Reciprocally-Rotating Velocity Obstacles

Andrew Giese awg4619@cse.tamu.edu
Daniel Latypov dlatypov@cse.tamu.edu
Nancy M. Amato amato@cse.tamu.edu

Technical Report TR13-009
Parasol Lab
Department of Computer Science and Engineering
Texas A&M University
College Station, Texas, 77843-3112, USA

October 28, 2013

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 of the biggest challenges faced by local collision avoidance techniques is the problem of deadlock, where the method fails to taxi one or more agents to their goals due to local minima or cycling. One state-of-the-art local collision avoidance technique is Reciprocal Velocity Obstacles (RVO). Despite being fast and efficient for circular or spherically-shaped agents, RVO can have a high rate of deadlock when other shapes are used. To address the deadlocking problem of RVO, we introduce Reciprocally-Rotating Velocity Obstacles (RRVO). RRVO generalizes RVO by introducing a notion of rotation for polygonally-shaped agents. This generalization permits more realistic motion than RVO and does not suffer from nearly as much deadlock. In this paper, we present RRVO's theory, discuss its complexity, provide implementation optimizations, and perform an empirical study comparing RRVO and RVO. Our results show that RRVO does not suffer from the deadlock issue of RVO for rectangular agents, at the cost of performance overhead linear in δ, a granularity parameter of RRVO.

1 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 [9]. Despite these theoretical hurdles, fast and efficient solutions have been designed using potential fields [14], priority-based decoupling [6], and sampling-based methods [13] [10] [16].

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 [21], emergency training [17], entertainment [20], urban planning [1], and more.

To satisfy real-time 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 [15]) 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 [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 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. When agents are represented as circles, translational movement alone is sufficient 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, like the example in Figure 1.

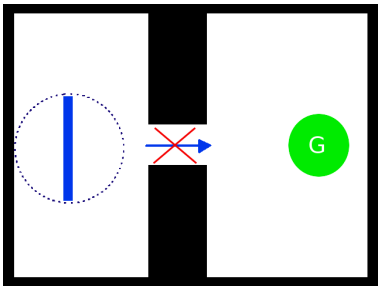

Figure 1: A long skinny agent (blue rectangle) cannot reach its goal, G (green), if represented as its bounding circle (blue dotted circle) or if only translational movement is permitted.

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 discretizing neighbors' potential orientations into a manageable set, agents can use this set to represent linear constraints on their own orientations and velocities. As a result of choosing high clearance rotations, RRVO breaks the symmetries that cause deadlock in RVO.
Our specific contributions include:

- Reciprocally-Rotating Velocity Obstacles (RRVO),
- A complexity analysis of RRVO, and
- Empirical evidence that RRVO suffers from less deadlock than RVO, at the cost of additional processing time approximately linear in one of RRVO's parameters.

2 Related Work

Real-time collision avoidance in multi-agent systems has a rich body of literature. Research can be roughly separated into agent-based and non agent-based approaches. In the former, individual

agents choose actions based on local information and in the latter, simulated vector fields act upon agents as if they were particles.

2.1 Agent-based Systems

Local collision avoidance was popularized largely by Reynolds' seminal work [19] where agents in a flock assigned a repulsive force to nearby neighbors to avoid collisions. Attractive forces between flock members to the flock centroid and the global goal drew the flock nearer to the goal while remaining cohesive. Helbing [8] expanded this idea to include general social forces acting as attractive and repulsive impulses.

Cognitive models attempt to mimic actual human behavior, perception, and locomotion at a fine scale. These methods achieve the highest amount of realism, but currently fail to achieve real time performance for more than small crowds. Shao and Terzopoulous [23] constructed a system where agents could sense ground height as well as static and dynamic obstacles. Agents were equipped with reactive rule-driven behaviors to avoid collision in crowded scenarios.

Predictive models anticipate future collisions so that agents can take steps to avoid them. Egocentric Affordance Fields [11] creates potential fields in concentric rings about an agent. Areas around other agents are rated as threatening based on their proximity and relative velocity. Each agent's resulting motion is the result of summing the forces acting on it at any point in time.

In [24], 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 less overall resources to achieve 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.

Our work most closely follows that of Reciprocal Velocity Obstacles (RVO), first presented by van den Berg *et al.* in [28]. RVO involves predicting the set of collision-causing velocities for each agent by assuming neighboring agents will continue along a linear velocity. Agents implicitly cooperate to choose velocities that guarantee collision-free movement.

In [12], the authors also linearly extrapolate the velocities of other agents to predict future collisions. Agents are then given a minimal evasive force to avoid the collision.

2.2 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 Chenney [4], these approaches discretize the environment into a grid of varying coarseness, and assign velocity fields to individual cells.

Treuille *et al.*'s Continuum Crowds [25] maps agents to grid cells and generates potential fields based on how much weight an agent assigns a particular variable such as path length, time, or congestion. Computing an agent's path is done by integrating the sum of potentials acting on it.

Narain *et al.* [18] 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. These cellular automata methods suffer from a lack of realism in that agents can never theoretically collide.

3 Problem Definition

In this section we describe the collision avoidance problem for multi-agent systems. This definition and its discussion follows the conventions used in [26]. 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 in the plane 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} as well as a preferred 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 $\geq \tau$.*

Agents do not explicitly coordinate to select new orientations and velocities; there is no way for an agent to know what the chosen velocity and orientations of its neighbors will be.

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. Local collision avoidance techniques such as RVO 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, $v_{pref_a} = (Goal_a - Start_a)/||Goal_a - Start_a||$ and $\theta_{pref_a} = \text{ATAN2}(v_{new_a})$.

4 Velocity Obstacles

In this section we review the concepts behind Velocity Obstacles, including defining them, constructing them, and using them to solve for an agent's new velocity.

A velocity obstacle was originally defined in [7] as the set of all robot velocities that will cause a collision with an obstacle at some future time. In its original formulation, agents assumed that other agents would continue moving along their observed velocities. In [26], this assumption was changed so that an agent would assume the other agents would take on half the responsibility of avoiding a collision (reciprocity). For obstacles, however, agents must still assume full responsibility.

4.1 Construction

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 1.

Geometrically, $VO^{\tau}_{a|b}$ is a 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.

Algorithm 1 Compute velocity obstacle induced by b on a

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

1: $M \leftarrow G_a \oplus G_b$
2: TRANSLATE$\Big(M,\ \frac{p_b(1-\tau)-p_a(1+\tau)}{\tau}\Big)$
3: SCALE$(M,\ \frac{1}{\tau})$
4: $(t_{left},\ t_{right}) \leftarrow$ COMPUTETANGENTS$(M,\ 0)$
5: **for all** $g_i \in G_M$ **do**
6: **if** $((t_{right} - t_{left}) \times (g_i - t_{left})) \leq 0$ **then**
7: $VO^{\tau}_{a|b} = VO^{\tau}_{a|b} \cup \{g_i\}$
8: //Represent unbounded sides with tangent vectors
9: $VO^{\tau}_{a|b}.left\_leg \leftarrow 2t_{left}$
10: $VO^{\tau}_{a|b}.right\_leg \leftarrow 2t_{right}$

An example of this construction is shown in Fig. 2(a) and Fig. 2(b).

The translation applied to M on line 2 of Algorithm 1 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 would like to 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 by $\frac{1}{\tau}$ (line 3), and similarly scale its position vector via translation. This has the effect of 'dragging' the Minkowski sum nearer to the origin to simulate future timesteps while maintaining the same tangent lines.

COMPUTETANGENTS() (line 4) 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 segments on M that additionally bound $VO^{\tau}_{a|b}$.

4.2 Geometric Linear Programming

Note that $VO^{\tau}_{a|b}$ is a constraint on the *relative* velocity of a and b, that we define 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 so that the two agents do not collide. We represent this minimal change by vector u:

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 [26], 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 can be considered 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 Fig. 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 [22].

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. The three-dimensional linear program also runs in $O(n)$ expected time [26].

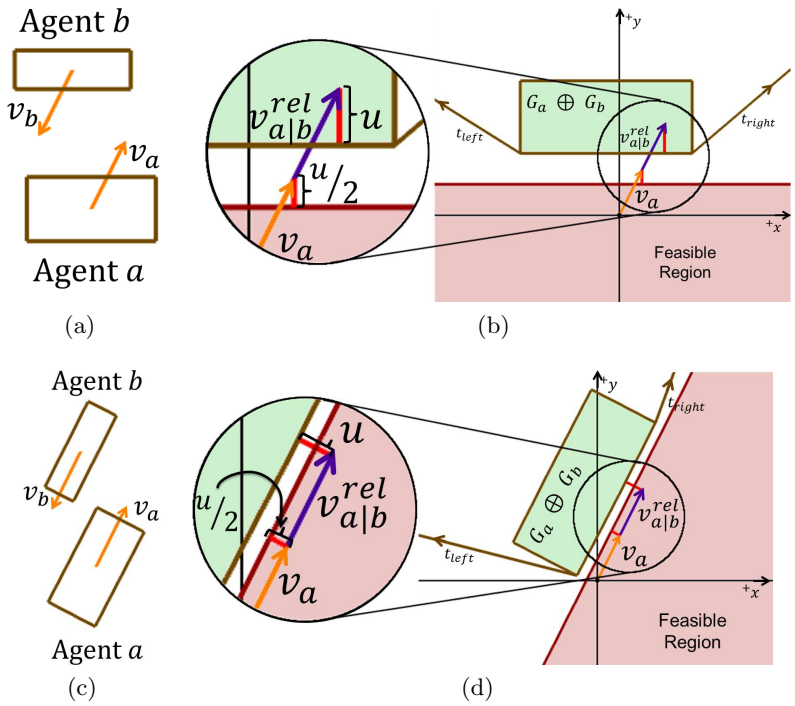

Figure 2: **(a)** Two rectangular robots a and b on a collision course. **(b)** Construction of $VO^{\tau}_{a|b}$ for the scenario shown in (a), and placed at egocentric coordinates to a. The velocity obstacle contains and is partly bounded by the Minkowski sum of a and b's geometries (green). Tangent points from the origin, t_{left} and t_{right}, serve as endpoints for rays that additionally bound the velocity obstacle. A linear constraint (pink region) on v_a is derived by computing u, the vector from $v^{rel}_{a|b}$ to the nearest point on the boundary of $VO^{\tau}_{a|b}$. The constraint is only $\frac{u}{2}$ distance from v_a due to the reciprocity assumption. Agent a must choose a new velocity within this feasible region, as v_a does not currently lie in it. *Magnified:* the region encompassing $v^{rel}_{a|b}$ and v_a, to show the projection of $v^{rel}_{a|b}$ onto $VO^{\tau}_{a|b}$'s boundary to discover u, and the corresponding placement of the linear constraint from v_a. **(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. *Magnified:* the region encompassing $v^{rel}_{a|b}$ and v_a, to show the projection of $v^{rel}_{a|b}$ onto $VO^{\tau}_{a|b}$'s boundary to discover u, and the corresponding placement of the linear constraint from v_a.

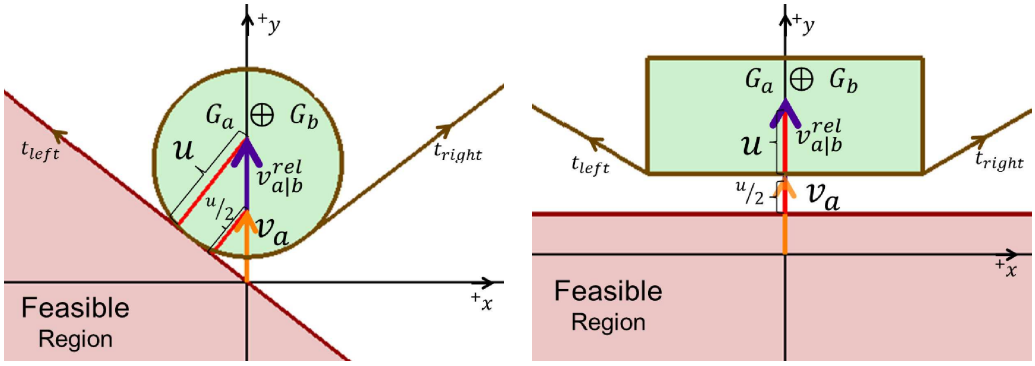

Figure 3: **(a)** When two circular agents with opposing velocities meet, eventually they are instructed to choose lateral velocities. **(b)** When two rectangular agents with opposing velocities meet, they may never choose a lateral velocity.

5 Reciprocal Rotation

The method in the previous section of 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. Even if two agents had opposing goals, $v_{a|b}^{rel}$ would eventually project onto a tangent line, causing agents to move around each other, as shown in Figure 3(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 $G_a \oplus G_b$ grow nearer as the agents do, which can cause deadlock, as shown in Figure 3(b). We can eliminate the possibility of deadlock between two agents by disallowing a projection of $v_{a|b}^{rel}$ onto a point oppositely parallel it. When we also allow agents to rotate, we can reduce overall congestion and greatly reduce the amount of deadlock.

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

If one agent may rotate, then it is reasonable for it to assume that others may rotate as well. A rotating agent must therefore take into account the possibility that its neighbors are using similar rotation strategies. 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. Unlike RVO, RRVO easily handles rotating agents, and considers convex obstacles as special cases of (convex) agents.

5.1 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.

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}. The notion of reciprocal rotation helps restrict the set of considered orientations to those that don't cause collision.

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.

Discovering the bounds on θ_{reach_a} can be a difficult problem to solve efficiently, which we'll address shortly. After θ_{reach_a} is computed, though, the pressure to choose an orientation that minimizes the possibility of collision imposes a natural optimization function. Our approach simultaneously searches for an orientation that maximizes distance to the nearest velocity obstacle (Definition 2) while attempting to only consider orientations inside θ_{reach_a}. It does so by discretizing the interval of reachable orientations initially without consideration of other agents, and then intelligently stepping through that interval. The pseudocode for our method is described by Algorithm 2.

Algorithm 2 discretizes the interval of reachable orientations (without consideration of other agents) by a user-specified resolution δ. Next, it attempts to step through reachable counter-clockwise orientations. If a tested orientation causes a collision when another agent reciprocally rotates, we've bounded our interval θ_{reach_a} on one side, and we do not attempt any more rotations in that direction (lines 6-16). An example of bounding an agent's orientations is displayed in Figure 4.

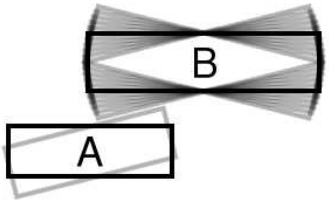

Figure 4: Agent A, represented as a black rectangle, bounds its maximum counter-clockwise rotation by assuming another agent, B may rotate at most as much as A. A discovers that a rotation of about 14° (light gray) is the maximum it can rotate such that it doesn't intersect the swept volume of B through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

Algorithm 2 Choose new orientation for agent a

Input: $a \in R$, time horizon τ, orientation resolution δ
Output: θ_{new_a}

1: $N_r \leftarrow$ RotationNeighbors(a, R)
2: $\Delta\theta_{max} \leftarrow \tau\omega_{max_a}$
3: $Lines \leftarrow [\emptyset][\emptyset]$
4: //Try CCW rotations
5: $\Delta\theta_a \leftarrow 0; collided \leftarrow false$
6: **while** $(\Delta\theta_a \leq \Delta\theta_{max} \wedge \neg collided)$ **do**
7: Rotate$(G_a, \Delta\theta_a)$
8: **for all** $b \in N_r$ **do**
9: $\Delta\theta_b \leftarrow -\Delta\theta_{max}$;
10: **while** $(\Delta\theta_b \leq \Delta\theta_{max} \wedge \neg collided)$ **do**
11: Rotate$(G_b, \Delta\theta_b)$
12: **if** $\neg$InCollision(G_a, G_b) **then**
13: $Lines[\Delta\theta_a] \leftarrow Lines[\Delta\theta_a] \cup$ ComputeLinearConstraint(b, τ)
14: **else**
15: $collided \leftarrow true$
16: $\Delta\theta_a \stackrel{+}{=} \frac{\Delta\theta_{max}}{2\delta}$
17: //Try CW rotations$\{\dots\}$
18: $\Delta\theta_{best} \leftarrow 0;\ u_{max} \leftarrow -\infty$
19: **for all** $ORCALines[\Delta\theta_i]\ l_i \in Lines$ **do**
20: $nextU \leftarrow$ ComputeMin(l_i)
21: **if** $nextU > u_{max}$ **then**
22: $u_{max} \leftarrow nextU$
23: $\Delta\theta_{best} \leftarrow \Delta\theta_i$
24: return $\Delta\theta_{best}$

If a rotation does not cause a collision, we compute a set of linear constraints for the potential orientations that the neighbor could take on, up to and including an equal rotation. We store them in *Lines*, a two-dimensional array indexed by a's potential orientations. Each constraint is computed by the subroutine ComputeLinearConstraint(). The subroutine computes a linear constraint by first finding the nearest point on $VO^{\tau}_{a|b}$ to $v^{rel}_{a|b}$, p_{near}, using Algorithm 3. The constraint itself is defined by the point $p_{near} - v^{rel}_{a|b}$, and has a direction along the line segment (or tangent line) of $VO^{\tau}_{a|b}$ upon which p_{near} is located.

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. As δ approaches infinity, though, the error in the approximation approaches zero. Empirically, we have found that even low values of δ (< 10) lead to very few collisions.

By line 18 we have computed a set of linear constraints for every potential orientation. Each of these sets will contain some linear constraint that has minimal distance from v_a. Recall that the variable u was used to indicate the vector from $v^{rel}_{a|b}$ to p_{near} in Definition 2. While vectors do not traditionally have negative magnitude, we consider u's magnitude as negative if $v^{rel}_{a|b} \in VO^{\tau}_{a|b}$. The

subroutine ComputeMin() computes the minimum u value for a set of linear constraints. (*Note: we exclude the definition of* ComputeMin() *because it is almost entirely composed of a series of distance checks.*) Each potential orientation is subsequently identified by its associated minimum length u vector from the resulting set of linear constraints.

By favoring the orientation that maximizes its associated minimum u vector, agent a chooses an orientation that maximize the probability that v_a already lies in the feasible region to the geometric linear program (lines 19-23). In less formal terms, agents choose rotations such that less work is required to modify their trajectories to avoid collision.

There are a few caveats to Algorithm 2. More than one rotation may maximize u. In this case, the orientation nearest to θ_{pref_a} is chosen, which could require solving multiple geometric linear programs following the example θ_{pref_a} discussed in Section 3. Also, because we do not allow agents to deduce others' maximum speeds, we are forced to approximate the set of rotation neighbors by using a conservative nearest neighbors check.

Algorithm 3 Compute nearest point to $v_{a|b}^{rel}$ on boundary of $VO_{a|b}^{\tau}$

Input: $VO_{a|b}^{\tau}$, $v_{a|b}^{rel}$
Output: p_{near}
1: // $VO_{a|b}^{\tau}$ consists of a set of line segments and tangent rays t_{left} and t_{right}
2: $nearTangent \leftarrow t_{left}$
3: **if** ($||$ProjectVector($v_{a|b}^{rel}$,t_{right})$-v_{a|b}^{rel}|| < ||$ProjectVector($v_{a|b}^{rel}$,t_{left})$-v_{a|b}^{rel}||$) **then**
4: $\quad nearTangent \leftarrow t_{right}$
5: $segmentProjection \leftarrow (\infty, \infty)$
6: **for all** Segments $l \in VO_{a|b}^{\tau}$ **do**
7: $\quad segmentVec \leftarrow (l.target - l.source)$
8: $\quad relativeVel \leftarrow (v_{a|b}^{rel} - l.source)$
9: $\quad nextProjection \leftarrow$ ProjectVector($relativeVel, segmentVec$)
10: $\quad$ **if** ($||nextProjection - v_{a|b}^{rel}|| < ||segmentProjection - v_{a|b}^{rel}|| \wedge |nextPr\hat{o}jection \times \hat{v_{a|b}^{rel}}| > 0$) **then**
11: $\quad\quad segmentProjection \leftarrow nextProjection$
12: **if** ($||segmentProjection - v_{a|b}^{rel}|| < ||tangentProjection - v_{a|b}^{rel}||$) **then**
13: $\quad p_{near} \leftarrow segmentProjection$
14: **else**
15: $\quad p_{near} \leftarrow tangentProjection$

The pseudocode for actually computing p_{near} in the polygonal case is shown in Algorithm 3. Algorithm 3 makes use of a subroutine ProjectVector, which is a stand-in for the computation of one vector's projection onto another. We abuse its notation slightly when projecting onto the tangent rays that bound $VO_{a|b}^{\tau}$. Because t_{left} and t_{right} are points of tangency on $G_a \oplus G_b$ from the origin, the rays actually begin at these points and continue indefinitely. Therefore, the projection of $v_{a|b}^{rel}$onto these rays is a projection onto the portion of the ray on or after its initial point (t_{left} and t_{right}); $v_{a|b}^{rel}$ is never projected between the origin and the initial point.

Algorithm 3 first computes a projection onto the nearest tangent ray to $v_{a|b}^{rel}$ (lines 2-4). Next, it computes a projection onto the nearest line segment from $G_a \oplus G_b$ that bounds $VO_{a|b}^{\tau}$ (lines 5-11). We treat line segments as pairs of points, *source* and *target*, from which we can create a vector via subtraction. Finally, we choose p_{near} to be the projection onto the boundary of $VO_{a|b}^{\tau}$ that is nearest to $v_{a|b}^{rel}$ (lines 12-15). Figure 2(b) demonstrates the case where p_{near} is a point on one of the

line segments bounding $VO^{\tau}_{a|b}$.

Of note is that, when computing the nearest projection of $v^{rel}_{a|b}$ onto the line segments bounding $VO^{\tau}_{a|b}$, we ensure that the vector from the projected point to the origin is not collinear with the relative velocity via testing the length of the cross product between vectors in the direction of the projected point and $v^{rel}_{a|b}$ (line 10). Such a test enables RRVO to avoid the deadlocking scenario between two agents described in Figure 3. While it helps to avoid deadlocking by forcing a projection onto a different line segment or a tangent ray in the worst case, it cannot guarantee that both agents will choose the same direction around their mutual velocity obstacles, and therefore alone cannot guarantee that agents will not deadlock. When the technique is coupled with rotation, however, the two strategies are very effective at removing deadlock. The authors observed anecdotally that such an optimization without also permitting rotation did not incur nearly the same reduction in deadlock.

5.2 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 with the exception that they need not assume obstacles will rotate. 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 RVO 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 an approaches in this paper.

5.3 Collisions

Despite an agent's best effort to avoid collision, a feasible solution to the linear program is not always guaranteed to 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. Consequently, a linear constraint cannot be constructed based on the distance from $v^{rel}_{a|b}$ to $VO^{\tau}_{a|b}$. Instead, we choose to construct a linear constraint based on the distance from p_a to the nearest point on M instead. Such a constraint encourages agents to choose velocities that escape collisions.

Algorithm 2 as written will terminate without testing any orientations at all due to the fact that the agent is already in collision (line 6). Therefore, we relax the requirement that the algorithm ceases searching for orientations upon discovering a colliding orientation. This has the effect of permitting the agent to search the full interval of reachable orientations in the hopes of finding a collision-free one.

5.4 Complexity

The steps involved in selecting a new orientation and velocity involve computing nearest neighbors, computing velocity obstacles for each neighbor, and then solving a geometric linear program.

In Reciprocal Velocity Obstacles, finding one's k-nearest neighbors is performed in $O(k \log n)$ time in the average case 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 due to the use of circles, and

solving the geometric linear program involves satisfying k constraints, one for each neighbor. As mentioned in Section 4.1, 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 RVO 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 logarithmic in the 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 test, we need to compute a "worst-case" $\Delta\theta_b$. This involves testing a's rotation against all reachable orientations of b up to $\pm\Delta\theta_a$ (line 9 in Algorithm 2). Because this set is discretized by δ, we must compute $O(\delta^2)$ potential velocity obstacles for each neighbor. The theoretical complexity RRVO incurs is therefore $O(\delta^2 \textsc{Max}(||G_i||))$. Because δ is a constant, though, its term drops from the complexity, making it $O(\textsc{Max}(||G_i||))$.

We can further improve our complexity by considering our choice of polygons to represent agents. We've already decided to use convex polygons to represent agents so that Minkowski sums can be performed in linear time, but if we choose to represent *all* agents with the same kind of convex polygon (e.g., a rectangle), then the Minkowski operation essentially becomes constant time. For example, the Minkowski sum between two rectangles results in at most 8 vertex additions.

Although we can implement RRVO such that it only adds a constant factor to the Big-O complexity of RVO, the practical overhead can be significant. However, with this increase in work also comes increased room for parallelism. We anticipate that higher core counts or GPU approaches can reduce this practical overhead to a negligible amount, and plan to explore this avenue in future work.

6 Experimental Results

In this section we compare how RRVO performs against RVO in terms of eliminating deadlock by getting agents to their goals in fewer frames, or timesteps of the simulation. We also compare their performance in terms of frames computed per second (framerate). We show that RRVO does not suffer from deadlock, although it suffers a performance degradation that is experimentally linear in the parameter δ, which discretizes the set of reachable orientations for each agent.

6.1 Experimental Setup

We implemented RRVO in C++ by adapting the RVO2 library [27]. The library had existing support for coarse OpenMP parallelization which we retained, although it by no means represents the limits of RRVO's scalability. Experiments were run on a quad-core Intel i5-2520M system with 4 GB of memory running Ubuntu 12.04.

We were interested in observing how well RRVO mitigates deadlock, and what trade-offs to performance must be made to achieve such mitigation. Accordingly, we chose the antipodal circle scenario shown in Figure 5 due to the high probability of deadlock for polygonal agents. In the scenario, agents are distributed evenly around a circle, and then instructed to reach the location directly opposite. This scenario causes extreme crowd density near the middle of the circle.

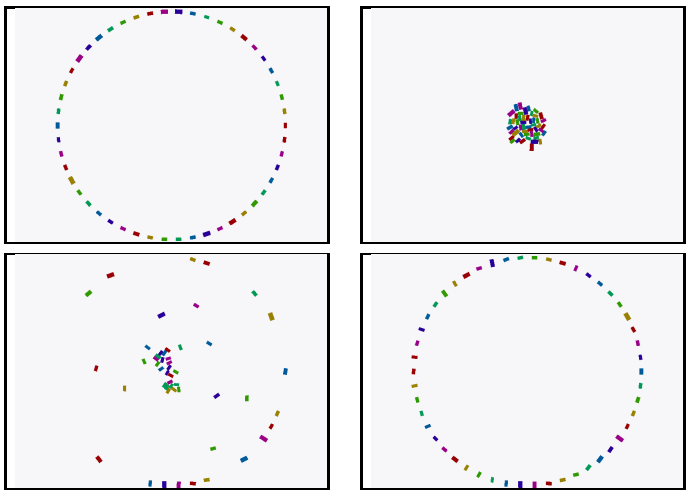

Figure 5: Progression of the antipodal circle scenario for 50 agents. Agents are positioned evenly around a circle and directed to reach their antipodal position. Congestion forms in the middle of the circle, but is eventually resolved, and the agents reach their goals.

For all trials, we used 50 rectangular agents evenly spaced on the circle. 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.

6.2 Metrics

Our metrics included average framerate (FPS) normalized by the framerate of RVO, and the average number of frames it took each agent to arrive at its goal. We varied the value of δ (the number of orientations for an agent to consider per timestep) to observe its effect on both of these metrics. To account for deadlock potentially skewing the frames-to-arrival metric, we capped each scenario at 20,000 frames. Each timing experiment (FPS) was repeated for 40 trials, while the frames-to-finish experiments were repeated for 20 trials.

6.3 Results

The results showed that RRVO enables agents in the antipodal circle scenario to avoid deadlocking. Concordantly with our analysis of complexity, we also found that RRVO's runtime increases the value of δ.

We recorded the frames-to-completion average for each value of δ used in the antipodal circle scenario in Table 1. When we use polygons with long parallel sides like rectangles, RVO suffers from a severe amount of deadlock; not a single trial of the antipodal circle was able to complete. On the other hand, because RRVO enables agents to rotate, they resolved deadlock, even for small values of δ, in all trials save one.

Because the parameter δ also acts as a resolution parameter to computing worst-case reciprocal rotation, using a lower value of δ will cause the agent to underestimate the bounds on collision-free continuous rotations more often, as discussed in Section 5. Such underestimation can cause an agent to choose orientations with less clearance. However, the values in Table 1 reinforce the idea that choosing orientations with more clearance by using higher values of δ does not always result in shorter overall paths.

δ	Avg Frames
0 (RVO)	–
2	6261.71
4	5232.62
8	5554.58
16	6526.27
32	5643.98

Table 1: Average frames until all agents reach their goals

The timing results shown in Figure 6, demonstrate that larger values of δ result in a significant decrease in performance, with $\delta = 2$ resulting in roughly a 7x slowdown, and a $\delta = 32$ resulting in a 400x slowdown. These results are optimistic when one considers that the theory predicts a slowdown of 1024x when δ is 32. This is likely due to the fact that agents do not always search the full interval of reachable orientations before discovering a potential collision. For $\delta = 2$, we suspect that the additional bookkeeping involved in RRVO contributed to slowdown beyond the predicted amount. Still, even a 7x slowdown in framerate may be preferred when the alternative is deadlock.

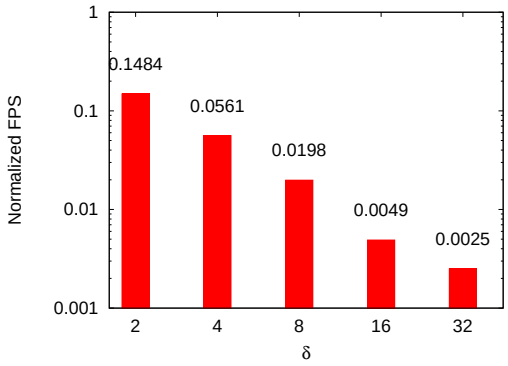

Figure 6: RRVO's FPS normalized against RVO's. Even a small value of δ incurs a significant amount of slowdown, although the slowdown is closer to linear than quadratic. *Note: the plot is a log-log scale.*

7 Conclusion

In this work we introduce Reciprocally-Rotating Velocity Obstacles, a generalization of Reciprocal Velocity Obstacles for local collision avoidance. RRVO enables better modeling of agents that allows for rotation, and helps mitigate potential deadlock scenarios that arise when using polygonal agents instead of discs. We present the theory of RRVO, analyze its complexity, and offer some optimizations to minimize practical overhead. Our results show that even a little bit of rotation results in much less deadlock, and that the performance overhead RRVO incurs is closer to linear than quadratic in the parameter δ.

Acknowledgments

We would like to thank the GAMMA group at UNC Chapel Hill for helping us understand and use their open-source RVO2 library. 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] I. Benenson. Multi-agent simulations of residential dynamics in the city. *Computers, Environment and Urban Systems*, 22(1):25–42, 1998.

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

[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 *Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242. Eurographics Association, 2004.

[5] M. de Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf. *Computational Geometry*. Springer, second edition, 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] D. Helbing and P. Molnar. Social force model for pedestrian dynamics. *Physical review E*, 51(5):4282, 1995.

[9] 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.

[10] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2719–2726. IEEE, 1997.

[11] 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.

[12] 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.

[13] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, 1996.

[14] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

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

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

[17] D. Massaguer, V. Balasubramanian, S. Mehrotra, and N. Venkatasubramanian. Multi-agent simulation of disaster response. In *ATDM workshop in AAMAS*, volume 2006. Citeseer, 2006.

[18] R. Narain, A. Golas, S. Curtis, and M. 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.

[19] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM SIGGRAPH Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[20] M. Riedl, C. J. Saretto, and R. M. 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.

[21] S. Rodriguez, A. Giese, N. M. Amato, S. Zarrinmehr, F. Al-Douri, and M. J. Clayton. Environmental effect on egress simulation. In *Motion in Games*, pages 7–18. Springer, 2012.

[22] 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.

[23] W. Shao and D. Terzopoulos. Autonomous pedestrians. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 19–28. ACM, 2005.

[24] 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.

[25] A. Treuille, S. Cooper, and Z. Popović. Continuum crowds. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 1160–1168. ACM, 2006.

[26] 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.

[27] 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.

[28] 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.