Proceeding of the IEEE
International Conference on Robotics and Biomimetics
Dali, China, December 2019

Curiosity Driven Deep Reinforcement Learning for Motion Planning in Multi-agent Environment

Gojko Perovic
Department of Automation
Shanghai Jiao Tong University
Shanghai, China
gojkara@sjtu.edu.cn

Ning Li
Department of Automation
Shanghai Jiao Tong University
Shanghai, China
ning_li@sjtu.edu.cn

***Abstract*—Motion planning in dynamic multi-agent environment tends to be a challenging feat by itself. If the agent is also required to complete an elaborate task the complexity of the problem substantially increases. Heuristically designing policies for unstructured environments can be unfeasible and time consuming for certain scenarios. We propose a Deep Reinforcement Learning approach for a continuous multi-agent setting that is robust enough to handle high-level task accomplishment and low-level collision avoidance control. To alleviate disadvantages of a sparse reward environment we introduce intrinsic reward inspired by the curious behavior of animals and humans. As for the low-level, collision avoidance segment of control, we introduce a general reward function. Furthermore, we formulate an agent-centric state space and implement robust Reinforcement Learning algorithm that is capable of handling reward signal from both sources. Suitable 3D physical environment is constructed to examine the feasibility of our approach. Subsequently, case study is performed and effectiveness of our method is validated when compared with the state-of-the-art Reinforcement Learning algorithm.**

***Index Terms*—autonomous agents, collision avoidance, motion planning, multi-agent systems, robot learning**

I. Introduction

Designing controllers for agents that are able to perform in unstructured environments is the imperative for having successful robotic systems in the future. Robots, today, are limited to their work spaces and have limited contact with human workers. Furthermore, they are usually heuristically programmed to perform a certain type of tasks. This approach can be challenging if the task is complicated, and generalization to other tasks would be virtually nonexistent. For example, imagine designing a policy for a robot that needs to build a house. We could start by designing policies that build certain parts first, like walls or floor. Additionally, we would have to consider all the design necessary for low-level joint control and motion planing, then how to grasp and use certain tools, and so forth. It is apparent that heuristic engineering quickly becomes time consuming and defeats the purpose of automating the process. Aforementioned reasons are the main motivation behind emergence of artificially intelligent agents that are able to interact with the environment and learn how to act from experience.

Recently, emerging popular algorithms that enable interactive, model-free, learning tend to be Reinforcement Learning (RL) based. RL methods enable learning agents to map actions to situations in order to maximize a numerical reward signal [1]. By doing so, through trial-and-error process, agent should learn a policy that successfully completes a certain task. Policy $\pi(a, s)$ is a function that maps the action $a(t)$ to current state $s(t)$. It does so to maximize the return of expected reward, r, defined by the reward function $R(t)$. The value function $V_\pi(s)$ represents how beneficial it is to be in a certain state. This function is defined as reward return from a certain state s_0, while following the policy π.

$$V_\pi(s) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s\right] \tag{1}$$

where γ is the hyperparameter refereed to as discount factor. Furthermore, in many RL applications we consider the state-action value function, or the Q function. This function defines the expected reward by performing a particular action a in a certain state s.

$$Q_\pi(s, a) = \mathrm{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a\right] \tag{2}$$

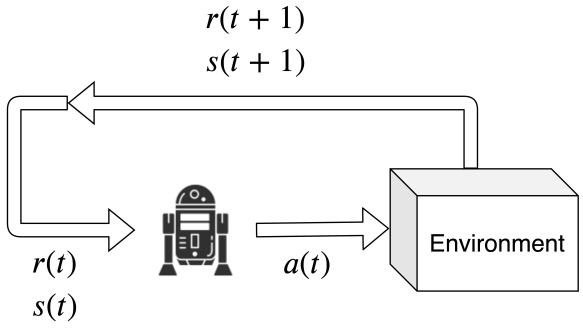


Fig. 1: Block diagram of basic Reinforcement Learning setting. Influenced by current state $s(t)$ and reward signal $r(t)$, agent interacts with the environment by performing the action $a(t)$. This in turn leads to new state $s(t+1)$ and reward $r(t+1)$.

While RL remains the most powerful approach for design of learning agents, it comes with certain drawbacks. Two of the most prominent handicaps of RL are:

- sparse reward environments
- exploration vs. exploitation compromise.

978-1-7281-6321-5/19//$31.00 ©2019 IEEE

Most practical problem settings don't have a form of constant feedback signal. Therefore, robotic environments could usually be characterized like those with sparse reward, as the agent gets the feedback signal only after it has completed the task. For reasons similar to those as heuristically designing policies, reward function engineering is not scalable. Besides, it can lead to unwanted agent behavior which completely neglects original goal [2].

Compromise between exploration and exploitation remains the most persistent and explored hindrance of the RL. In order for an agent to obtain reward it has to exploit the learned value function. In contrast, it has to explore, and perform seemingly wrong actions to discover new states and policies [1]. For the most part, basic exploration techniques tend to solve simple problems. However, as environments and tasks get more complex, the chance of getting stuck at local optima increases and sub-optimal policies are learned.

In this paper, we propose a Deep Reinforcement Learning based approach for motion planning and task completion in multi-agent environment. Our method is designed to be robust to all the requirements that are necessary to successfully complete a task in such environment. In order for the agent to learn complex behavior from limited extrinsic reward we supplement it with intrinsic reward. This reward signal is inspired by the curiosity that motivates animals and humans alike to explore their environments. As for collision avoidance control, we employ a general reward function. Therefore, our approach is able to consolidate high-level and low-level control. When compared with state-of-art RL algorithm, our method shows significantly better performance in complex continuous environment.

This paper is outlined as follows. Section II covers related work on RL algorithms for continuous control, intrinsically motivated agents, and motion planning and collision avoidance in RL. Section III outlines our methodology and explains the components of the algorithm. In Section IV we perform a case study and validate our method when compared with the state-of-art algorithm. Finally, in Section V we consolidate our findings and propose further extensions to this work.

II. Related Work

A. Reinforcement learning algorithms for continuous control

Traditionally, RL has been used to solve Markov Decision Processes (MDPs) which are discrete time stochastic processes. Furthermore, these applications were based on MDPs with finite number of actions. In practical applications, discretization of state space might not be useful or fine enough to produce good results [3]. Moreover, same issues arise from discretization of the action space. The most common approaches for RL in continuous space are policy gradient methods. These methods work by optimizing the policy π directly. Parametrized function approximators (FAs), are used to model the policy. These algorithms can be further extended with actor-critic (AC) methods. AC methods model both policy and state function. While policy represents the actor worker, the value function is the critic that sets the goals for the worker. Numerous state-of-the-art algorithms for continuous control are policy gradient or AC based, such as: DDPG [4], A3C [5], TRPO [6], PPO [7] etc.

Policy gradient methods suffer from poor convergence characteristics. Thus, training policies can be notoriously slow and sample inefficient. In contrast, increasing step size can overwhelm the training with noise. Proximal policy optimization (PPO) algorithm is designed to improve on these flaw by constraining the policy update. To reduce variance in training, state-action value function is sometimes substituted with advantage function A.

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{3}$$

Ratio p of probabilities between old and new policy is defined:

$$p_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \tag{4}$$

The objective function can be then defined as:

$$L^C(\theta) = \mathrm{E}_t[min(p_t(\theta)A^t_\pi, clip(p_t(\theta), 1-\epsilon, 1+\epsilon)A^t_\pi)] \tag{5}$$

If the probability ratio between the policies is outside the range $(1-\epsilon)$ and $(1+\epsilon)$, the advantage function will be clipped. This will in turn prevent large policy updates that might lead to divergence.

B. Intrinsically motivated agents

Humans and animals alike are not purely motivated by their biological needs, but are also driven by their curiosity [8], [9]. The motivation to explore novel situations can be just as prevalent as completing the task for the sake of the reward. Inspired by this cognitive trait there has been an effort to recreate intrinsic motivation artificially [10], [11].

Intrinsically motivated RL methods introduce a reward signal, r^i_t, that doesn't come externally from the environment but from the agent itself. Most popular method of artificially modeling curiosity is by representing it as error in forward dynamic prediction model [10]–[13].

$$r^i_t = \frac{1}{2}||\hat{\phi}(s_t+1) - \phi(s_t+1)||^2_2 \tag{6}$$

where $\hat{\phi}(s_t+1)$ and $\phi(s_t+1)$ are prediction of future state in feature representation and its actual value respectively. Advancements in algorithms based on intrinsically motivated agents demonstrated that some of the classical obstacles in RL can be alleviated. Pathak et al. developed a scalable curiosity module [12] and showed that competitive agents can be trained purely from intrinsic reward [13]. Another intrinsically based method, Random network distillation [14], achieved first better than human performance on Montezuma's Revenge, video game with sparse rewards. Aforementioned approaches are based on environments with visual observations and are concerned with limitations that stem from them. Intrinsic methods have also seen application as high-level control for curious humanoid agents [15]. This approach, however, focuses on developing a complex embedded system for an existing hardware platform.

C. Motion planning and collision avoidance in RL

Even the earliest RL algorithms found application in obstacle collision and motion planning settings [16]. Because RL approaches tend to be model-free, there is an appeal for their application in navigational problems. As such, extensive work has been done at the intersection of these fields, thus we chose to briefly highlight those that inspired our work the most.

Majority of the more recent approaches use Deep Reinforcement Learning techniques to address motion control in continuous environments [17], [18]. While most of these approaches were limited to static obstacles, the work of Yu et al. [18] showed that RL method can successfully be used to address the collision avoidance between moving agents. Aforementioned methods concerned themselves with low-level safety control and did not extend to settings with more complicated tasks.

III. Methodology

As the focus of our work is on control in continuous environment with continuous action space, we consider a policy gradient RL algorithm as the base of the method. We set up the agent to train from two reward signals, an intrinsic, r_t^i, and an extrinsic one, r_t^e. Multiple reward signals increase complexity of RL training, and this can lead to sub-optimal policies. Furthermore, we are training our agent to solve a complex task in a multi-agent environment. We consider other agent to be a non-communicating opponent, with a separate task. Learning agent will only have the limited information about opponents state, as it is often case in practice. This implies that our agent should be aware of possible collisions and work to avoid them. All of these requirements, make motion planning task challenging.

A. RL algorithm baseline

Algorithm with robust training qualities is required to handle the complexity of the setup. While we explored using different actor critic algorithms such as A3C [5], TRPO [6], and DDPG [4], the one that proved most efficient was PPO [7]. When compared with TRPO [6] e.g. PPO [7] has same robust qualities in safely updating the actor network, while being more scalable and easier to implement. Clipping the advantage function has shown to be crucial in training our agent. Therefore, if we were not to clip the advantage function, influenced by the two reward signals, our policy would likely diverge.

We apply some of the common modifications to the base PPO algorithm by also clipping the value function and normalizing the advantages [19].

B. Intrinsic module

For the intrinsic module, we decided to use the reward based on prediction error of a dynamic model. This approach has been shown to be scalable and easily parallelizable [13]. In addition to forward dynamic model we add an inverse model to encode our observations into feature space [12]. While we could train the forward model from the observations, featurizing observations in this way reduces the loss in the forward model. To make the training smoother we normalize the vector observations before inputting them into the neural networks.

$$s_{norm} = \frac{s - \mathrm{E}[s]}{\sqrt{\mathrm{Var}[s]}} \tag{7}$$

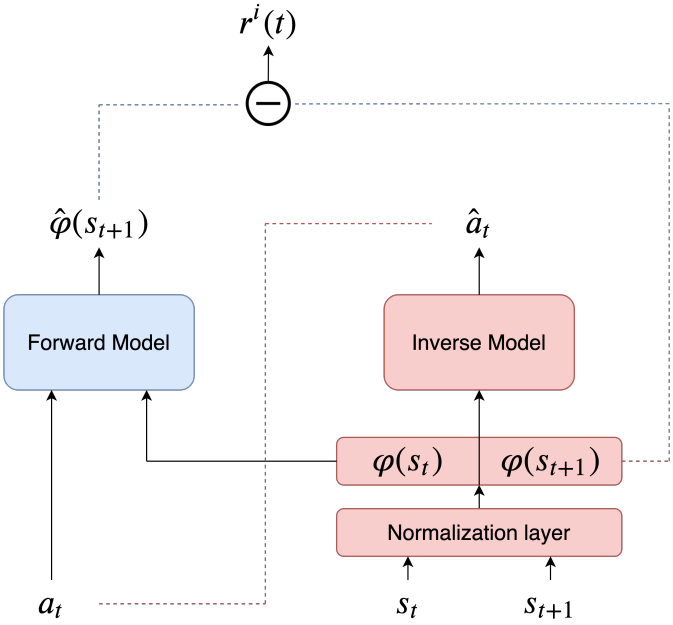


Fig. 2: Curiosity module. Forward and inverse model are represented by blue and red respectively. Variables used to calculate the loss of neural networks are connected by dotted lines.

As we can see from the Figure 2 agent will receive more intrinsic reward if the prediction of forward dynamic model is bad. This will lead the agent to explore the states previously unexplored. Seeing that they get explored, the predictions of forward model will get better, reducing the intrinsic reward. Hence, as intrinsic reward gets smaller, we ensure better exploration and our policy can converge to a desired one. Additionally, as curiosity signal can overwhelm the external reward, we will limit it by a constant c. We will consider the influence of this hyperparameter in Section IV.

C. State space formulation

State space of a multi-agent system can be defined as conjecture of all of the agents' states $s^{jn} = [s_1, s_2 ... s_n]$. As we regard our agents as non-communicating, we can divide agent's state vector into observable and hidden vectors $s_n = [s_n^o, s_n^h]$. Observable states are those visible to other agents such as location and speed, while hidden are not known to other agents, like the agent's intent and his goal position. To successfully train the agent we formulate an agent-centric state representation that includes the agent's state and the observations of the opposing agents:

$$s = [s_n, s_1^o, s_2^o, ... s_{n-1}^o] \tag{8}$$

D. General reward function

We construct the reward function that emphasizes the avoidance of other agents and obstacles, while also providing the feedback if the task is completed. Reward function should fulfill two requirements:

- robustness to collision avoidance segment
- independent to task completion steps

By fulfilling first requirement, we ensure that our method can generalize to other RL settings that might include collision avoidance. To avoid overfitting to one particular task, reward should be independent from the particular steps necessary to complete it. Reward function is constructed in equation (9).

$$R(t) = \begin{cases} 1 & d_{goal} = 0 \\ -0.1 & d_{wall} = 0 \\ -0.25 & d_{opp} = 0 \\ -0.1 - \dfrac{1 - d_{opp}}{4} & d_{opp} < 0.2 \\ -0.001 & else \end{cases} \tag{9}$$

where d_{goal}, d_{wall}, and d_{opp} are distances between learning agent and goal, wall and the opponent respectively. Agent is penalized for collisions with the opposing agent or the walls. Also, dynamic negative reward is added for near-collisions between the agent to discourage unsafe states. To encourage faster task completion small negative reward is added to each step. Furthermore, to increase the penalty on unsuccessful episodes we terminate the episode if the accumulated reward exceeds -1 and append additional negative reward of -1. Thus, constructed function fulfills both requirements. It encompasses the collision avoidance problem while providing no feedback for task completion steps.

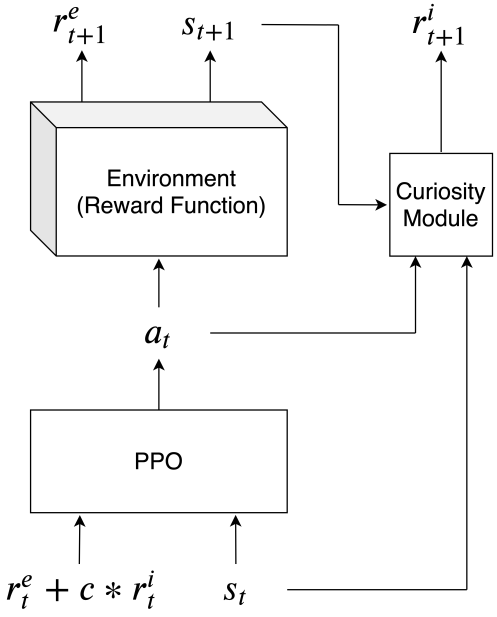
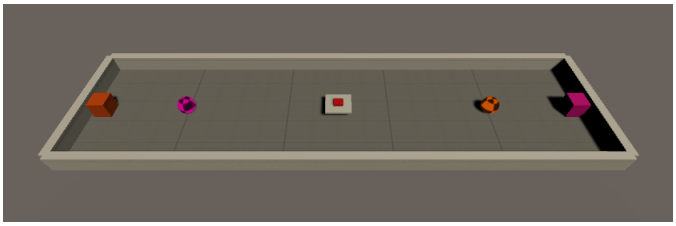
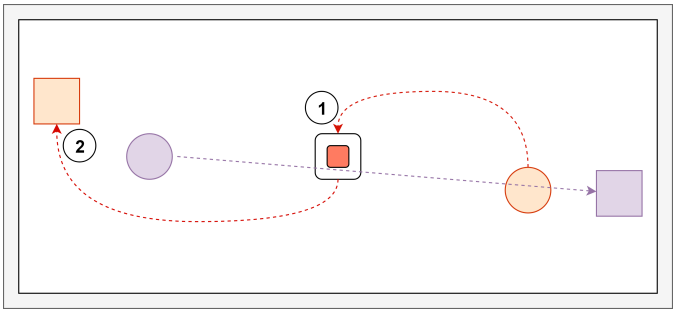


Fig. 3: Block diagram of the complete system.



(a) 3D simulation



(b) 2D representation

Fig. 4: 3D simulation and 2D representation of experiment setup. Learning agent is represented as orange sphere, and its target as orange block. Opposing agent is represented as purple sphere, and its target as purple block. Learning agent should trigger switch (1) before reaching target (2). Opponent goes straight to its target without any consideration.

IV. Case Study

The validity of our method is verified by conducting an experiment. To create an environment that encompasses all of the requirements we build a physical simulation in the Unity3D engine. Aforementioned physical simulation represents two agents in a shared corridor. Purple sphere is the non-communicating goal oriented agent, heading to its target block without considering the other agent. Learning agent is represented in orange color. To make the task more challenging, learning agent has to trigger the switch in the middle before reaching its target block.

Agent centric state space that we input into the algorithm is constructed as described in equation (8).

$$s = [p, V, p_g, d_g, sw_{ind}, p_{op}, d_{op}, V_{op}] \tag{10}$$

where p, p_g, and p_{opp} are positions of learning agent, its goal and the opponent respectively. V and V_{op} are agents' velocities, while sw_{ind} is a boolean value that indicates if the switch has been triggered. Note that the learning agent only has the information on observable states of the opposing agent. We construct the reward function as per requirements described in Section III. It's important that we do not include any reward signal for just flipping the switch, as we want the agent that is able to explore the solutions to complex tasks on its own. Furthermore, we can see from the state vector s in equation (10) that we are just including the indicator value of the switch, not its location or distance to the agent.

To train the agent we use wrappers provided by Unity3D ML-Agents [20]. The environment wrapped in OpenAI Gym [21] is trained using Tensorflow [22]. PPO implementation is based on Stable Baselines implementation [19]. Models were trained on *Windows 64-bit OS* with *Intel(R) Core(TM) i5-6500 CPU* and *8GB RAM*. All trainings were conducted for *1.5M* episodes. Average training time for the proposed method, on a single environment, was *1hr54min*.

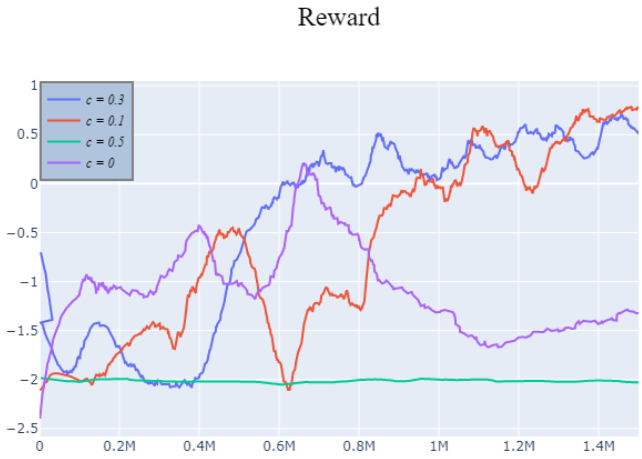
Two case studies are conducted. First one verifies the addition of intrinsic reward and explores the influence of curiosity ratio c on training. Second case study demonstrates the improvements on agent performance by adding an element of penalty to reward function.

A. *Influence of curiosity ratio*

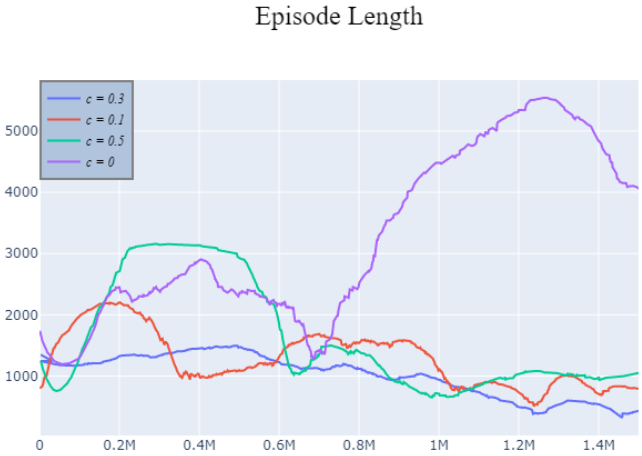
Learning agent is trained with four different values of hyperparameter c. This constant reduces the intrinsic reward signal before it is added to the extrinsic one and input in PPO algorithm. Reward returns are represented in Figure 5a. It is apparent that base PPO without intrinsic reward is unable to complete the task. Two other cases with different values of $c = 0.3$ and $c = 0.1$ have seemingly similar performance. If we consider number of steps necessary to complete the task, represented in Figure 5b, we can see that the agent trained with $c = 0.3$ has slightly better performance. However from the training run with $c = 0.5$, it is obvious that intrinsic reward can easily overwhelm extrinsic reward. This in turn leads to unwanted behavior. If overloaded with curiosity, the agent will perform the actions that yield more intrinsic reward, and forgo extrinsic reward. From Figure 5a it is apparent that the overwhelmed agent seeks to end episode as quickly as possible in order for the environment to reset and supply new, curious observations.

B. *Influence of added penalty in Reward Function*

For this case study we will again consider the most successful agent from Section IV-A. We train another agent with same set of hyperparamaters, but without added negative penalty of -1 that is added to every unsuccessful episode in the reward function (9). Base PPO model is trained in the same fashion only with the exception of $c = 0$. It is again apparent that the base PPO model is not able to handle complexity of the task and fails to find a solution. If we consider Figure 6a, agent trained with added penalty starts with lower return than the one without the penalty. However, this penalty quickly incentivizes the model to find a solution to the task. Furthermore, this solution is better than the one obtained by the model without the penalty. This is also apparent when we consider the respective Episode Lengths in Figure 6b. Also it is worth to note that because of dynamic negative reward in equation (9), agents are not able to obtain a perfect score. Still, this dynamic element is important because it limits unsafe behavior and some unwanted strategies (i.e. crashing into other agent and then continuing to target).



(a) External reward

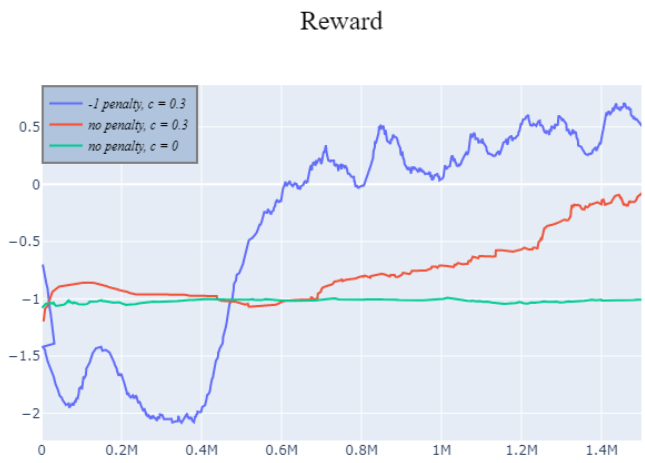


(b) Episode Length in steps

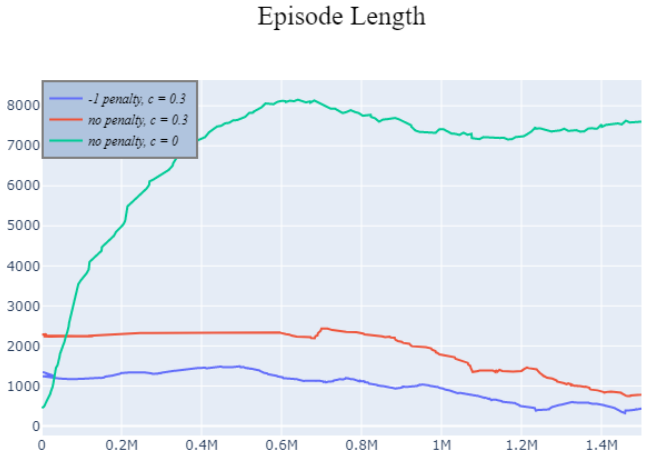
Fig. 5: Reward and Episode Length for different hyperparameter c. $c = 0.3$ in blue. $c = 0.1$ in red. $c = 0.5$ in green. Base PPO without curiosity as comparison in purple.

V. Conclusion

This work developed a method for motion planning in multi-agent environment based on Deep Reinforcement Learning approach and inspired by modeling of artificial curiosity. Additionally, general reward function was constructed to address low-level control. The approach has been shown to be robust to requirements of collision avoidance and task completion settings. Case studies demonstrated the validity of our approach, as state-of-the-art algorithm without extensions was not able to find solution to the experiment problem. Possible extensions to this work could include extensions for Human-Robot Interaction settings and reevaluating the state-action value function introduced by intrinsic reward signal.



(a) External reward



(b) Episode Length in steps

Fig. 6: Reward and Episode Length for different reward functions. Added penalty of -1 in blue. No added penalty in red. Base PPO without curiosity or added penalty as comparison in green.

ACKNOWLEDGMENT

This work is supported by National Nature Science Foundation under Grant (61773260, 61590925); National Key R&D Program of China (2018YFB1305902).

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[2] A. Irpan, "Deep reinforcement learning doesn't work yet," https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018.

[3] H. van Hasselt and M. A. Wiering, "Reinforcement learning in continuous action spaces," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, April 2007, pp. 272–279.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2015.

[5] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016. [Online]. Available: http://arxiv.org/abs/1602.01783

[6] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, "Trust region policy optimization," *CoRR*, vol. abs/1502.05477, 2015. [Online]. Available: http://arxiv.org/abs/1502.05477

[7] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[8] R. W. White, "Motivation reconsidered: The concept of competence," *Psychological Review*, vol. 66, no. 5, pp. 297–333, 1959.

[9] D. E. Berlyne, "Curiosity and exploration," *Science*, vol. 153, no. 3731, pp. 25–33, 1966. [Online]. Available: https://science.sciencemag.org/content/153/3731/25

[10] J. Schmidhuber, "Formal theory of creativity, fun, and intrinsic motivation (1990–2010)," *IEEE Trans. on Auton. Ment. Dev.*, vol. 2, no. 3, pp. 230–247, Sep. 2010. [Online]. Available: https://doi.org/10.1109/TAMD.2010.2056368

[11] R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *In Proceedings of the Seventh International Conference on Machine Learning*. Morgan Kaufmann, 1990, pp. 216–224.

[12] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven exploration by self-supervised prediction," in *ICML*, 2017.

[13] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-scale study of curiosity-driven learning," in *arXiv:1808.04355*, 2018.

[14] Y. Burda, H. Edwards, A. J. Storkey, and O. Klimov, "Exploration by random network distillation," *CoRR*, vol. abs/1810.12894, 2018. [Online]. Available: http://arxiv.org/abs/1810.12894

[15] M. Frank, J. Leitner, M. Stollenga, A. Förster, and J. Schmidhuber, "Curiosity driven reinforcement learning for motion planning on humanoids," *Frontiers in Neurorobotics*, vol. 7, p. 25, 2014. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnbot.2013.00025

[16] T. J. Prescott and J. E. W. Mayhew, "Obstacle avoidance through reinforcement learning," in *Proceedings of the 4th International Conference on Neural Information Processing Systems*, ser. NIPS'91. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1991, pp. 523–530. [Online]. Available: http://dl.acm.org/citation.cfm?id=2986916.2986980

[17] K. Macek, I. Petrovic, and N. Peric, "A reinforcement learning approach to obstacle avoidance of mobile robots," in *7th International Workshop on Advanced Motion Control. Proceedings (Cat. No.02TH8623)*, July 2002, pp. 462–466.

[18] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," *CoRR*, vol. abs/1609.07845, 2016. [Online]. Available: http://arxiv.org/abs/1609.07845

[19] A. Hill, A. Raffin, M. Ernestus, A. Gleave, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.

[20] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *CoRR*, vol. abs/1809.02627, 2018. [Online]. Available: http://arxiv.org/abs/1809.02627

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/