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 multiagent scenario is challenging when the environment is only partially observable. Current solutions to this problem often work by assuming near perfect knowledge of their neighbors 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 agents neighborhood, this is often a problem when applied in the real world. This problem can be solved by using deep reinforcement learning to train agents end-to-end with a sensor, such as LIDAR, as input [5]. Unfortunately LIDAR scanners are expensive, as 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 and camera location modeled after the Turtlebot 2 platform [1]. There was only one scenario used in training where each agent was positioned near the midpoint of each side of a 10×10 meter square with a goal on center of the opposite wall. This scenario can be seen in Figure 1.

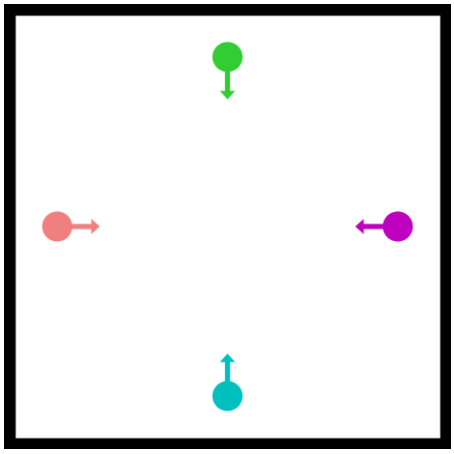

Figure 1: Visualization of the scenario used in training

The agent is able to observe this environment through three separate inputs. The first is a history of the last 10 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. 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 field of view of the most recent image. 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 ω velocity. The linear velocity is bound between 0 and 1 so the agent cannot move into unobservable space, and the rotation is bound between $-\pi/2$ and $\pi/2$.

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

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

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

[1] https://clearpathrobotics.com/turtlebot-2-open-source-robot/

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

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.

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

The final function, Equation 4, deters the agent from excessive rotational movement. Without this the agents learn to spin around their starting point to maximize $R_g(a)$ without actually progressing towards the goal.

$$R_r(a) = -1 \cdot |a_\omega| \quad (4)$$

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.

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
```

The neural network architecture is a comprised of two heads, one convolutional for the depth images and two linear layers for the velocity and goal position, this can be seen in Figure 2. The architecture is copied from [3] with the idea that the network is learning an extremely similar task.

Hyperparameter	Value
Epoches, E	200
Max Timesteps, N	1000
Clip, ϵ	0.2
Action Standard Deviation, σ	0.1
Learning Rate, α	0.0003
Reward Discount, γ	0.99
Update Timesteps, M	100
Agents, N	4

Table 1: Training Hyperparameters

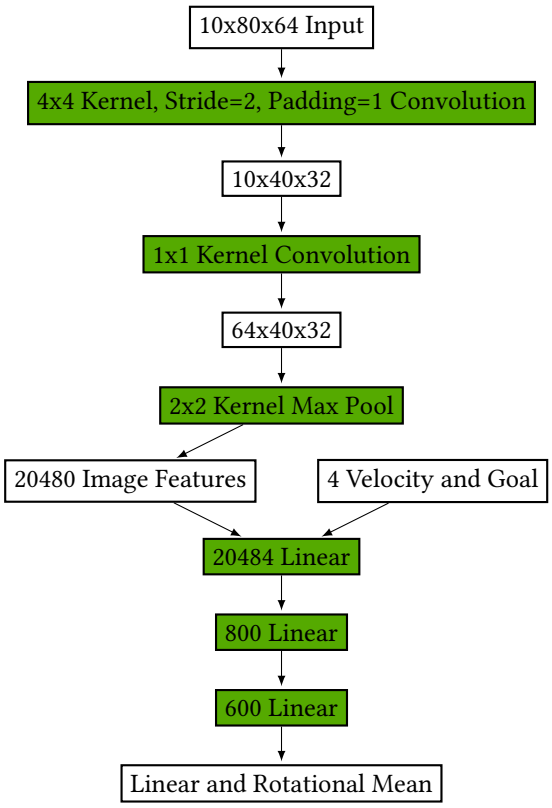

Figure 2: Policy Network Architecture

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 agents cummulative rewards for each episode can be seen in Figure 3 and the length of each episode can be seen in 4. Based off these graphs it is clear

The distribution of the agent

The trajectory that the trained agents took is shown in Figure 6. Based off of this figure it is clear that the red agent did not learn any collision free movement however the purple

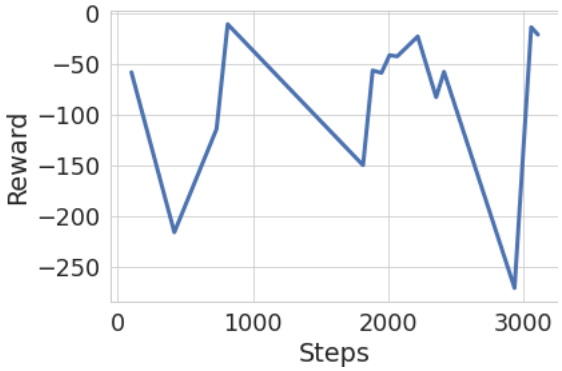

Figure 3: Rewards accumulated during training

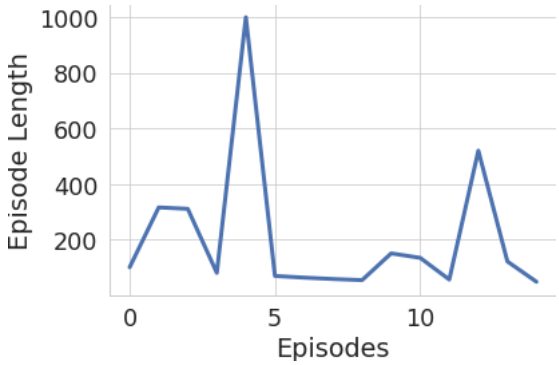

Figure 4: Length of each epoisode during training

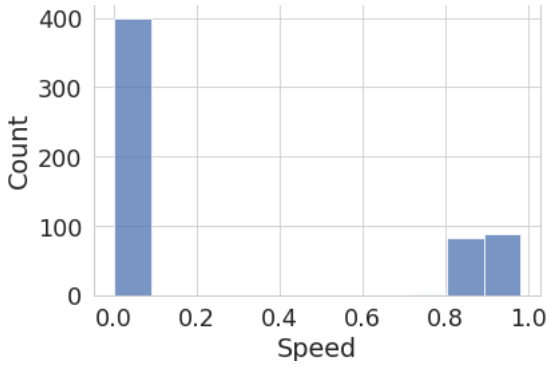

Figure 5: Distribution of agent speeds at the end of training

agent seems to avoid a collision with teal. This is interesting as all agents draw from the same policy and start with identical views of the world due to the deterministic nature of the scenario.

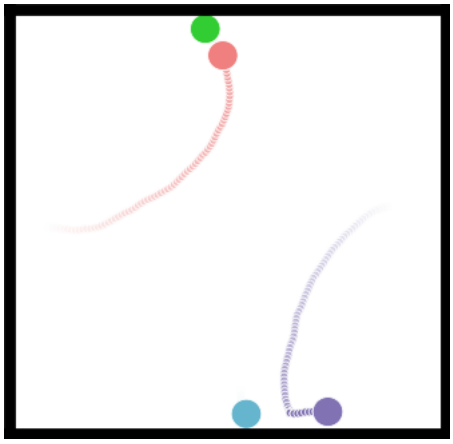

Figure 6: Visualization of the last trajectory in training

4 FUTURE WORK

It seems that the agents struggled to reach the goal to recieve the large insentive that would motivate them in future trajetories. To rectify this hightsight experience replay [1] could be utilized to move the goal and modify the state space of the agent after the trajetory is completed.

On top of modifiying 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 longer amount of time.

Finally, more stochasticity should be added to the scenario. Due to the time constraints of this project I 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 method of collision avoidance was proposed utilizing deep reinforcement learning and depth maps. While the method, in its' current state, is does not produce optimal paths to the agents goal the work forms a solid foundation for future research on this topic.

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.