# Decentralized Collision Free Velocities from Depth Maps using Deep Reinforcement Learning

Alex Day
Clemson University
Clemson, SC
adday@clemson.edu

## ABSTRACT

Producing an optimal path for robots in a multi-agent scenario is challenging when the environment is only partially observable. Current solutions to this problem often work by assuming near-perfect knowledge of their neighbor's states. This work presents a solution to this problem based only on a point cloud reading of the environment in front of the scenario, current velocity, and goal position.

## 1 INTRODUCTION

Calculation of collision-free velocities in a decentralized manner is a well-studied problem. Previous approaches include geometric [7] and force [4] based approaches. These approaches rely on near-perfect knowledge of an agent's neighborhood; this is often a problem when applied in the real world. This problem can be solved using deep reinforcement learning to train agents end-to-end with a sensor, such as LIDAR, as input [5]. Unfortunately, LIDAR scanners are normally expensive. Especially when compared to RGB-D cameras, which capture both a full-color image and a depth map. Deep reinforcement learning has been applied to depth maps before, for both general [8] and goal-oriented [5] movement. However, this approach has not yet been applied to avoiding collisions with other agents in the scene.

The purpose of this work is to apply deep reinforcement learning to the collision avoidance problem where the agent can sense its' local environment through a depth map.

## 2 REINFORCMENT LEARNING METHODOLOGY

### 2.1 Environment

The environment used to train the agents is MiniWorld [2] which has been modified with better support for both multi-agent scenarios and depth-based interpretations of the environment. Agents within this scenario are modeled as cylinders with proportions, camera locations, and camera specifics based on the Turtlebot 2 [1] equipped with a Microsoft Kinect [2]. There was only one scenario used in training. Each agent was positioned near the midpoint of each side of a 10×10 meter square with a goal on the center of the opposite wall. This scenario can be seen in Figure 1.

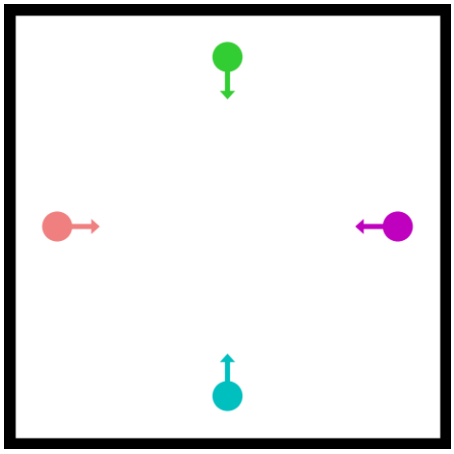

**Figure 1: Square scenario used in training**

The agent can observe this environment through three separate inputs. The first is a history of the last ten depth maps from a Kinect-like camera with a 60-degree field of view and a resolution of 800×640, which is downsampled to 80×64. An example of this depth map with an agent within the field of view can be seen in Figure 2. The agent is given the history rather than the most recent image to increase its knowledge about agents that might be nearby but out of the most recent field of view. Along with the depth map, the agent also has access to its' velocity from the previous frame and the offset from the current position to the goal, both in polar coordinates.

Agents can move within the environment through a linear, $l$, and rotational $\omega$ velocity. The linear velocity is bound between 0 and 1 $m/s$, so the agent cannot move into the unobservable space behind them, and the rotation is bound between $-\pi/2$ and $\pi/2$.

The reward function, Equation 1, consists of three separate functions to motivate goal-oriented, collision-free, and efficient movement, respectfully.

[1] https://clearpathrobotics.com/turtlebot-2-open-source-robot/
[2] https://docs.depthkit.tv/docs/kinect-for-windows-v2

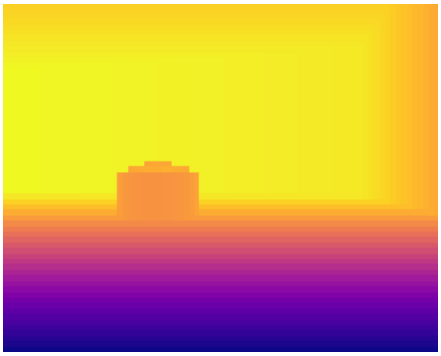

**Figure 2: Sample of a depth observation used in training**

$$R(a) = R_g(a) + R_r(a) + R_c(a) \tag{1}$$

The first function, Equation 2, consists of a large terminal reward and a small driving force derived from the distance towards the goal that the agent covers due to the new velocity.

$$R_g(a) = \begin{cases} 15 & \text{if } \|\vec{a_{pos}} - \vec{a_{goal}}\| < 0.1 \\ \|\vec{a'_{pos}} - \vec{a_{pos}}\| & \text{else} \end{cases} \tag{2}$$

The second function, Equation 3, deters the agent from colliding with other agents in the scenario with a large negative reward, as well as ending the current simulation.

$$R_c(a) = \begin{cases} -15 & \text{if collision} \\ 0 & \text{else} \end{cases} \tag{3}$$

The final function, Equation 4, deters the agent from excessive rotational movement. Lacking this, agents learn to large rotational velocities in efforts to maximize $R_g(a)$.

$$R_r(a) = -1 \cdot |a_\omega| \tag{4}$$

## 2.2 Proximal Policy Optimization

The agents are trained using the policy gradient-based approach Proximal Policy Optimization (PPO) [6]. Each agent draws from a central policy but and executes the given action in the scenario. The policy network is optimized using the trajectories gathered by all robots, which results in 4 times as many experiences per environment time step. The training algorithm is summarized in 1.

The neural network architecture comprises two heads, one convolutional for the depth images and two linear layers for the velocity and goal position; this can be seen in Figure 3. The architecture is copied from [3] with the idea that the network is learning a similar task. The network outputs a mean for both linear and rotational actions for the agent.

**Algorithm 1:** PPO Algorithm

```
Initialize policy networks π_θold and π_θ and set
  Hyperparameters as shown in Table 1
Trajectory Buffer ← []
for epoch = 1, 2, ..., E do
    for agent = 1, 2, ..., N do
        Run policy π_θold with standard deviation σ for
          T timesteps
        Add trajectory to Trajectory Buffer
        if ||Trajectory Buffer|| > M then
            Calculate Monte-Carlo Estimate of
              Rewards
            Optimize loss L wrt π_θ for K epochs
            π_θold = π_θ
            Clear Trajectory Buffer
        end
    end
end
```

| Hyperparameter | Value |
|---|---|
| Epochs, $E$ | 250 |
| Max Timesteps, $N$ | 1000 |
| Clip, $\epsilon$ | 0.2 |
| Action Standard Deviation, $\sigma$ | 0.1 |
| Learning Rate, $\alpha$ | 0.0003 |
| Reward Discount, $\gamma$ | 0.99 |
| Update Timesteps, $M$ | 100 |
| Agents, $N$ | 4 |

**Table 1: Training Hyperparameters**

These values are passed through sigmoid and tanh activation functions, and then the rotation mean is multiplied by $\pi/2$ to scale to the actual domain of the rotation action.

## 3 RESULTS

The agents were trained for around 10 hours on a system with an 8-core AMD processor and an Nvidia GTX 970. The agent's cumulative rewards for each episode can be seen in Figure 4 and the length of each episode can be seen in 5. The agents struggle to break 0 reward, although, they do approach 0 as training goes on. Theoretically if training were allowed to progress such that the agents had multiple trajectories where they reached the goal they would be able to break 0.

Agents did learn faster velocities, as seen in the distribution of the agent's speeds after training can be seen in Figure 6. The agents favor a higher speed that is around the max speed of 1 $m/s$.

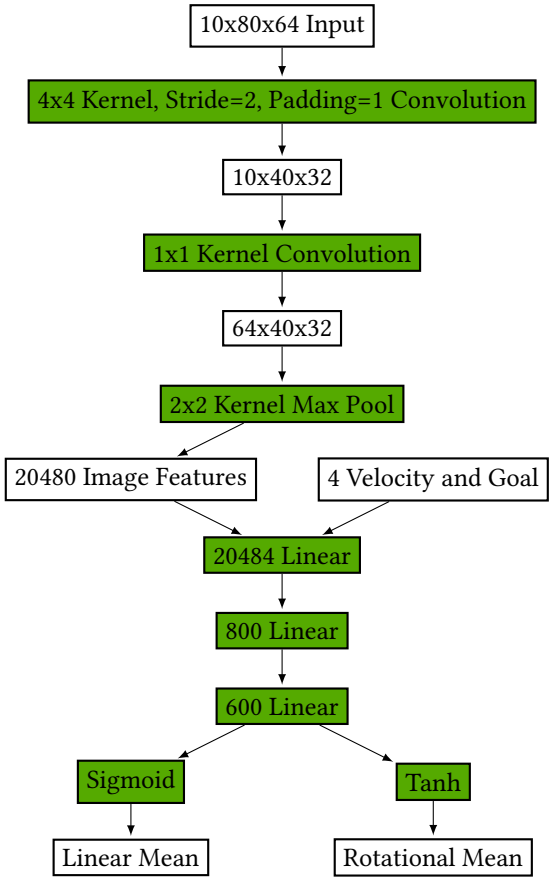


**Figure 3: Policy Network Architecture**

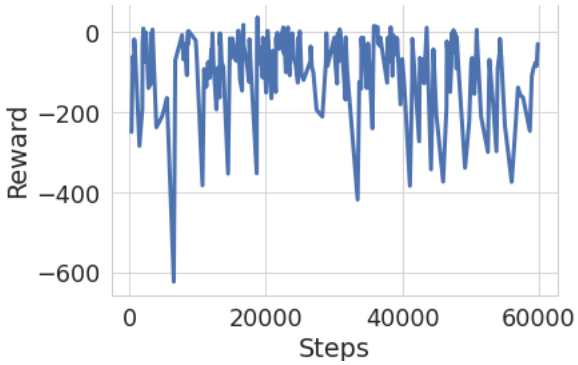


**Figure 4: Rewards accumulated during training**

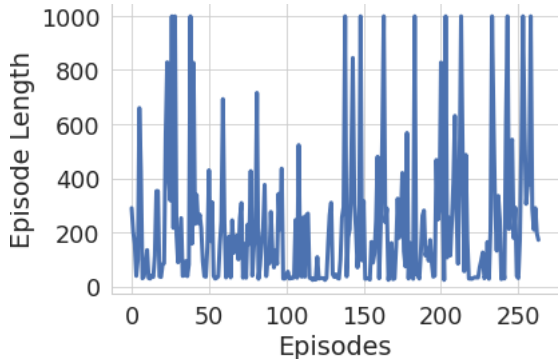


**Figure 5: Length of each epoisode during training**

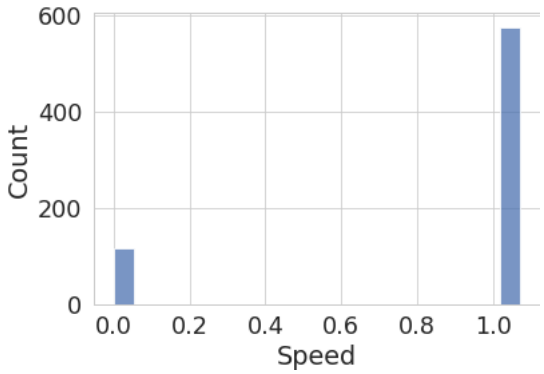


**Figure 6: Distribution of agent speeds at the end of training**

**Figure 7: Visualization of the last trajectory in training**

However, it does not seem that the agents learn efficient paths to their goals. The trajectory that the trained agents take is shown in Figure 7. It seems that the agents do learn to move, but they learn large angular trajetories which cause them to move in circles of varying radii.

The agents were able to reach the goal at around 20,000 steps. However they were not able to repeat this feat for the rest of the training. It is possible this could be repeated if the training were allowed to progress for long enough.

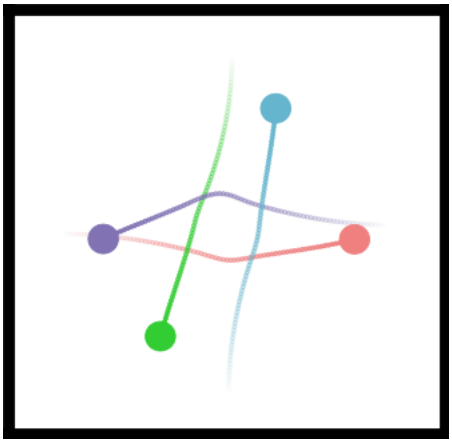

**Figure 8: Visualization of an early trajectory that moves the agents close to their goals**

## 4 FUTURE WORK

The agents struggled to reach the goal and avoid collisions to receive the significant incentive that would motivate them in future trajectories. To rectify this hindsight experience replay [1] could be utilized to move the goal and modify the agent's state space after the trajectory is completed.

On top of modifying the training, the environment could also be modified to remove dependencies that make the code unusable on headless machines. Without this limitation, the agents could train on better hardware for a more extended amount of time.

Currently, the reward function does not dissuade the agent from learning large rotational velocities, drastically slowing progress. These rotational velocities can be seen in Figure 7. This issue may be caused by naive reward function shaping, so more time should be spent testing the reward function on different proposed actions.

Finally, more stochasticity should be added to the scenario. Due to this project's time constraints, it was assumed it would be easier to train the agents in a single scenario with no noise in the starting or goal positions. For this work to generalize to the real world or even more complex simulations, the agents should be trained on either random scenarios, fixed complex scenes, or a combination of both.

## 5 CONCLUSION

In this work, a new end-to-end collision avoidance method was proposed utilizing deep reinforcement learning and depth maps. While the method, in its' current state, does not produce optimal paths to the agent's goal, this lays a promising foundation for future work.

## REFERENCES


[1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 5055–5065.

[2] Maxime Chevalier-Boisvert. 2018. gym-miniworld environment for OpenAI Gym. https://github.com/maximecb/gym-miniworld.

[3] Reinis Cimurs, Jin Han Lee, and Il Hong Suh. 2020. Goal-oriented obstacle avoidance with deep reinforcement learning in continuous action space. *Electronics* 9, 3 (2020), 411.

[4] Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. 2014. Universal power law governing pedestrian interactions. *Physical review letters* 113, 23 (2014), 238701.

[5] Pinxin Long, Tingxiang Fanl, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. 2018. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 6252–6259.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[7] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. 2011. Reciprocal n-body collision avoidance. In *Robotics research*. Springer, 3–19.

[8] Keyu Wu, Mahdi Abolfazli Esfahani, Shenghai Yuan, and Han Wang. 2018. Learn to steer through deep reinforcement learning. *Sensors* 18, 11 (2018), 3650.