA is a drawback of our method, it may be preferable when the alternative is deadlock.

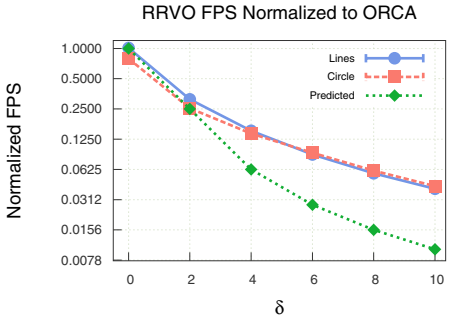
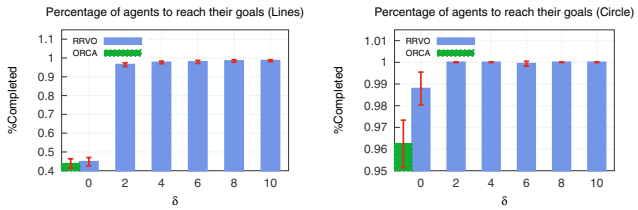


Fig. 8. Normalized frame rate of RRVO to ORCA. RRVO's performance worsens relative to ORCA on the order of δ^2.

2) Completion Rate: RRVO enables more agents to reach their goals than ORCA. Figures 9(a) & (b) show the percentage of agents that reached their goals in the *Lines* and *Circle* scenarios, respectively. Even RRVO with $\delta = 2$ had an average completion percentage of about 96% while ORCA's completion rate was 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 $\delta > 0$ had statistically higher completion percentages ($p_0 < 0.005$ for *Lines*, $p_0 < 0.001$ for *Circle*) against $\delta = 0$.

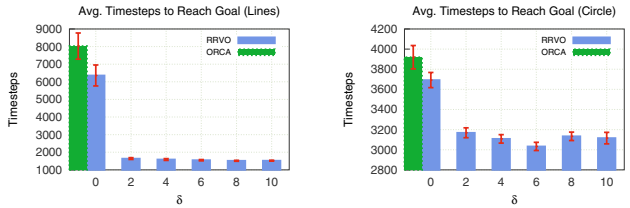
3) Completion Time: For those agents that do reach their goals, Reciprocally-Rotating Velocity Obstacles allows them to do so sooner than ORCA. Figures 10(a) and 10(b) demonstrate that even RRVO with $\delta = 0$, agents reach their goals quicker ($p_0 < 0.001$ except for $\delta = 0$ in the *Circle* scenario, where $p_0 < 0.005$). This indicates that agents using ORCA experience more interference from other agents



(a) (b)

Fig. 9. 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.

and choose less deadlock-avoiding velocities than in RRVO. Even in the $\delta = 0$ case, RRVO precludes a projection of a relative velocity from being parallel on the velocity obstacle to account for the deadlocking case shown in Figure 3(b).



(a) (b)

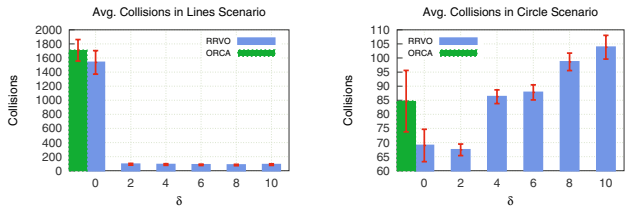
Fig. 10. The number of frames, or timesteps, it took for agents to reach their goals for the (a) *Lines* and (b) *Circle* scenarios. RRVO outperforms ORCA for all values of δ in that agents reach their goals quicker (However, $p_0 < 0.005$ for $\delta = 0$ in *Lines* instead of the usual $p_0 < 0.001$).

4) Collisions: RRVO approximates collision-avoiding rotation whereas RRVO uses an exact geometric solution (for translational motion only). Despite this apparent drawback, the number of collisions that RRVO agents encounter may actually be far less than ORCA agents, as shown in Figure 11(a), which depicts the amount of collisions for completing agents in the *Lines* scenario. Because of the significant deadlock faced by ORCA agents, they continue to 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 less collision 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). Othewise there is no difference in collision counts. The collision results for the *Circle* scenario are shown in Figure 11(b).

VI. CONCLUSION

In this work, we introduce Reciprocally-Rotating Velocity Obstacles, a 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) (b)

Fig. 11. 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.

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, low amounts of collision, 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.

ACKNOWLEDGMENT

We would like to thank the GAMMA group at UNC Chapel Hill for helping us 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

REFERENCES

[1] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

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

[3] J. Canny and J. 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.

[4] S. 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.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry (2nd revised ed.). pages 267–290, 2000.

[6] 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.

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

[8] M. Garber and M. 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.

[9] R. Gayle, W. Moss, M. C. Lin, and D. Manocha. Multi-robot coordination using generalized social potential fields. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 106–113, 2009.

[10] A. Golas, R. Narain, and M. 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.

[11] S. J. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin, and D. 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.

[12] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. 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.

[13] S. J. Guy, S. Kim, M. C. Lin, and D. 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.

[14] L. He and J. van den Berg. Meso-scale planning for multi-agent navigation. In *(ICRA), 2013*, 2013.

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

[16] J. E. Hopcroft, J. T. Schwartz, and M. 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.

[17] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. *Int. J. Comput. Geom. & Appl.*, pages 495–517, 1999.

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

[19] M. Kapadia, S. Singh, W. Hewlett, and P. Faloutsos. Egocentric affordance fields in pedestrian steering. In *Proceedings of the 2009 symposium on Interactive 3D graphics and games*, pages 215–223. ACM, 2009.

[20] I. Karamouzas, P. Heil, P. van Beek, and M. H. Overmars. A predictive collision avoidance model for pedestrian simulation. In *Motion in Games*, pages 41–52. Springer, 2009.

[21] 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.

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

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

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

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

[26] S. A. Sadat and R. 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.

[27] R. 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.

[28] A. Treuille, S. Cooper, and Z. Popovi. Continuum crowds. *ACM Trans. Graph.*, 25(3):1160–1168, 2006.

[29] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In C. Pradalier, R. Siegwart, and G. Hirzinger, editors, *Robotics Research*, volume 70 of *Springer Tracts in Advanced Robotics*, pages 3–19. Springer Berlin Heidelberg, 2011.

[30] J. van den Berg, S. J. Guy, J. Snape, M. C. Lin, and D. Manocha. Rvo2 library: Reciprocal collision avoidance for real-time multi-agent simulation, 2011.

[31] J. van den Berg, M. Lin, and D. 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.

[32] 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.