AA277: Collision Avoidance Final Project

Valentin Antoine
Dept. of Mechanical Engineering
Stanford University
valent1@stanford.edu

Bradley Collicott
Dept. of Aeronautics & Astronautics
Stanford University
collicott@stanford.edu

Brian Dobkowski
Dept. of Mechanical Engineering
Stanford University
bdobkows@stanford.edu

Torstein Eliassen
Dept. of Electrical Engineering
Stanford University
torsteoe@stanford.edu

***Abstract*—Multi-robot systems have been widely applied to perform a given task collaboratively and cooperatively. A major challenge for these systems is to avoid collision, either between robots, other agents, or the surrounding environment. Collision avoidance strategies traditionally leverage model-based algorithms with many tunable parameters. Recent advances in reinforcement learning have allowed for learning-based methods to supplement the state-of-the-art model-based collision avoidance algorithms. This work presents a decentralized multiagent collision avoidance algorithm based on deep reinforcement learning, which learns from a model-based algorithm named Optimal Reciprocal Collision Avoidance (ORCA). Specifically, the proposed approach develops a value network that encodes the estimated time to the goal given an agent's joint configuration (positions and velocities) with its neighbor. This value network is then used to generate control policies in a two-agent system via a one-step lookahead. In addition to studying the output of the value network, the reward function parameters have been varied to evaluate how well the reinforcement learning algorithm captures desired behavior.** [1]

I. Introduction

Collision avoidance is a preeminent problem in multi-robot control. Many path planning methods in literature make broad assumptions about information sharing and connection to a centralized planner to ensure collision-free paths; however, such a system becomes unwieldy and unrealistic when dealing with many agents in a real-time environment. Thus, the subject of interest for this project is decentralized collision avoidance, in which multiple agents plan safe trajectories without communication with neighboring agents. The literature can be divided between model-based and learning-based approaches to collision avoidance, both of which will be surveyed here.

II. Literature Review

A. Model-Based Collision Avoidance

The Dynamic Window Approach to Collision Avoidance [1] presents an approach for collision avoidance for single-robot systems. This is foundational work that takes into account the dynamics of the robot for creating collision-free paths. The method is in the group of local methods allowing for fast reaction to world changes. By solving an optimization problem the approach successfully weighs three objectives: "heading": i.e. progress towards goal, "clearance": distance to the closest obstacle and "velocity": prioritizing high velocities, constrained to collision-free velocities.

Reciprocal n-Body Collision Avoidance [2] discusses multiple-robot collision avoidance. The paper provides an approach to collision-free movement for each robot for a fixed duration, for robots using the same local protocol. The approach uses linear programming in order to find collision-free velocities. The term reciprocal collision avoidance refers to multiple robots attempting to avoid collisions simultaneously without communicating, but using the same strategy. The paper introduces ORCA: Optimal Reciprocal Collision Avoidance. The method deals with both intelligent dynamical objects and static objects. Kinematics and dynamic constraints are not taken into account in this paper.

This paper leverages the idea of Reciprocal Velocity Obstacles (RVO) to define constrained regions in the motion of each robot. A convex cone is developed to describe the relative velocity of robot A with respect to robot B over a time interval τ. If the relative velocity is outside the cone for the window τ, it can be guaranteed to be collision free for that time window. The ORCA formulation finds the optimal velocity of robot A that avoids being in this constrained region, and is the closest to its "optimization velocity" or desired velocity. To implement this approach, no inter-robot communication is technically required, but it is assumed the robots have perfect sensing capability (of their own states and other robot's states) and know the other robots' optimization velocities. Future directions involve incorporating robot motion constraints, sensing uncertainty, and more dimensions of motion (3D). Alonso et. al. [3] apply motion/dynamics constraints to extend the ORCA approach to nonholonomic robots.

Confidence Aware Motion Planning [4] attempts to improve predictions of other intelligent agents. By introducing a confidence parameter, a Bayesian belief, over the predictions, the robot better reacts to unexpected behaviour.

Schwager et. al. in 2017 [5] attempted to resolve some of the computational challenges associated with the RVO-

[1] Code available at https://github.com/bcollico/aa277_project

based techniques (namely ORCA) by using buffered Voronoi regions to ensure collision avoidance between multiple robots. Borrowed from coverage control, these Voronoi regions are buffered to ensure that the whole of the robot geometry is encompassed within the cell at all times. Using this approach requires only position estimates of the other robots, thereby avoiding the need to obtain velocity estimates which are usually more corrupted by noise due to the practicality of the sensors used. This method is suitable for application in online settings because a solving method that leverages simple geometric details was developed to reduce the complexity incurred by a quadratic program at each time step. Another advantage of this method is that it does not require all robots to be running with the same algorithm.

B. Learning-Based Collision Avoidance

To approximate the performance of centralized collision avoidance algorithms in a decentralized manner, several works have leveraged deep reinforcement learning to approximate the optimal policy for agents to cooperatively avoid collisions. Chen et. al. [6] formulate the collision avoidance problem as a sequential decision-making problem with partially observable agent states. Rather than using a model-based algorithm to plan control inputs, the authors use offline deep reinforcement learning to encode a learned value function into a neural network. This value function is embedded in each robot, and the optimal action at each observation time is selected by repeatedly maximizing the one-step lookahead value. This ensures that, on average, each robot behaves in a way that maximizes the value of the joint state. The authors pre-train the neural network using supervised learning with demonstration trajectory solutions from a state-of-the-art model-based collision avoidance algorithm, followed by reinforcement learning episodes with randomized environments. The resulting learned value function from this method, termed CADRL, leads to a policy that significantly outperforms the ORCA algorithm in terms of path quality.

Long et. al. [7] also formulate the collision avoidance problem as a partially observable sequential decision-making problem; however, in contrast to the CADRL policy that maps ego and neighbor agent state information to control decisions, the authors of [7] attempt to learn a policy that maps raw sensor information to a control decision. The purpose of crafting the policy in this manner is to reduce the necessary complexity of the perception stack as compared with a policy that accepts derived quantities as inputs and to explicitly account for sensing uncertainty in the learned collision avoidance policy. The authors eschew a supervised learning step in their algorithm, and instead opt for a policy-gradient-based reinforcement learning algorithm that is trained on a set of curated environments with varying number of agents. Using 2D LiDAR-like range measurements as the input to the learned policy, this sensor-level reinforcement learning process yields a policy that outperforms both the ORCA algorithm and a supervised-learning-based policy in terms of success rate and time-to-goal. The authors demonstrate several scenarios unseen in training to which the agents adapted.

While the aforementioned reinforcement learning strategies demonstrated great success, they did not explicitly encode agent cooperation into their learned value networks. Sartoretti et. al. [8] attempt to impart this behavior into decentralized collision avoidance by crafting a reward function that penalizes actions that hinder other agents in the environment. In addition, the authors use a joint reinforcement learning and imitation learning strategy, similar to [6], that randomly introduces episodes of 'expert' trajectory demonstrations into the learning process. The problem is formulated as a partially observable sequential decision-making problem, where the partial-observability is introduced by limiting the field-of-view that each robot can see in the environment. This work, although promising in terms of scalability and collaborative collision avoidance and trajectory planning, is limited in scope due to its discrete state and action space, whereas the previously surveyed deep reinforcement learning collision avoidance works operated on continuous state and action spaces.

Kahn et. al. [9] attempt to address the problem of online reinforcement learning for collision avoidance. By recognizing that agents must experience collisions to learn how to avoid them, they formulate an uncertainty-aware reinforcement process that uses a velocity-dependent collision cost function in tandem with uncertainty-aware collision estimates that results in agents navigating uncertain environments, and thereby experiencing collisions, at significantly lower speeds than traditional reinforcement learning methods.

Socially Aware Motion Planning with Deep Reinforcement Learning [10] aims at enhancing the behaviour of a robot in an environment filled with humans, taking into account the stochasticity in people's behaviour. A learning-based approach is adopted - as opposed to a model-based approach - in order to reach socially aware collision avoidance by learning human-like navigation conventions. This was achieved with deep reinforcement learning for inducing socially aware behaviors in a reinforcement learning framework.

Crowd-Aware Robot Navigation With Attention-Based Deep Reinforcement Learning [11] underlines the effectiveness of reinforcement frameworks [10] to learn socially cooperative policies, but shows that such approach needs to be enhanced for a crowd as these cooperative policies assume a one-way Human-Robot interaction problem. Such approach is enhanced by using pairwise interactions between the robot and each human and by capturing the interactions among humans via local maps.

Model-based approaches differ from learning-based approach as they usually use additional parameters to account for social interactions with humans, adding that to usual multi-agent collision avoidance algorithms. Such approach is proposed in [12], quoted by [10], where they are the first to use Social Force Model in order to represent the social interactions. The idea is that changes in trajectories can be explained in terms of social fields or forces, and these social fields depend upon the nature of the agents interacting.

III. Problem Approach

Given that the state-of-the-art model-based methods in collision avoidance have now been eclipsed by reinforcement-learning-based methods, this project focuses on reproducing a foundational work in collision avoidance using deep reinforcement learning (CADRL). A general challenge in using learning policies on real robots is the question of interpretability; therefore, the intent of the project is to reproduce the learned value network from [6], vary the hyperparameters of the algorithm and study the value network output and pathplanning performance of the system.

A. Problem Formulation

Decentralized, un-communicative collision avoidance is posed here as a partially-observable sequential decision-making problem. The pairwise joint state to be considered by the policy for ego-robot i and neighbor robot j is $\boldsymbol{s}^{ij} = \left[\boldsymbol{s}^i, \boldsymbol{s}^j_o\right] \in \mathbb{R}^{14}$ where each robot state is comprised of observable $\boldsymbol{s}_o$ and hidden $\boldsymbol{s}_h$ quantities. The observable quantities are position and velocity $\boldsymbol{p}, \boldsymbol{v} \in \mathbb{R}^2$ and robot body radius $r \in \mathbb{R}$, and the hidden quantities are goal position $\boldsymbol{p}_g \in \mathbb{R}^2$, preferred speed, and heading angle $v_{pref}, \phi \in \mathbb{R}$.

The considered action space is continuous with a policy $\pi(\boldsymbol{s}^{ij}) = \boldsymbol{u}^i : \mathbb{R}^{15} \mapsto \mathbb{R}^2$. During propagation, upper and lower bounds on the change in heading angle restrict the action space. By constraining the action space, we enforce reasonable commanded control input. Single-integrator dynamics will be used as the baseline model for propagating the agents:

$$\boldsymbol{s}^i_{t+1} = \boldsymbol{s}^i_t + \boldsymbol{u}^i_t \Delta t = \boldsymbol{s}^i_t + \pi(\boldsymbol{s}^{ij})\Delta t \tag{1}$$

$$\boldsymbol{s}^j_{t+1} = \boldsymbol{s}^j_t + \boldsymbol{u}^j_t \Delta t = \boldsymbol{s}^j_t + \pi(\boldsymbol{s}^{ji})\Delta t \tag{2}$$

A natural optimization problem may be stated as minimizing the expected time-to-goal under dynamic constraints (1, 2) and collision avoidance constraint (4).

$$\arg\min_{\pi_{\boldsymbol{s}}} \mathbb{E}\left[t_g \mid \boldsymbol{s}, \pi_{\boldsymbol{s}}\right] \tag{3}$$

$$s.t. \quad ||\,\boldsymbol{p}_i(t) - \boldsymbol{p}_j(t)\,||_2 \geq r_i + r_j \tag{4}$$

Model based methods can provide guarantees for solving this optimization problem, but are often assumption heavy. Reinforcement Learning and other learning based methods may allow for more flexible and often more nonlinear and complex policies. Our approach will combine a model-based approach phase and a reinforcement learning phase. The reward function will be composed of a combination of incentives for reaching the goal and penalties for close approaches and collisions with other agents, such that $R(\boldsymbol{s}^{ij}, \boldsymbol{u}^i) = r_g + r_{prox} + r_{coll}$. The parameters used in the reward function will follow the approach outlined in [6], and an alternate reward formulation will be presented in following sections.

B. Training Process

The first process of training is a variant of Inverse Reinforcement Learning (IRL), where generated ORCA trajectories are used as "expert" demonstrations. The output of this step will be a trained neural network imitating a value function for collision avoidance policies. The optimal value function is defined below.

$$V^\star(\boldsymbol{s}^{ij}) = \sum_{t=0}^{T} \gamma^{t \cdot v_{pref}} R(\boldsymbol{s}^{ij}_t, \pi^\star(\boldsymbol{s}^{ij}_t)) \tag{5}$$

The neural network approximation of the optimal value function can then be used for retrieving the optimal policy of an agent.

$$\pi^\star(\boldsymbol{s}^{ij}_{t+1}) = \arg\max_{\boldsymbol{u}} R(\boldsymbol{s}^i_t, \boldsymbol{u}^i) + \gamma^{\Delta t \cdot v_{pref}} \int_{\boldsymbol{s}^{ij}_{t+1}} T(\boldsymbol{s}^{ij}_t, \boldsymbol{s}^{ij}_{t+1} \mid \boldsymbol{u}^i) V^\star(\boldsymbol{s}^{ij}_{t+1})\, d\boldsymbol{s}^{ij}_{t+1} \tag{6}$$

The second phase of training is reinforcement learning in an attempt to improve the value function learned from the ORCA demonstrations and incorporate reward functions. During this phase, trajectories are rolled out using ϵ-greedy exploration, while training the value function to learn and improve the policy.

1) Rotational invariance: Because the optimal policy should be invariant to any coordinate transformation (rotation and translation), there is some redundancy in the rotated joint state $\boldsymbol{s}^{ij'}$ that is provided as an input to the learned value network. Note that $(\cdot)'$ indicates a rotated quantity and that d_g and d_{ij} represent the distance from robot i to goal and to robot j, respectively.

$$\begin{aligned} \boldsymbol{s}^{ij} &= \texttt{rotate}(\boldsymbol{s}^{ij}) \\ &= (d_g,\, v_{pref},\, v'_{i,x},\, v'_{i,y},\, r_i,\, \phi'_i,\, v'_{j,x},\, v'_{j,y}, \\ &\quad p'_{j,x},\, p'_{j,y},\, r_j,\, r_i + r_j,\, \cos(\phi'_i),\, \sin(\phi'_i),\, d_{ij}) \end{aligned} \tag{7}$$

In order to remove rotational ambiguity, an agent-centric frame is defined, with the origin at the ego-agent's position, and the x-axis pointing toward the ego-agent's goal, as shown in Fig. 1. The neural network is only queried using rotated states to ensure it learns a proper value function relative to robot i's state.

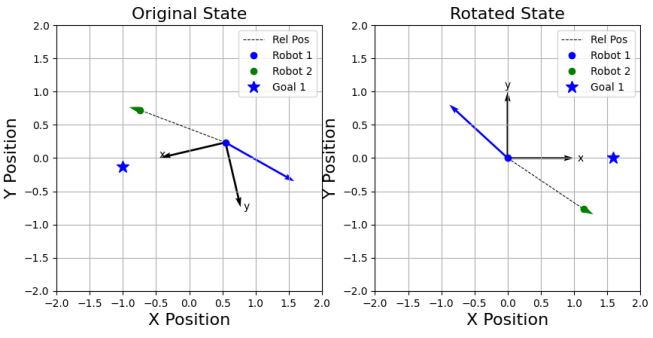


Fig. 1: Visualization of the Rotated State.

As angles are periodic, we constrain our θ to be between $-\pi$ and π before feeding it to the NN, something not done in [6]. As $V(\theta)$ should be equal to $V(\theta + 2\pi)$, we assume this can only improve the algorithm.

C. Proposed experiments

The resulting policy is evaluated in a discretized grid environment with no modeled obstacles except the boundary of the grid.

The original reward function proposed by Chen et al. awards the agent for reaching its goal and penalizes the agent for getting too close or colliding with the other agent [6]. It is described mathematically below:

$$R(\boldsymbol{s}^{ij}, \mathbf{a}) = \begin{cases} -0.25 & \text{if } d_{min} < 0 \\ -0.1 - d_{min}/2 & \text{else if } d_{min} < 0.2 \\ 1 & \text{else if } \mathbf{p} = p_g \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

where d_{min} is the minimum separation distance between the two agents over a duration Δt. In order to examine the efficacy of the reinforcement learning process, we offer a modified reward function using different values to further disincentivize collision. To do this, thee penalty and reward values, as well as the allowable value of d_{min}, were changed. The alternate reward function is given below:

$$R(\boldsymbol{s}^{ij}, \mathbf{a}) = \begin{cases} -0.75 & \text{if } d_{min} < 0 \\ -0.5 - d_{min}/2 & \text{else if } d_{min} < 0.5 \\ 1 & \text{else if } \mathbf{p} = p_g \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

These reward functions will be later referred to as "reward function 0" and "reward function 1". As in the original algorithm, cooperation is encouraged by adding a penalty term based on a comparison of the two agents' extra time to reach the goal,

$$t_e = t_g - \frac{d_g}{v_{pref}} \tag{10}$$

If the extra time is outside of some lower and upper bounds, i.e. $t_e < e_l$ and $t_e > e_u$, one agent reached its significantly more quickly than the other. In this case a penalty of 0.1 will be subtracted from the training value.

D. Evaluation

In order to evaluate the learned value function both before and after reinforcement learning, the value function is visualized through a series of heatmaps. The value function is evaluated on its interpretability.

In order to evaluate our policy, throughout training we randomly select initial states and propagate robots forward using a greedy one-step lookahead with respect our reward and value function. We do this until we have 10 successful runs (successful implies reached goal in a reasonable time and no collision) and examine the total number of trajectories that are

- Collision-leading
- Slow to reach goal: > 25 timesteps

During training, statistics on these simulations are monitored to ensure convergence. After training, we conduct 100 rollout simulations and collect statistics for these successes, collisions and timeouts to evaluate the effictiveness of the learned value function.

A qualitative analysis of the trajectories, involving plotting trajectories and analyzing them, is done to ensure the network is learning what we intend. Some qualities looked for include:

- Trajectory smoothness
- Time to reach goal compared with ORCA trajectories
- Successful behavior that is not captured by RVO collision avoidance
- Cooperative robot behavior

IV. Implementation

A. Value Network Supervised Learning

The value network was first trained using supervised learning (SL) with expert ORCA trajectories. To encourage interaction in training, ORCA trajectories were generated with two robots starting on opposite halves of the plane, with their corresponding goals opposite to them. Figure 2 shows this distribution of initial states and goals. As seen in the figure, a buffer was placed in the middle of the plane to ensure that the robots did not start in an overlapping configuration. ORCA trajectories were generated for 1000 random sets of initial states and goals. Note that some trajectories still may not require a collision-avoidance technique (e.g. if the robots do not cross each others' path en route to goal). However, upon analysis it was confirmed that the majority of the trajectories are altered in order to avoid collision.

The value network is not trained on full trajectories. Rather, each data sample is a rotated joint state at one timestep. Before feeding to the neural network, the datapoints are shuffled. Each trajectory is a maximum of 100 time steps long, leading to an upper bound of $(1000 \text{ simulations}) \times (100 \text{ time steps}) \times (2 \text{ robots}) = (200,000 \text{ training samples})$. As some trajectories take fewer than 100 time steps to reach goal, the true number of training examples is $\approx 108,000$.

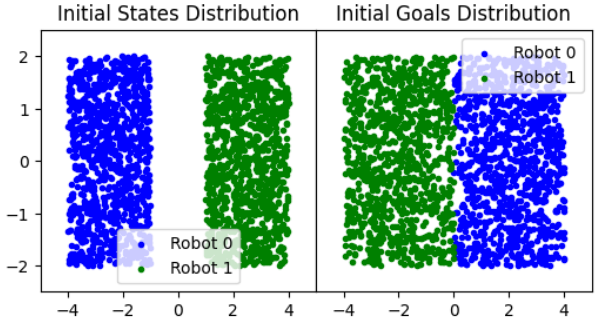


Fig. 2: SL Training Data Initial State and Goal Distribution - 1000 Simulations

The purpose of the training process is to encode a function that predicts $\gamma^{t_g \cdot v_{pref}}$ given $\boldsymbol{s}^{ij}$, so the truth value for each state given an ORCA trajectory is computed as $y(\boldsymbol{s}_t^{ij}, t) = \gamma^{(t_f - t) \cdot v_{pref}}$ for each step along a trajectory. An example ORCA trajectory from the training dataset is shown in Fig. 3.

This step can be viewed as an inverse reinforcement learning step as the value network attempts to learn the ORCA algorithm's evaluation of states, i.e. its "intent". The learned value of each state is then be used to warm start the reinforcement learning.

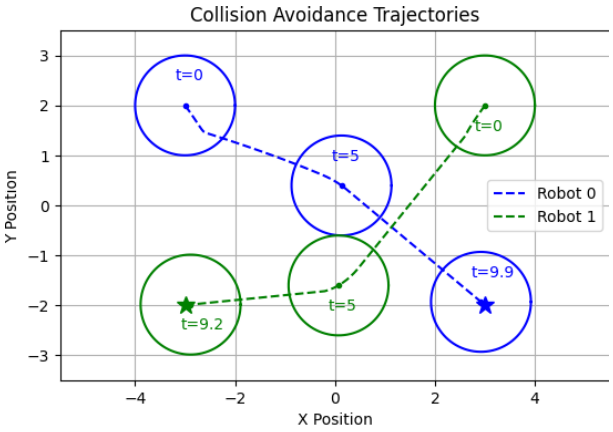


Fig. 3: "Expert" ORCA Trajectory Example

B. Value Network Reinforcement Learning

The value network was improved via reinforcement learning by selecting random episodes from the training dataset and using ϵ-greedy exploration to generate trajectories for the two robots moving past each other. This process involves selecting a random action for each robot at each timestep with probability ϵ and using a one-step lookahead (Eq. 6) to select the best available action otherwise. The ϵ probability parameter was decayed linearly from 0.5 to 0.1 over all RL training episodes. This was a liberty taken in our approach to inject more randomness into the training process than the Chen et al. implementation. To expedite the lookahead process and emulate a kinematically-constrained robot, the action space was limited to a subset of 25 evenly-spaced actions near the current heading direction and 25 random actions also in this neighborhood. The restricted action space is shown in Fig. 4, where the $\phi \pm \frac{\pi}{6}$ cone and 5 velocity magnitude increments can be seen for an agent at the origin with $v_{pref} = 1$. This action generation technique offered more granularity than the Chen et al. implementation, and seemed to improve performance in training.

Given the simulated trajectories using the value function, the successful (collision-avoiding and goal-reaching) trajectories were used to create state-value pairs (labels for training the network). The value of each state was calculated through one-step lookahead using the actual actions taken. The resulting state-value pairs were assimilated into the dataset, meaning the corresponding ORCA trajectories were replaced with the simulated trajectories. Gradually, the network is trained on fewer ORCA trajectories and more RL trajectories. This was believed to lead to steady training progress. We employ two reward functions ("reward function 0" and "reward function 1") in reinforcement learning to evaluate the difference in performance.

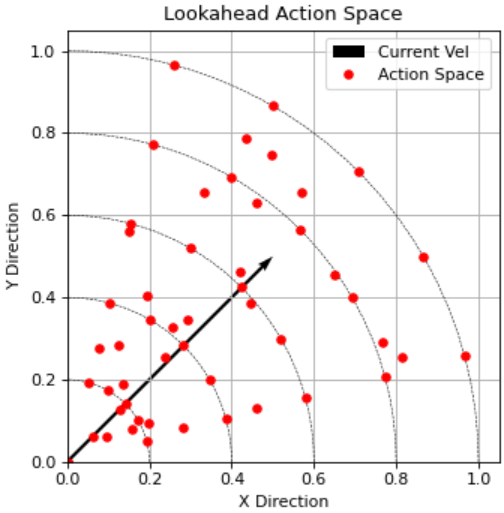


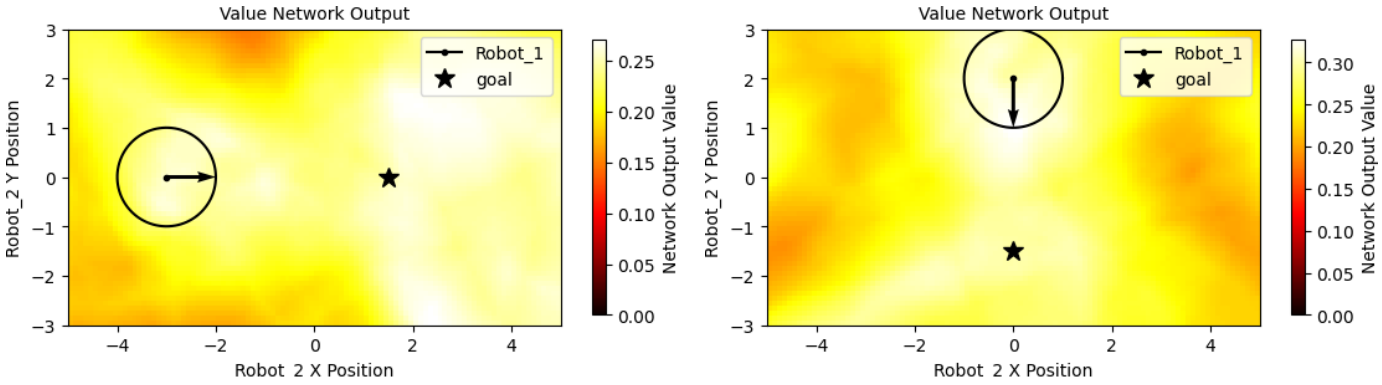
Fig. 4: Action Space for Lookahead

V. Results

The value function attained via SL using expert ORCA trajectories can be visualized in a simplified manner using a heatmap in the 2D plane. By holding all variables constant except for the position of one of the robots, the output of the value function can be mapped to Cartesian coordinates. Keep in mind that in actuality, the rotated state fed to the neural network as an input is 15-dimensional. Therefore, this heatmap is a restriction of the 15-dimensional input space to the 4-dimensional position space.

The value function heatmap can be seen in Fig. 5, where the position and velocity of Robot 1 are held constant while Robot 2 is moved around the plane. These two heatmaps show that the value function learned a rotation invariance, as we attain nearly the same value distribution in both Fig. 5a and Fig. 5b with a simple $90°$ rotation. We may also visualize the resulting value distribution by varying Robot 1's state while holding Robot 2 constant. In Fig. 6, Robot 2 is stationary in the plane, and Robot 1's position is varied while maintaining a velocity vector in the direction of goal. We can compare the SL policy to the RL policy by observing the difference in Fig. 6a and Fig. 6b. In both cases, the value network predicts that the time-to-goal will be low when Robot 1 is near its goal, and the time-to-goal will be high when Robot 1 is far from its goal or impeded by Robot 2. Improvement can however be seen in this aspect after RL value network improvement, as contrasts become higher, indicating confidence.

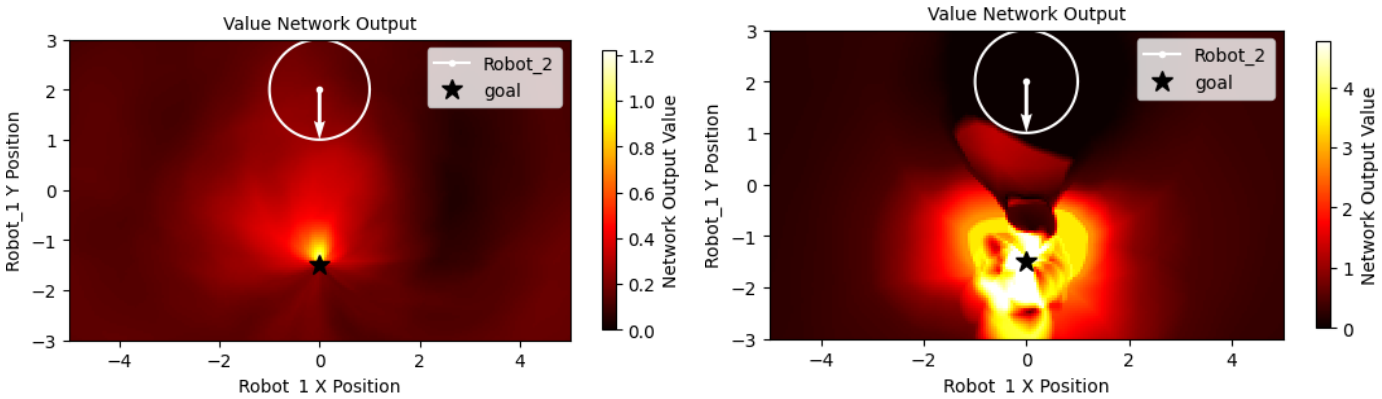
It also seems like pre-RL, the robot only learns to approach goal (as values are high around goal and gradually decreasing elsewhere), but post-RL it has learned collision-avoidance as well (as the region around robot 2 is darker). We hypothesize that a major detriment to the SL/IRL learning using expert demonstrations is that the ORCA trajectories never collided - they were mathematically guaranteed not to. The value function could not have been expected to learn to avoid collisions without ever witnessing one. This is where the reinforcement learning approach helped.



(a) Original Orientation.

(b) Rotated Orientation.

Fig. 5: Example of Rotation Invariance of Supervised Learning Value Network Output.



(a) SL Value Heatmap Varying Robot 1.

(b) RL Value Heatmap Varying Robot 1.

Fig. 6: Comparison of Supervised- and Reinforcement-Learning Value Networks.

We compare results from our two reward functions in Fig. 7. The figure shows that collisions decrease rapidly in both cases before reaching a more steady and slow collision reduction rate. It can be seen in these figures that there is a tradeoff between the number of colliding runs and the number of successful (or time-efficient) runs. The lower half of Figure 7 shows performance given by the reward function from Eq. 9, illustrating that with a more aggressive collision-avoiding strategy, the robots collide less often but also do not reach their goals within the allotted time as successfully. This figure proves the algorithm is finding more value in avoiding collisions to the extent of potentially being unable to reach the goal. These results suggest that it would be prudent to use the less conservative reward function 0 or expand the time horizon for reward function 1.

At the end of training we have these results for our two reward functions and our initial value function.

TABLE I: Policy Performance Summary.

	#Attempts	#Collision	#Timeout	#Success
Pre RL	100	84	8	8
Rew. 0	100	31	6	63
Rew. 1	100	5	73	22

We see a clear improvement from using the supervised learned value function directly for both reward functions.

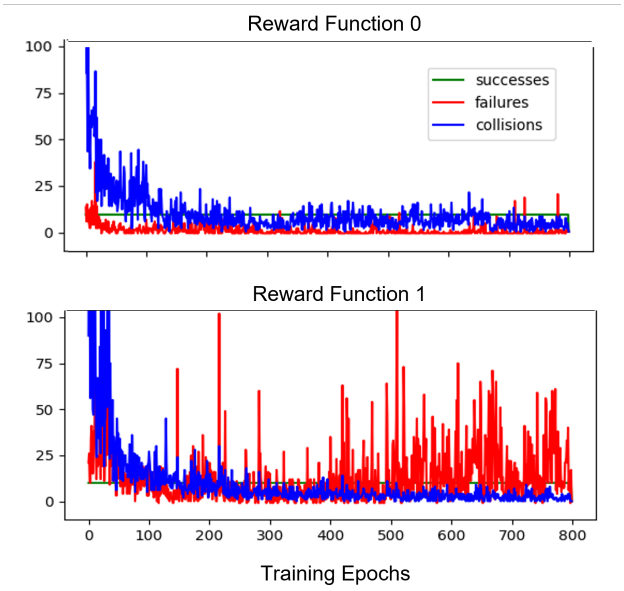
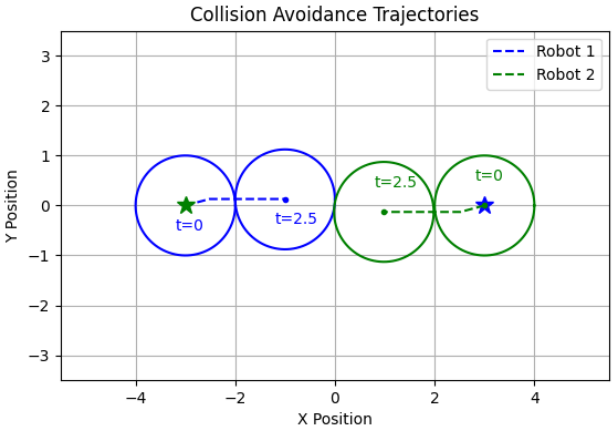
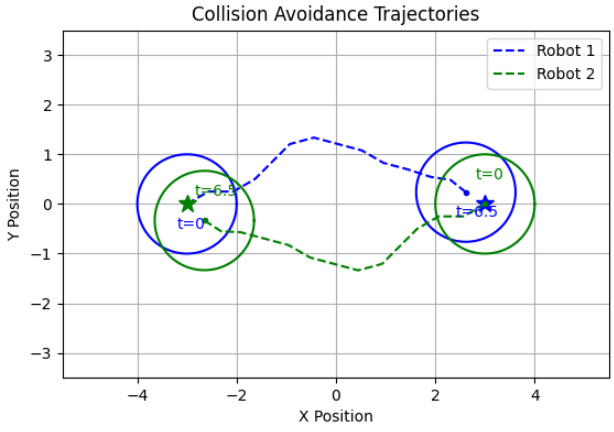


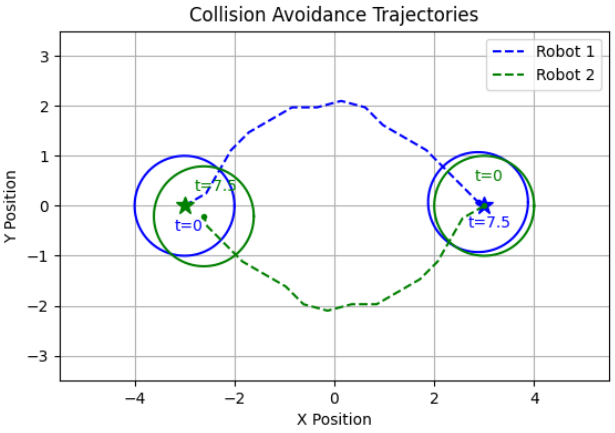
Fig. 7: Reward function 0 & 1: Collisions, Timeout vs. Time



(a) Initial ORCA value function



(b) Reward function 0



(c) Reward function 1

Fig. 8: Sample Trajectories with Each Value Function

Finally, we show a propagated trajectory from three different value functions: The value function outputted from the ORCA imitation learning (Fig. 8a), the value function from the RL with reward function 0 (Fig. 8b), and the value function from reward function 1 (Fig. 8c). Firstly, the RL trajectories both show collision avoiding trajectories in contrast with the pre-RL trajectory where the agents collide. Further, the agent trained on the updated reward function is significantly more collision avoiding than the other post-RL agent. This is clearly an effect of higher penalty in the reward.

VI. Conclusion

In this paper, we have adapted the method from Chen et. al. [6], modifying several parameters and functions, and improving on some implementation details. The results are promising showing that model-based collision-avoiding behavior can be influenced and controlled by reinforcement learning methods and reward tuning. The RL based approach does not require knowing the goal states or preferred velocities of other robots as do many model-based approaches. This partially observable non-communicative scenario is more robust to issues with sensing or communication between robots. However, the RL approach loses the guarantees of the model-based approaches.

In any case, further testing on different scenarios and multi-agent systems should be done to verify our results. Regarding future research, one could also modify the behaviors of the other robots and observe the effects on training and evaluation. In addition, some states may be redundant in the input of the neural network (e.g, r_j,r_i, $r_j + r_i$) and one should study the effect of removing the redundancy. Researching changes in reward/cooperation parameters and also tuning hyperparameters of the neural network/reinforcement learning algorithm are low-hanging fruit for other interesting projects.

Lastly, expanding the strategy to encompass a multi-robot scenario is a clear direction for improvement. The leap should not be difficult, as an effective technique would be using the same training infrastructure for training, and considering only the closest robot to robot i in the formulation of the joint state.

References

[1] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[2] J. v. d. Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011, pp. 3–19.

[3] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed autonomous robotic systems*. Springer, 2013, pp. 203–216.

[4] D. Fridovich-Keil, A. Bajcsy, J. F. Fisac, S. L. Herbert, S. Wang, A. D. Dragan, and C. J. Tomlin, "Confidence-aware motion prediction for real-time collision avoidance," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 250–265, 2020.

[5] D. Zhou, Z. Wang, S. Bandyopadhyay, and M. Schwager, "Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells," *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 1047–1054, 2017.

[6] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 285–292.

[7] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6252–6259.

[8] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. K. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[9] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," 2017.

[10] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1343–1350.

[11] C. Chen, Y. Liu, S. Kreiss, and A. Alahi, "Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 6015–6022.

[12] G. Ferrer, A. Garrell, and A. Sanfeliu, "Robot companion: A social-force based approach with human awareness-navigation in crowded environments," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1688–1694.