Evaluating Preferred Velocity Methods Used with ORCA

Mollie Bianchi

Abstract

The ORCA method [1] is a procedure for finding the closest collision-free velocity to a preferred velocity for each robot in a multi-robot environment. The computation of this preferred velocity is separate from the collision avoidance and is generally achieved using traditional motion planning techniques. The only restriction is that it must be computed quickly as the collision avoidance procedure is computation heavy. Exactly how it was computed was excluded from the original paper. The preferred velocity is important in order to prevent deadlock situations and to ensure smooth paths. This project implements the ORCA method from scratch in MATLAB and presents and evaluates a method for computing the preferred velocity.

I. INTRODUCTION

Essential to robotics is collision avoidance, especially when multiple robots are interacting in a shared environment. This could include mobile robots in a warehouse, aerial robots performing acrobatic displays, or swarming robots. This project looks at the problem of avoiding collisions between multiple robots, or the reciprocal n-body collision avoidance problem.

The first decentralized method to guarantee local collision-free motion for a large number of robots in a cluttered work space was the ORCA method [1]. It was developed in 2011 by Jur van den Berg et al at the University of North Carolina at Chapel Hill. It is a decentralized control method, that is each robot decides for itself how it should behave based on its observations of the world. The original paper deal with holonomic, two-dimensional robots. It has since been adapted to higher dimensions for unmanned aerial vehicles [16], non-holonomic vehicles [7], for artistic pattern formation, [13] [15]. It has been demonstrated in simulations with thousands of robots and still resulted in collision-free trajectories.

The ORCA method works by using velocity obstacles to constrain a robot's permitted velocity. It then finds the closest velocity in free velocity space to a robot's preferred velocity. This preferred velocity can have a significant impact on the smoothness, length, and feasibility of the resulting path. It is important for avoiding deadlock scenarios. This project implemented the ORCA method from scratch in MATLAB and evaluates a proposed method for computing the preferred velocity.

II. BACKGROUND

The ORCA procedure starts by defining a velocity obstacle, $VO^{\tau}_{A|B}$, as the set of all relative velocities of robot A with respect to robot B that will result in a collision between robot A and robot B at some time before time τ. This is defined as:

$$VO^{\tau}_{A|B} = \{\mathbf{v} | \exists t \in [0, \tau] :: t\mathbf{v}| \ ||t\mathbf{v} - (\mathbf{p}_B - \mathbf{p}_A)|| < (r_A + r_B)\} \tag{1}$$

Graphically, this looks like a truncated cone with its apex at the origin and its legs tangent to the circle of radius $(r_A + r_B)/\tau$ centered at $(\mathbf{p}_B - \mathbf{p}_A)/\tau$. The cone is truncated by this circle. See Figure 1.

In order to avoid a collision between robot A and robot B, the relative velocity needs to be outside of $VO^{\tau}_{A|B}$. Let $\mathbf{u}$ be the vector from $\mathbf{v}^{\text{cur}}_A - \mathbf{v}^{\text{cur}}_B$ to the closest point on the boundary of the $VO^{\tau}_{A|B}$:

$$\mathbf{u} = \underset{\mathbf{v} \in \text{bd}(VO^{\tau}_{A|B})}{\text{argmin}} ||\mathbf{v} - (\mathbf{v}^{\text{cur}}_A - \mathbf{v}^{\text{cur}}_B)|| - (\mathbf{v}^{\text{cur}}_A - \mathbf{v}^{\text{cur}}_B) \tag{2}$$

and let $\mathbf{n}$ be the outward normal of the $VO^{\tau}_{A|B}$ boundary at the closest point. If the relative velocity, $\mathbf{v}^{\text{cur}}_A - \mathbf{v}^{\text{cur}}_B$, is already inside $VO^{\tau}_{A|B}$, the value $\mathbf{u} \cdot \mathbf{n}$ is the minimum amount the relative velocity needs to change by in order to avoid a collision. If the relative velocity, $\mathbf{v}^{\text{cur}}_A - \mathbf{v}^{\text{cur}}_B$, is outside $VO^{\tau}_{A|B}$, then $\mathbf{u} \cdot \mathbf{n}$ is the maximum amount the relative velocity can change by before a collision occurs. Since both robots follow the same strategy, they can share the responsibility. The constraint on each robot is then only $\frac{1}{2}\mathbf{u} \cdot \mathbf{n}$. The permitted velocity for robot A is then constrained to the half-plane pointing in the direction $\mathbf{n}$ starting at the point $\mathbf{v}^{\text{cur}}_A + \frac{1}{2}\mathbf{u}$:

$$ORCA^{\tau}_{A|B} = \{\mathbf{v} | (\mathbf{v} - (\mathbf{v}^{\text{cur}}_A + \frac{1}{2}\mathbf{u})) \cdot \mathbf{n} \geq 0\} \tag{3}$$

This set is noted as $ORCA^{\tau}_{A|B}$.

Robot A needs to avoid collision with all other robots. Its velocity is constrained by multiple half-planes. The set of velocities that are permitted for robot A with respect to all robots is the intersection of all these half-planes:

$$ORCA_A^\tau = \bigcap_{B \neq A} ORCA_{A|B}^\tau \tag{4}$$

The new velocity that robot A selects for itself is then:

$$\mathbf{v}_A^{\text{new}} = \underset{\mathbf{v} \in ORCA_A^\tau}{\text{argmin}} ||\mathbf{v} - \mathbf{v}_A^{\text{pref}}|| \tag{5}$$

The n-body reciprocal collision avoidance method finds the velocity that is closest to the robots preferred velocity while also guaranteeing a collision free trajectory for at least time τ.

The preferred velocity in the original ORCA paper [1] is defined as the velocity which the robot would assume had no other robots been in its way. For instance, a velocity directed towards the robot's goal with a magnitude equal to the robot's preferred speed. Early in testing for this project, it was discovered that this "naive" preferred velocity could lead to deadlock situations. For example, two robots swapping positions will have a preferred velocity that goes directly through the other robot. Since the ORCA method finds the closest velocity that is also collision free, given a perfect solver this collision free velocity would eventually converge to 0. This is somewhat analogous to two people heading towards each other and not knowing to which side they should move in order to go around the other person. There needs to be some mechanism so both robots or people do not go to the same side. Unlike the analogy with people, we want to solve this problem without the robots being able to communicate with each other. This project proposes a method to do so and evaluates its performance on five scenarios.

III. Literature Review

A. Collision Avoidance Methods

The most basic form of collision avoidance is to consider all obstacles as static. The robot can then plan a path around this obstacle. For example, [6] accomplishes this by using a vector field histogram and [4] uses a dynamic window approach to search for the best collision-free velocity that minimizes an objective function. If the obstacle is moving, these methods could be repeated so that the robot will continually replan based on the new position of the obstacle. This approach works when the obstacles are moving much slower than the robot. A slightly more sophisticated approach is known as "asteroid avoidance". In this case, the velocity of the obstacle is used to extrapolate its position in the future which the robot then plans around [5].

Velocity obstacles were introduced in [2]. This work presented a method for avoiding obstacles in the velocity space which would yield collision free trajectories. Reciprocal collision avoidance is the idea that the other obstacles are moving intelligently around the environment. Reciprocal Velocity Obstacles were a natural extension to account for robot to robot collision avoidance [3]. This formulation only guarantees collision avoidance under certain conditions. ORCA was developed to address these limitations.

B. Adaptation and Applications of the ORCA Method

The original ORCA paper used holonomic, 2D vehicles in simulation. It has since been adapted to NH-ORCA for non-holonomic, differential drive robots [7] and B-ORCA specifically for bicycle model vehicles [8]. These adaptations involve solving the original ORCA problem and mapping the optimal holonomic velocity to non-holonomic control inputs. ORCA has also been adapted to higher dimensions for use on UAVs [16]. The algorithm scales to higher dimensions quite easily and additional constraints to account for UAV dynamics were added to improve performance.

Learning has also been used to solve the reciprocal collision avoidance problem. [10] takes a system with multiple nonlinear agents and breaks it into multiple two body problems. A model is then trained to find collision free velocities for the two body problem. When this trained policy is then applied to the real agents it is used in addition to ORCA. The constraints from ORCA ensure that the velocity sent to the agent is collision free.

The ORCA method has been used in multiple artistic applications [15], [13] where robots are rearranged to form specific patterns. Ideally, the paths the robots follow should be smooth so that they are aesthetically pleasing and collision free.

C. Methods for Computing the Preferred Velocity

In the original ORCA paper, there is no specification for how to compute the preferred velocity. Selecting a bad preferred velocity can lead to deadlock situations as noted in [14].Other works that have used the ORCA method have provided more details into this computation. [9] deals with decentralized segregated robotic swarm navigation. In this application, a large number of robots arrange themselves into groups based on common features and these groups remain intact as the robots move around to complete some task. They achieve this by combining ORCA with a modified version of the classical flocking behaviours. They compute the preferred velocity used by the ORCA algorithm for each robot by:

$$\mathbf{v}^{\text{pref}} = \alpha \mathbf{v}^{\text{goal}} + \beta \mathbf{v}^{\text{flock}} + \gamma \mathbf{v}^{\text{aux}} \tag{6}$$

where $\mathbf{v}^{\text{goal}}$ is an attractive velocity towards the robot's goal, $\mathbf{v}^{\text{flock}}$ is a flocking velocity, and $\mathbf{v}^{\text{aux}}$ is an auxiliary velocity. The coefficients, α, β, γ, are set according to the situation the robot is in. These situations are defined by a finite state machine, and include states such as when the robot only perceives robots within its group nearby, when the robot has a clear path to the goal, when a robot in its group has a clear path to the goal, and when robots from other groups are nearby.

[12] tackled the problem of deadlock with ORCA in highly congested scenarios by introducing a set of rules for each robot to follow. These rules were inspired by vehicular traffic rules. If a robot detects another robot on its right, it yields right of way. If a robot detects another robot in front of it and close by, it will pass it on the left. If a robot detects another robot approaching from behind it will slow down and allow that robot to pass.

The proposed method for computing the preferred velocity in this project was inspired by these two works. The main differences is that here the proposed method only uses a single rule, that is moving to the left if another robot is nearby (how nearby is defined is based on the method and explained in greater detail in Section **??**, and that unlike [9] a complicated finite state machine is not required to set the coefficients, and unlike [12] none the robots are instructed to stop and yield the right of way.

Another method for choosing the preferred velocity is a reinforcement learning based method, ALAN [11]. It has a set of preferred velocities an agent can choose from and a reward function the agent uses to evaluate velocities.

IV. Methodology

A. System Overview

The implementation of ORCA and the preferred velocity strategies were written from scratch in MATLAB. The code can be found at `https://github.com/MollieBianchi/planning_project`.

This project models circular, holonomic robots moving in two dimensions. Each robot is modelled by a current position, $\mathbf{p}^{\text{cur}}$, a current velocity, $\mathbf{v}^{\text{cur}}$, and a radius, r. These parameters can be viewed by the other robots. Each robot also has a maximum speed, v^{max}, and a preferred velocity, $\mathbf{v}^{\text{pref}}$, that are not observable by the other robots.

The full algorithm is shown in Algorithm 1. There are two main components. The first *for* loop corresponds with the ORCA method which finds the collision free velocity closest to the preferred velocity for each robot. The second component is updating all the robots' positions and velocities and reevaluating the robots' preferred

velocities.

Algorithm 1: Collision Avoidance

```
while not all robots at goal position do
    for A ∈ robots do
        half_planes = [ ] ;
        normals = [ ] ;
        for B ∈ robots \ A do
            u, n = find_halfplane(A, B) ;
            half_planes.append(u) ;
            normals.append(n) ;
        end
        solve optimization problem for v_A^new
    end
    for A ∈ robots do
        v_A^cur ⟵ v_A^new ;
        v_A^pref ⟵ p_A^cur + v_A^new Δt ;
    end
    for A ∈ robots do
        v_A^pref ⟵ get_preferred_velocity(A, robots) ;
    end
end
```

B. Define Velocity Obstacles

In order to solve for the collision free velocity for robot A, we first need to define the velocity obstacle for robot A and robot B with look ahead time, τ, noted as $VO^{\tau}_{A|B}$. As detailed in Section II, $VO^{\tau}_{A|B}$ is a truncated cone with apex at the origin. It is truncated by an arc of a circle of radius $(r_A + r_B)/\tau$ centered at $(\mathbf{p}_B^{\text{cur}} - \mathbf{p}_A^{\text{cur}})/\tau$ and the legs are tangent to this circle. To solve for the equation for each of the legs, note that the equations will be of the form:

$$v_y = k_i v_x \tag{7}$$

since the line intercepts the origin. The equation for the circle is

$$(v_x - c_x)^2 + (v_y - c_y)^2 = R^2 \tag{8}$$

where c_x and c_y are the x and y components of $(\mathbf{p}_B^{\text{cur}} - \mathbf{p}_A^{\text{cur}})/\tau$ and $R = (r_A + r_B)/\tau$. To solve for the legs tangent to the circle, plug (7) into (8):

$$(v_x - c_x)^2 + (kv_x - c_y)^2 = R^2 \tag{9}$$

which can be expanded and re-grouped as follows:

$$\underbrace{(1 + k^2)}_{A} v_x^2 + \underbrace{(-2c_x - 2c_y k)}_{B} v_x + \underbrace{c_x^2 + c_y^2 - R^2}_{C} = 0 \tag{10}$$

Since we want the leg to be tangent to the circle, there should only be a single solution for the interception points. In other words, the discriminant of the quadratic equation should be equal to 0:

$$0 = B^2 - 4AC \tag{11}$$

$$0 = (-2c_x - 2c_y k)^2 - 4(1 + k^2)(c_x^2 + c_y^2 - R^2) \tag{12}$$

$$0 = (4R^2 - 43c_x^2)k^2 + 8c_x c_y k + 4R - 4c_y^2 \tag{13}$$

Solving this quadratic equation gives two values for k, one for each of the legs. These k values can then be used to solve for the tangent interception points, $\mathbf{p}_i^{\text{tangent}}$. The boundary of $VO^{\tau}_{A|B}$ can then be described by $v_y = k_1 v_x$ and $v_y = k_2 v_x$ when $|(v_x, v_y)| > |\mathbf{p}_i^{\text{tangent}}|$ and by $(v_x - c_x)^2 + (v_y - c_y)^2 = R^2$ when $|(v_x, v_y)| <= |\mathbf{p}_i^{\text{tangent}}|$. See Figure 1.

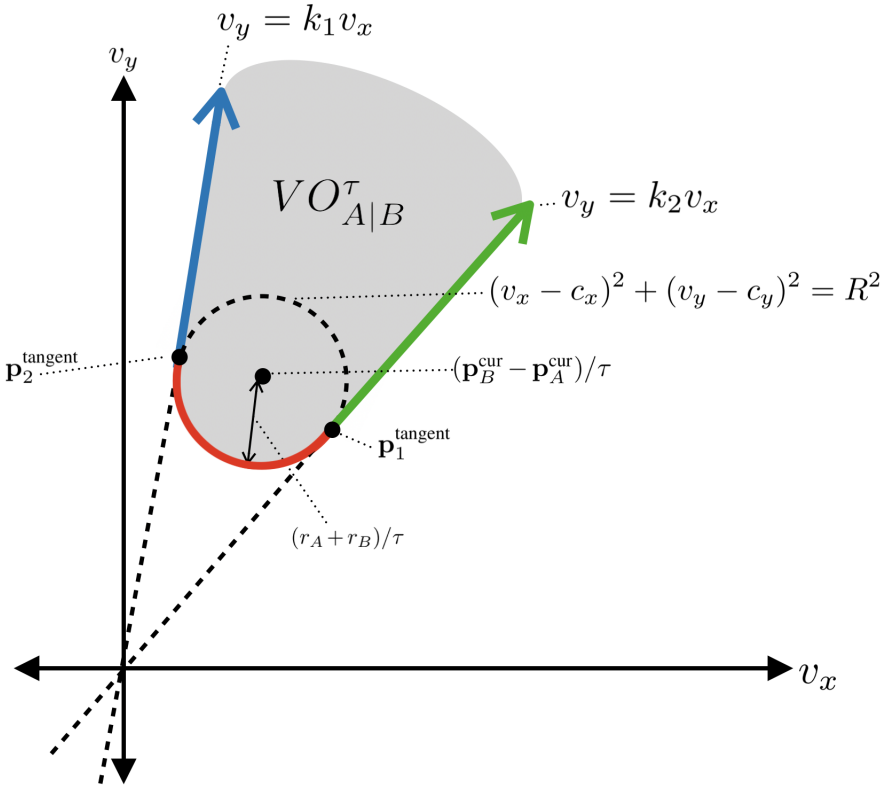


Fig. 1: The velocity obstacle boundary is described by the three segments shown in blue, red, and green.

C. Finding Half-Plane Constraints

We now want to solve for the vector pointing from $\mathbf{v}_A^{\text{cur}} - \mathbf{v}_B^{\text{cur}}$ to the nearest point on the $VO^{\tau}_{A|B}$ boundary, that is we want to solve the following:

$$\mathbf{u} = \mathbf{v} - (\mathbf{v}_A^{\text{cur}} - \mathbf{v}_B^{\text{cur}}) \tag{14}$$

where

$$\mathbf{v} = \underset{\mathbf{v} \in \text{bd}(VO^{\tau}_{A|B})}{\text{argmin}} ||\mathbf{v} - (\mathbf{v}_A^{\text{cur}} - \mathbf{v}_B^{\text{cur}})|| \tag{15}$$

Solving for $\mathbf{u}$ requires solving an optimization for $\mathbf{v}$. Finding $\mathbf{v}$ using a single optimization would require a piecewise constraint function to represent the boundary condition which is complicated to implement in MATLAB. Instead three optimization problems, one for each segment of the boundary, are solved. Finding the closest point on the legs of $VO^{\tau}_{A|B}$ means solving:

$$\mathbf{v}_i^{\text{leg}} = \underset{\mathbf{v}^{\text{leg}}}{\text{argmin}} \ \frac{1}{2} \mathbf{v}_i^{\text{leg}T} \mathbf{v}_i^{\text{leg}} - (\mathbf{v}_A^{\text{cur}} - \mathbf{v}_B^{\text{cur}})^T \mathbf{v}_i^{\text{leg}} \tag{16}$$

subject to:

$$[-k_i \ 1] \mathbf{v}_i^{\text{leg}} = 0 \tag{17}$$

$$||\mathbf{v}_i^{\text{leg}}|| > ||\mathbf{p}_i^{\text{tangent}}|| \tag{18}$$

If we remove the second constraint, MATLAB's `quadprog` function can be used to solve this since the objective function is quadratic and the constraint is linear. The solution $\mathbf{v}_i^{\text{leg}}$ can then be checked to see if it is valid. If $||\mathbf{v}_i^{\text{leg}}|| < ||\mathbf{p}_i^{\text{tangent}}||$, this solution can be discarded because the closest $\mathbf{v}$ on the boundary will be on the circular segment.

The third optimization problem solves for the closest point $\mathbf{v}^{\text{circle}}$ on the circular segment. This means solving the following optimization problem:

$$\mathbf{v}^{\text{circle}} = \underset{\mathbf{v}^{\text{circle}}}{\text{argmin}} \ \frac{1}{2}\mathbf{v}^{\text{circle}T}\mathbf{v}^{\text{circle}} - (\mathbf{v}_A^{\text{cur}} - \mathbf{v}_B^{\text{cur}})^T\mathbf{v}^{\text{circle}} \tag{19}$$

subject to:

$$(\mathbf{v}_x^{\text{circle}} - c_x)^2 + (\mathbf{v}_y^{\text{circle}} - c_y)^2 = R^2 \tag{20}$$

$$||\mathbf{v}^{\text{circle}}|| < ||\mathbf{p}_i^{\text{tangent}}|| \tag{21}$$

Since this has nonlinear objectives and constraints, MATLAB's `fmincon` solver needs to be used. After eliminating any invalid points from the leg optimizations, the closest point from the candidates is selected and $\mathbf{u}$ is computed according to (14).

Care needs to be taken when computing the outward normal, $\mathbf{n}$. When $\mathbf{v}_A^{\text{cur}} - \mathbf{v}_B^{\text{cur}} \notin VO_{A|B}^{\tau}$, the outward normal will be facing in the opposite direction of $\mathbf{u}$, otherwise it will point in the same direction.

Using $\mathbf{u}$ and $\mathbf{n}$, the half-plane of permitted velocities for robot A with respect to robot be is defined as:

$$ORCA_{A|B}^{\tau} = \{\mathbf{v}|(\mathbf{v} - (\mathbf{v}_A^{\text{cur}} + \frac{1}{2}\mathbf{u})) \cdot \mathbf{n} \geq 0\} \tag{22}$$

Solving for $\mathbf{u}$ and $\mathbf{n}$ is repeated for robot A with all other robots. Then the set of permitted velocities for robot A with respect to all the other robots is the intersection of all the half-planes:

$$ORCA_A^{\tau} = \bigcap_{B \neq A} ORCA_{A|B}^{\tau} \tag{23}$$

This allowable set is then used as the constraint when optimizing the new velocity for robot A.

D. Solve Optimization for $\mathbf{v}^{new}$

To find the new velocity for robot A, the following optimization needs to be solved:

$$\mathbf{v}_A^{\text{new}} = \underset{\mathbf{v} \in ORCA_A^{\tau}}{\text{argmin}} \ ||\mathbf{v} - \mathbf{v}_A^{\text{pref}}|| \tag{24}$$

where $\mathbf{v} \in ORCA_A^{\tau}$ was defined in (23). This $\mathbf{v} \in ORCA_A^{\tau}$ constraint can be written in a linear form:

$$\begin{bmatrix} -\mathbf{n}_1^T \\ -\mathbf{n}_2^T \\ \vdots \\ -\mathbf{n}_{N-1}^T \end{bmatrix} \mathbf{v} \leq \begin{bmatrix} -(\mathbf{v}_A^{\text{cur}} + \frac{1}{2}\mathbf{u}_1)^T\mathbf{n}_1 \\ -(\mathbf{v}_A^{\text{cur}} + \frac{1}{2}\mathbf{u}_1)^T\mathbf{n}_1 \\ \vdots \\ -(\mathbf{v}_A^{\text{cur}} + \frac{1}{2}\mathbf{u}_{N-1})^T\mathbf{n}_{N-1} \end{bmatrix} \tag{25}$$

where N is the total number of robots. Since this optimization has a quadratic objective function and linear inequality constraints, it can be solved using MATLAB's `quadprog` function. The result of this optimization is the new velocity that will be assigned to the robot.

E. Update Step

Once all the new velocities for the robots have been found, the simulation is stepped forward one time step. To do so, the current velocity of each robot is changed to the new velocity. Each robot's position is them updated by adding $\mathbf{v}_A^{\text{new}}\Delta t$ to their current position. After all the positions have been updated, each robot decides on a new preferred velocity based on their current postion and the position of all the other robots.

F. Preferred Velocity Computation

The preferred velocity computation proposed here takes a simplified approach to the methods in [9] and [12]. Namely, if there is a nearby robot, nudge the preferred velocity slightly to the left. This way if multiple robots need to cross through a single point at the same time, the preferred velocity will lead to behaviour that mimics a traffic roundabout instead of a multi-way stop.

More formally, if the robot has reached its goal position, it gets assigned $\mathbf{v}^{\text{pref}} = 0$. Otherwise, the preferred velocity is computed according to:

$$\mathbf{v}^{\text{pref}} = \mathbf{v}^{\text{goal}} + \alpha\mathbf{v}^{\text{nudge}} \tag{26}$$

where

$$\mathbf{v}^{\text{goal}} = (\mathbf{p}^{\text{goal}} - \mathbf{p}^{\text{cur}})/\Delta t \tag{27}$$

$$\mathbf{v}^{\text{nudge}} = \begin{bmatrix} \cos(\pi/2) & -\sin(\pi/2) \\ \sin(\pi/2) & \cos(\pi/2) \end{bmatrix} \mathbf{v}^{\text{goal}} \tag{28}$$

and α is computed based on the selected method. $\mathbf{v}^{\text{nudge}}$ points perendicular and to the left of $\mathbf{v}^{\text{goal}}$.

There are five methods: *Direct*, *Right*, *Front-Right*, *Front*, and *360*. For the *Direct* method, $\alpha = 0$. This is the "naive" approach; the robot will always head directly towards the goal. Each of the other methods consider specific sectors around the ego robot. If there is another robot in the sector under consideration and within some distance, R, α is computed according to:

$$\alpha = 0.3(R - d) \tag{29}$$

where d is the measured distance to the closest robot in that sector, and otherwise $\alpha = 0$.

The most important sector for the robot is in front and to the right. If there are other robots behind the ego robot or already on the left hand side, less action is required to move out of the way. So the different sectors of interest come from this idea. Assigning the ego robot's forward direction in the direction of $\mathbf{v}^{\text{goal}}$, the *Right* method considers robots on the right hand side of the ego robot. This is indicated by the blue sector in Figure 2. Similarly, the *Front* method, considers robots in front of the ego robot, indicated in yellow. The *Right-Front* method, considers robots on the right hand side and in front of the robot, shown in pink, and the *360* method considers robots all around the ego robot, shown in green. The *360* method was included to see if it could promote smoother paths because the $\mathbf{v}^{\text{nudge}}$ is not suddenly removed when a nearby robot leaves the sector under consideration.

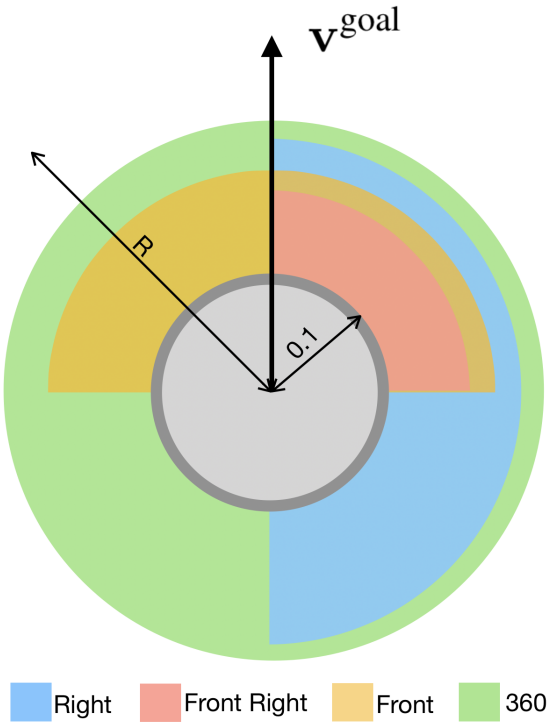


Fig. 2: The sectors under consideration for each of the preferred velocity methods.

The preferred velocity is also bounded by the maximum speed of the robots, which in these experiments was limited to 1m/s.

V. Experiments

There are five scenarios used to evaluate the preferred velocity strategy, as shown in Figure 3. *swap-2* involves two robots positioned 4m apart that are required to swap positions. *swap-3* and *swap-5* involve three or five robots evenly distributed around a circle of radius 2m that are required to travel to a position on the circle directly across

from their starting position. *swap-8* has 8 robots arranged evenly on a circle of radius 2m and are required to swap with positions with the robot directly across from them. *swap-8-3* is the same as *swap-8* but there are three additional robots on a concentric circle inside the larger circle. These three robots need to travel to a position on their circle directly across from their initial position.

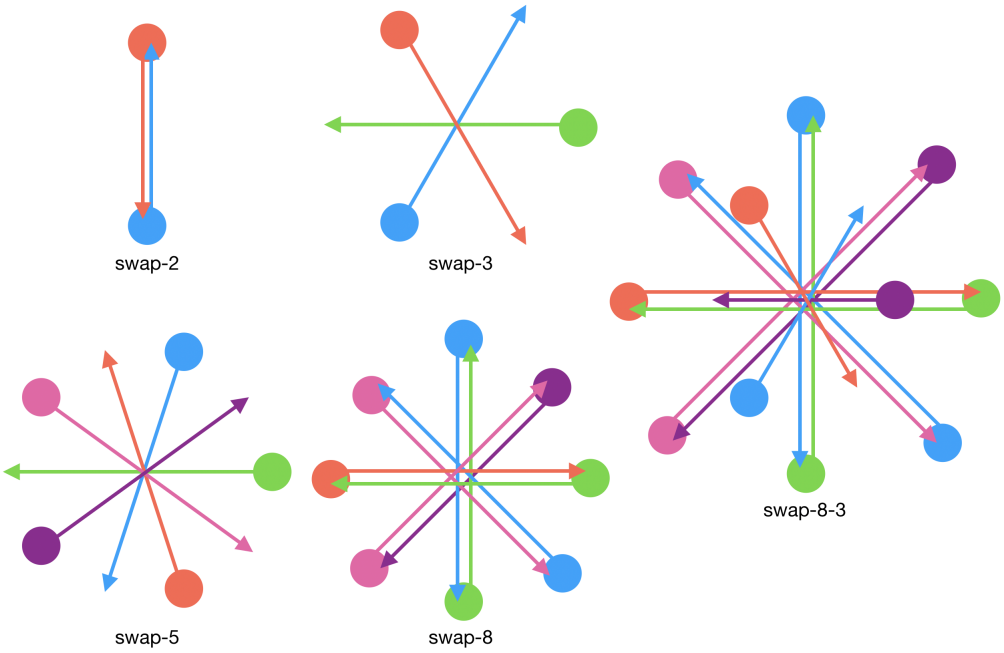


Fig. 3: The five scenarios used for evaluation.

The resulting trajectories are evaluated using three metrics: path length, roughness, and simulation time. The path length is defined as the sum of the distance travelled by each robot involved in the scenario. Ideally, we want the robots to take the shortest possible path, while maintaining a collision free trajectory. For certain applications of collision avoidance, such as artistic pattern formation, it is desired that the robots follow smooth paths. Therefore we want a metric that penalizes paths containing sharp changes or that look like multiple straight segments joined together, while rewarding paths with smooth, curved transitions and low amounts of jitter. So, the rate of change of curvature was selected as the roughness metric. This is computed as follows:

$$Roughness = \sum^{N}\sum^{T} \frac{x'y'' - y'x''}{(x'^2 + y'^2)^{3/2}} \tag{30}$$

where N is the number of robots, T is the simulation time, and the derivatives were approximated using numerical differences. The simulation time is the simulated amount of time required for all the robots to reach their goal positions. Overall, we desire trajectories that are short, smooth, and fast. If during a simulation the optimization fails or the robots reach a deadlock position (all velocities are zero), this is recorded as Did Not Finish (DNF).

The following parameter values were used for all simulations: $\tau = 0.5$ and $\Delta t = 0.05$, and for all robots, $r = 0.1m$ and $v^{\max} = 1m/s$.

VI. Results

The path length and smoothness results from the experiments are shown in charts included at the end of this section. The simulation times did not vary significantly between the methods but can be found in Tables 1-5 included in the Appendix. The *Direct* method and the *Right* and *Right-Front* methods with $R = 0.5$ were only able to complete the *swap-2* scenario. With all other scenarios, the robots became too close before altering their velocities and ended up in a deadlock. The *Front* and *360* methods with $R = 0.5m$ were able to complete the *swap-2* scenario and the *swap-5* scenario. Although, the *swap-5* scenario resulted in a significant increase in roughness and simulation time. This was because the robots got very close to each other before α became non-zero. They were

then only able to move at very slow speeds to ensure they would not hit each other. Interestingly, the *Front* and *360* methods were unable to complete the *swap-8* scenario with $R = 1m$ but the *Right-Front* and *Right* methods were able to complete it. None of the methods were able to complete the *swap-8-3* scenario with $R = 1m$. When $R \geq 2m$, all four methods were able to complete all the scenarios.

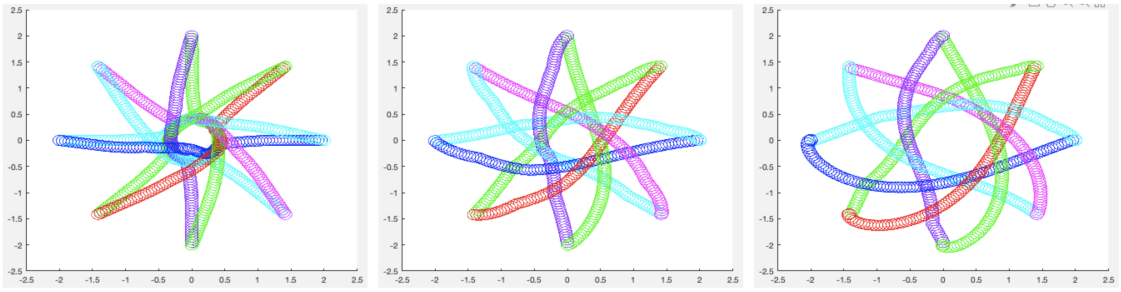


Fig. 4: Trajectories resulting from the *Right* method in the *swap-8* scenario with $R = 1m$ (Left), $R = 2m$ (Middle), and $R = 3m$ (Right).

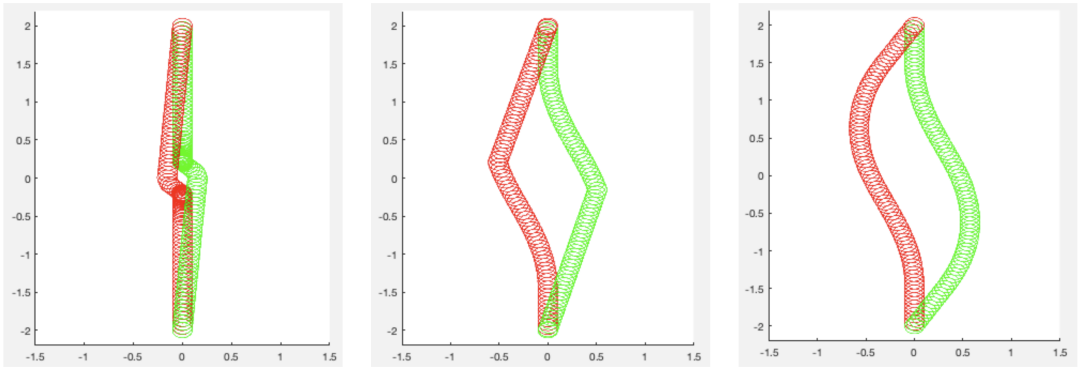


Fig. 5: Left: Trajectories resulting from the *Direct* method in the *swap-2* scenario. Middle: Trajectories resulting from the *Front* method with $R = 3m$ in the *swap-2* scenario. Right: Trajectories resulting from the *Right* method with $R = 3m$ in the *swap-2* scenario

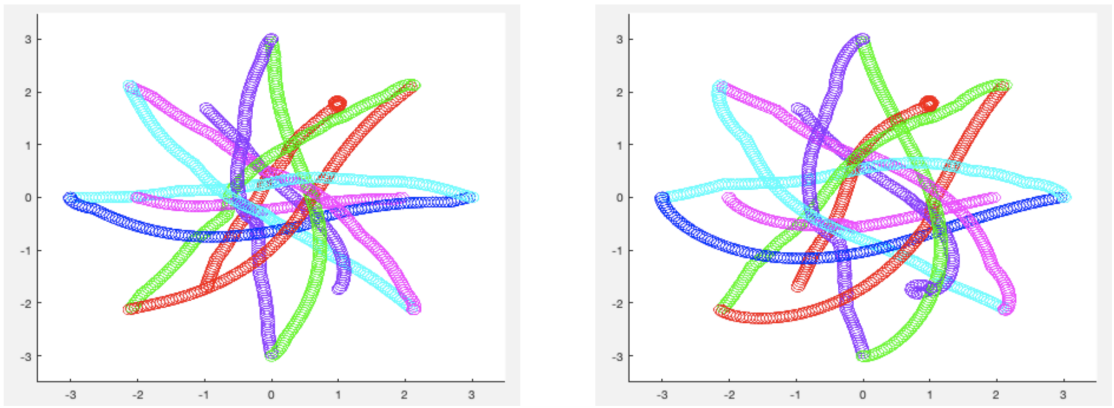


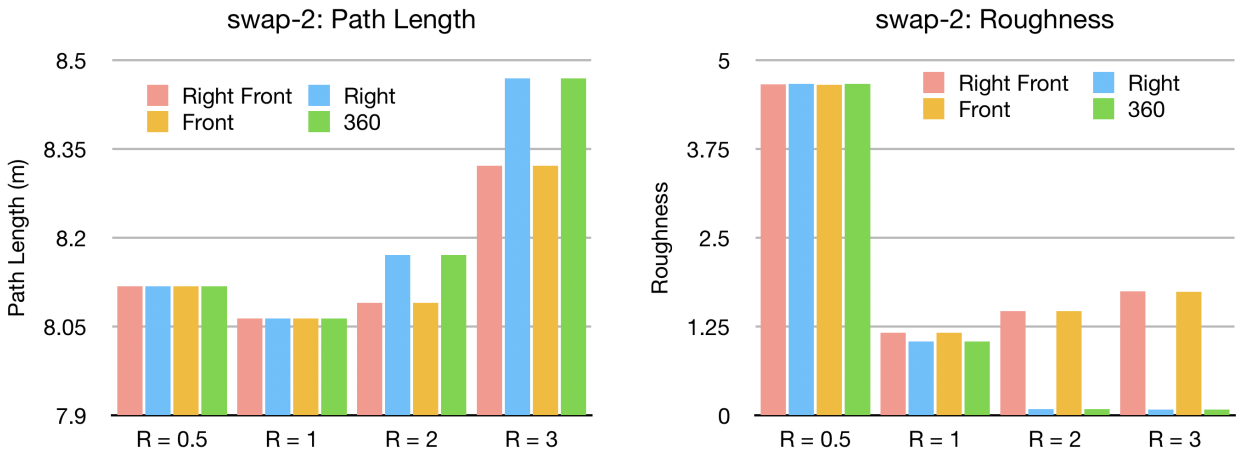
Fig. 6: Trajectories resulting from the *360* method in the *swap-8-3* scenario with $R = 2m$ (Left) and $R = 3m$ (Right).

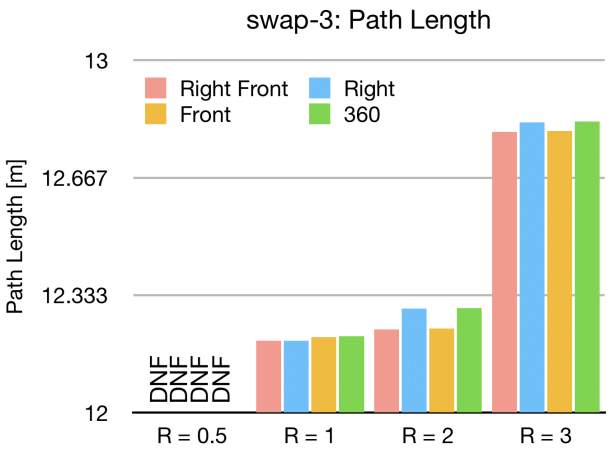
As would be expected, increasing R increased the path length but also decreased the roughness. Paths created with higher R values went further off the direct goal routes, but in doing so were able to generate smoother trajectories.

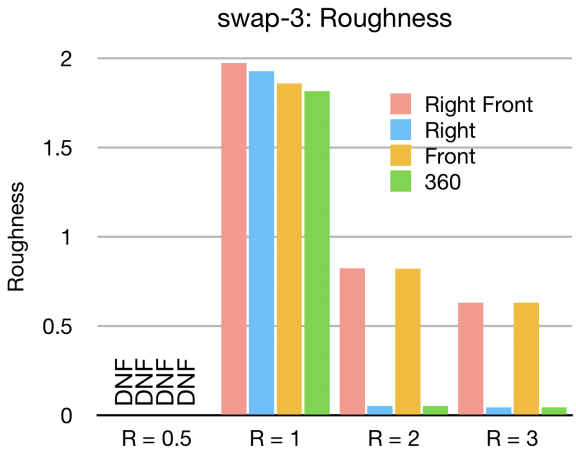
Higher R values lead to paths where the robots start deviating from the direct goal route quite early, but at a slow rate. These paths have lower curvature and a lower rate of change of curvature, but as a consequence they also have much higher path lengths. This can be seen in the trajectories shown in Figure 4. With $R = 1m$, the robots travel along the direct goal route for as long as possible and then make a sharp detour resulting in a small, tight roundabout circle. With $R = 2m$ and $R = 3m$, the robots start deviating from the path earlier which leads to larger, looser roundabout circles.

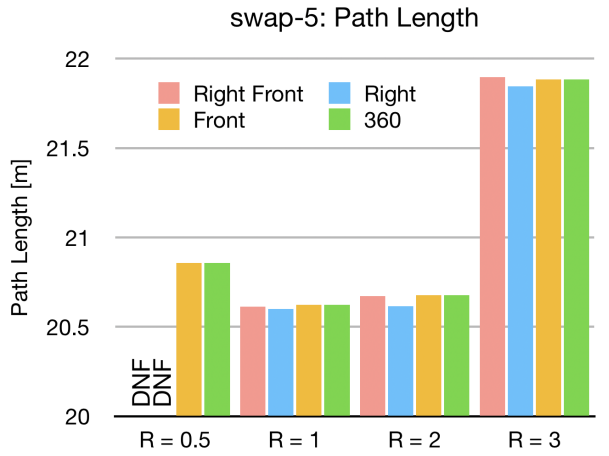
With the same R value, all four of the proposed methods perform similarly with the exception of the DNFs mentioned above and some variations in roughness. For the *swap-2* and *swap-3* scenarios, the *360* and *Right* methods are much smoother. This is likely because there was not a significant amount of congestion. For the *Front* and *Right-Front* methods, as soon as all nearby robots left the sector of interest, the ego robot would snap back to a direct goal route. This can be seen in Figure 5. The middle image in this figure shows the paths resulting from the *Front* method with $R = 3m$ and one can see that as soon as the robots pass each, their trajectories snap back to a direct line to the goal. This behaviour increases the roughness score. The rightmost image in this figure shows the paths resulting form the *Right* method. With this method the robots continue to avoid each other for as long as the nearby robot is on the ego robot's right hand side and within $R = 3m$. This leads to a smooth path without the sharp transition. In the more congested scenarios, the opposite seems to occur. For *swap-8*, both *Right* and *360* methods have much higher roughness values at $R = 3$. This could be because there are lots of robots passing around the ego robot for which it does not necessarily need to change its trajectory for but it does anyway. For example, if a robot comes close to the ego robot behind it and on the right, the robot will alter its trajectory under the *360* and *Right* methods but not with *Front* or *Right-Front*. Figure 6 shows the trajectories from the on the *swap-8-3* scenario for the *360* method and one can see areas where the paths "wiggle".

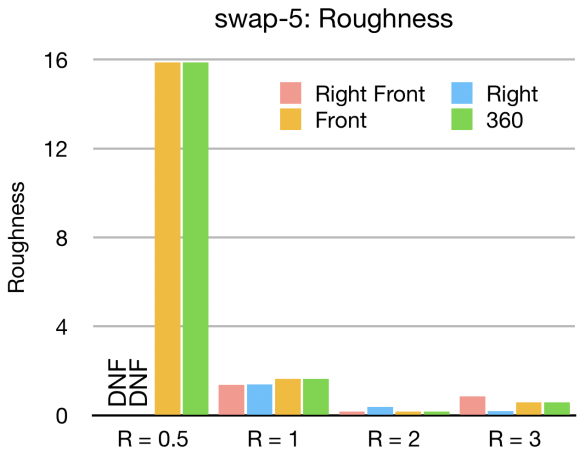
For the scenarios with 5 or fewer robots, there is not a significant decrease in roughness that comes with increasing R from 2m to 3m, but there is a significant increase in path length. For the scenarios with 8 or 11 robots, the effect of increasing R on roughness is less clear, but in all scenarios, $R = 3m$ increased the path lengths. Overall, it seems that the ideal value for R is 2m. The *Right-Front* and *Front* methods seem to promote smoother paths in more congested scenarios while the *Right* and *360* methods promote smoother paths in less congested scenarios.

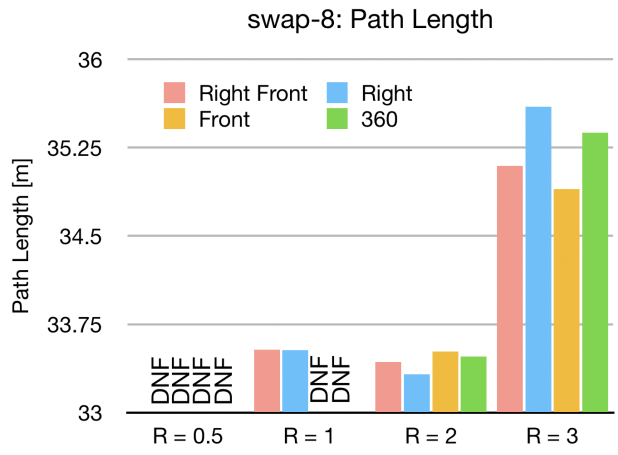


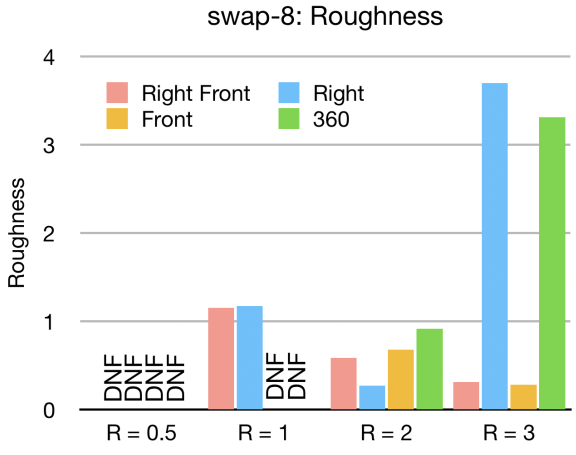

swap-3: Path Length
Path Length [m]
13
12.667
12.333
12
Right Front
Right
Front
360
DNF
DNF
DNF
DNF
R = 0.5
R = 1
R = 2
R = 3

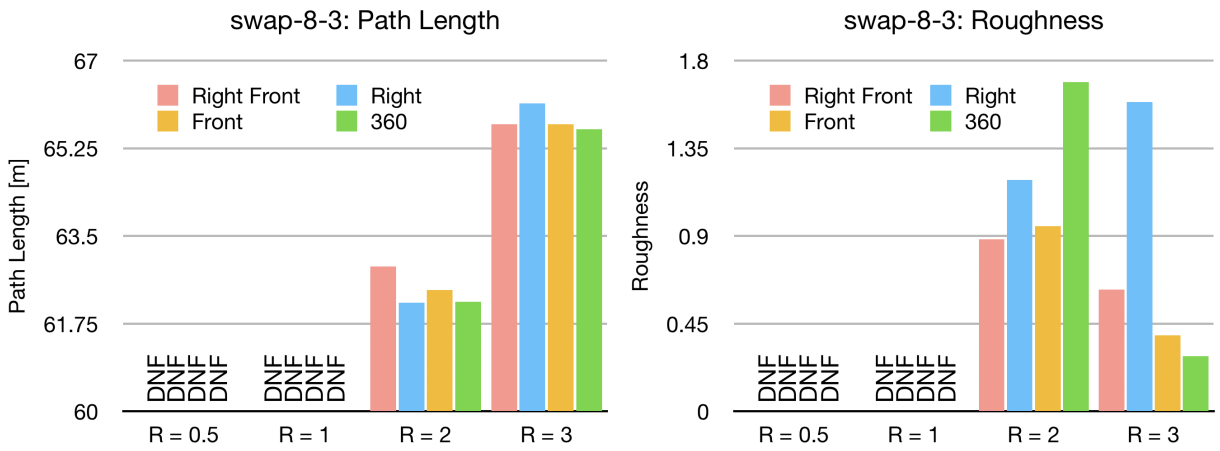

swap-3: Roughness
Roughness
2
1.5
1
0.5
0
Right Front
Right
Front
360
DNF
DNF
DNF
DNF
R = 0.5
R = 1
R = 2
R = 3


swap-5: Path Length
Path Length [m]
22
21.5
21
20.5
20
Right Front
Right
Front
360
DNF
DNF
R = 0.5
R = 1
R = 2
R = 3


swap-5: Roughness
Roughness
16
12
8
4
0
Right Front
Right
Front
360
DNF
DNF
R = 0.5
R = 1
R = 2
R = 3


swap-8: Path Length
Path Length [m]
36
35.25
34.5
33.75
33
Right Front
Right
Front
360
DNF
DNF
DNF
DNF
DNF
DNF
R = 0.5
R = 1
R = 2
R = 3


swap-8: Roughness
Roughness
4
3
2
1
0
Right Front
Right
Front
360
DNF
DNF
DNF
DNF
DNF
DNF
R = 0.5
R = 1
R = 2
R = 3



VII. Conclusions and Future Work

The preferred velocity methods presented here are simple to compute yet still able to successfully complete all the scenarios under certain parameter selections. This is an improvement over the *Direct* method which was not able to complete any but the most simplistic scenario. The downside to this approach is that certain parameters do need to be selected and can affect performance. It may be difficult to predict how the parameters would affect performance on an unseen scenario.

Future work could look at solving this problem using reinforcement learning similar to ALAN [11]. A network could be trained to output a $\mathbf{v}^{\text{nudge}}$ based on the position and velocities of robots around the ego robot. This $\mathbf{v}^{\text{nudge}}$ could then be added directly to the $\mathbf{v}^{\text{goal}}$ without the need for an α term. The model could be trained to minimize path length and roughness and would likely learn a more sophisticated combination of the preferred velocity methods proposed here. Instead of specifying an R value, the model would implicitly learn how to react to other robots at nearby distances. Since n-body reciprocal collision avoidance procedure would still be used, this makes training easier as the model does not need to produce $\mathbf{v}^{\text{nudge}}$ values that guarantee collision free trajectories.

References

[1] Van Den Berg, Jur, et al. "Reciprocal n-body collision avoidance." Robotics research. Springer, Berlin, Heidelberg, 2011. 3-19.

[2] Fiorini, Paolo, and Zvi Shiller. "Motion planning in dynamic environments using velocity obstacles." The International Journal of Robotics Research 17.7 (1998): 760-772.

[3] Van den Berg, Jur, Ming Lin, and Dinesh Manocha. "Reciprocal velocity obstacles for real-time multi-agent navigation." 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008.

[4] Fox, Dieter, Wolfram Burgard, and Sebastian Thrun. "The dynamic window approach to collision avoidance." IEEE Robotics and Automation Magazine 4.1 (1997): 23-33.

[5] de Lamadrid, James Gil. "Avoidance of obstacles with unknown trajectories: Locally optimal paths and periodic sensor readings." The International journal of robotics research 13.6 (1994): 496-507.

[6] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," in IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pp. 278-288, June 1991.

[7] Alonso-Mora, Javier, et al. "Optimal reciprocal collision avoidance for multiple non-holonomic robots." Distributed autonomous robotic systems. Springer, Berlin, Heidelberg, 2013. 203-216.

[8] J. Alonso-Mora, A. Breitenmoser, P. Beardsley and R. Siegwart, "Reciprocal collision avoidance for multiple car-like robots," 2012 IEEE International Conference on Robotics and Automation, Saint Paul, MN, 2012, pp. 360-366.

[9] Inácio, Fabrício R., Douglas G. Macharet, and Luiz Chaimowicz. "United we move: Decentralized segregated robotic swarm navigation." Distributed Autonomous Robotic Systems. Springer, Cham, 2018. 313-326.

[10] Li, Hao, et al. "Reciprocal Collision Avoidance for General Nonlinear Agents using Reinforcement Learning." arXiv preprint arXiv:1910.10887 (2019).

[11] Godoy, Julio, et al. "ALAN: adaptive learning for multi-agent navigation." Autonomous Robots 42.8 (2018): 1543-1562.

[12] Khan, Shehryar Ali, et al. "Collaborative Optimal Reciprocal Collision Avoidance for Mobile Robots." International Journal of Control and Automation 8.8 (2015): 203-212.

[13] Alonso-Mora, Javier, et al. "Multi-robot system for artistic pattern formation." 2011 IEEE international conference on robotics and automation. IEEE, 2011.

[14] Trautman, Peter, and Andreas Krause. "Unfreezing the robot: Navigation in dense, interacting crowds." 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010.

[15] Alonso-Mora, Javier, et al. "Image and animation display with multiple mobile robots." The International Journal of Robotics Research 31.6 (2012): 753-773.

[16] Alejo, Dominique, et al. "Optimal reciprocal collision avoidance with mobile and static obstacles for multi-UAV systems." 2014 International Conference on Unmanned Aircraft Systems (ICUAS). IEEE, 2014.

Appendix: Experimental Results

Table 1: swap-2

Preferred Velocity		Path Length	Roughness	Sim Time
Direct to Goal		8.1287	6.007	4.9
R = 0.5	Right Front	8.1181	4.6639	4.75
	Right	8.1184	4.6694	4.75
	Front	8.1181	4.6543	4.75
	360	8.1184	4.6694	4.75
R = 1	Right Front	**8.0637**	1.1652	4.2
	Right	**8.0637**	1.0445	4.2
	Front	**8.0637**	1.1632	4.2
	360	**8.0637**	1.0445	4.2
R = 2	Right Front	8.0900	1.4697	**4.1**
	Right	8.1713	0.0920	4.15
	Front	8.0900	1.4682	**4.1**
	360	8.1713	0.0920	4.15
R = 3	Right Front	8.3221	1.7450	4.25
	Right	8.4698	**0.0838**	4.3
	Front	8.3221	1.7438	4.25
	360	8.4698	**0.0838**	4.3

Table 2: swap-3

Preferred Velocity		Path Length	Roughness	Sim Time
Direct to Goal		DNF	DNF	DNF
R = 0.5	Right Front	DNF	DNF	DNF
	Right	DNF	DNF	DNF
	Front	DNF	DNF	DNF
	360	DNF	DNF	DNF
R = 1	Right Front	**12.2036**	1.9718	4.3
	Right	**12.2036**	1.9276	4.3
	Front	12.2143	1.8593	4.3
	360	12.2166	1.8142	4.3
R = 2	Right Front	12.2362	0.8214	**4.15**
	Right	12.2956	0.0513	**4.15**
	Front	12.2381	0.8206	**4.15**
	360	12.2970	0.0513	**4.15**
R = 3	Right Front	12.7960	0.6298	4.35
	Right	12.8239	0.0423	4.35
	Front	12.7988	0.6306	4.35
	360	12.8261	**0.0422**	4.35

Table 3: swap-5

Preferred Velocity		Path Length	Roughness	Sim Time
Direct to Goal		DNF	DNF	DNF
R = 0.5	Right Front	DNF	DNF	DNF
	Right	DNF	DNF	DNF
	Front	20.8563	15.8732	19.9
	360	20.8563	15.8732	19.9
R = 1	Right Front	20.6125	1.3670	4.65
	Right	**20.6007**	1.3807	4.65
	Front	20.6221	1.6316	4.7
	360	20.6221	1.6316	4.7
R = 2	Right Front	20.6717	0.1657	**4.2**
	Right	20.6163	0.3793	**4.2**
	Front	20.6769	**0.1654**	**4.2**
	360	20.6769	**0.1654**	**4.2**
R = 3	Right Front	21.8953	0.8517	4.5
	Right	21.8458	0.1816	4.5
	Front	21.8821	0.5924	4.5
	360	21.8821	0.5924	4.5

Table 4: swap-8

Preferred Velocity		Path Length	Roughness	Sim Time
Direct to Goal		DNF	DNF	DNF
R = 0.5	Right Front	DNF	DNF	DNF
	Right	DNF	DNF	DNF
	Front	DNF	DNF	DNF
	360	DNF	DNF	DNF
R = 1	Right Front	33.5357	1.1522	5.2
	Right	33.5304	1.1731	5.15
	Front	DNF	DNF	DNF
	360	DNF	DNF	DNF
R = 2	Right Front	33.4295	0.2939	**4.3**
	Right	**33.3257**	**0.2742**	**4.3**
	Front	33.5169	0.6797	4.35
	360	33.4754	0.9145	4.35
R = 3	Right Front	35.0946	0.3112	4.65
	Right	35.5971	3.698	4.8
	Front	34.8972	0.2798	4.75
	360	35.3764	3.3139	4.8

Appendix: Experimental Results

Table 5: swap-8-3

Preferred Velocity		Path Length	Roughness	Sim Time
Direct to Goal		DNF	DNF	DNF
R = 0.5	**Right Front**	DNF	DNF	DNF
	Right	DNF	DNF	DNF
	Front	DNF	DNF	DNF
	360	DNF	DNF	DNF
R = 1	**Right Front**	DNF	DNF	DNF
	Right	DNF	DNF	DNF
	Front	DNF	DNF	DNF
	360	DNF	DNF	DNF
R = 2	**Right Front**	62.8945	0.8843	6.55
	Right	**62.1682**	1.1890	6.5
	Front	62.4208	0.9506	6.5
	360	62.1861	1.6895	**6.45**
R = 3	**Right Front**	65.7348	0.6254	6.85
	Right	66.1517	1.5879	6.75
	Front	65.7296	0.3900	6.7
	360	65.6321	**0.2834**	6.75