Enhancing Exploration through Deep Learning in SL Enviroment

Shuo Zhang
Technical University Darmstadt
Karolinenplatz 5, 64283 Darmstadt
zhangshuo19930709@gmail.com

Lu Wan
Technical University Darmstadt
Karolinenplatz 5, 64283 Darmstadt
wlkobe@gmail.com

Abstract

The principle of reinforcement learning(RL) algorithms is to achieve the goal by maximizing the rewards from the environment. However, generally the definition of dense reward is a complicated problem. Such as in the case of exploration within multiple balls, it is not easy to find the best definition of dense rewards, and the reward in finding simple one ball task can also not be scaled in the complex task. The intrinsic reward, which is generated by the inner curiosity, enables the agent to explore the environment itself, and thus is more adaptable to the sparse reward scenarios. The Intrinsic Curiosity Module(ICM) is such a intrisic reward generator, which was before shown to perform well on tasks of discrete action space. Therefore, in continuous action space, we are also motivated to build and evaluate the Intrinsic Curiosity Module(ICM) based on Proximal Policy Optimization Algorithms (PPO) to yield the inner curiosity for enhancing the exploration of robot-arm. Since no work with deep RL has been done on SL, we have firstly tried to implement deep RL algorithms, namely Deep Deterministic Policy Gradient(DDPG) and PPO, in SL with a simple task of finding one ball. Then we evaluated and compared their performance in SL. Furthermore, on the task of finding the optimal ball among multiple balls, we validated the effectiveness of curiosity-driven exploration with PPO-based ICM in continuous action space. Finally, we realized the task of reaching the optimal ball among multiple balls while meanwhile avoiding the obstacles by enhancing exploration. Overall, the algorithms we have investigated are DDPG, PPO and curious ICM based on PPO.

1 Introduction

1.1 Motivation

SL simulator[1] from is a powerful robot-arm evaluation environment developed by the IAS at TU-Darmstadt. Currently it is available for many classical control methods such as PID control. However, with respect to the generality to a complex environment or a problem not fully known, all classical methods including "Linear Quadratic Regulator"(LQR) controller[2], are inferior to the reinforcement learning(RL) method[3]. In SL, though reinforcement learning is possible through the communication with MATLAB or Python, the most popular method in recent years, deep learning[4] with reinforcement learning, is not yet able to be implemented in SL. Thus, our first aim is to implement deep reinforcement learning methods in SL.

RL algorithms learn policies for achieving the goal by maximizing rewards provided by the environment. However, in many real world scenarios, extrinsic rewards from environment may be hard to be shaped densly, only have sparse from or be missing at all, so that a shaped reward function can not be formed. In such cases, there will be a problem, which is the agent will update its policy only if it succeeds in finding the goal. Thus, a random exploration will be far from efficiency. Thus, an

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

intrinsic reward is also necessary to be added as a part of the total reward to make the agent curious for such cases. The curiosity, as the intrinsic reward, has been formulated differently in recent papers. One way is to use "visitation counts"[5], which encourages the agent to visit novel states. Another way is to use "prediction error"[6] as intrinsic rewards, which encourages the agent to perform actions leading to a big prediction error of the actions' consequence, while the error/uncertainty in the agent's prediction the consequence of its own actions should be reduced at the same time. A successful prediction error model for curiosity has been proposed in 2017 as "Intrinsic Curiosity Model"(ICM)[7]. Furthermore, an extension to even the case of no extrinsic reward by using only ICM has been investigated[8], showing surprisingly good performance. Though algothrims working also for continuous action space have been used in these two papers, none of them has reported an evaluation of the exploration with ICM on a continuous action space. Thus, our second goal of this paper is to enhance curiosity-driven exploration with ICM in a continuous action space.

1.2 Reinforcement Learning Algorithms

In recent years, many algorithms in reinforcement learning has been designed. Apart from typical methods such as Q-Learning and SARSA[3], the "Deep Q Network"(DQN) algorithm[9] firstly combined deep learning and reinforcement learning and was very successful due to the capability of performing well on many Atari video games using raw pixels as input. However, these are all for the applications in the case of discrete action space. The "Deep Deterministic Policy Gradient"(DDPG) algorithm[10] in 2015 extended the high level performance firstly even to the tasks with high dimensional, continuous action space. Results after training of 2.5 million steps turned to be very good on many benchmark tasks, such as Pendulum. Meanwhile, the "Trust Region Policy Optimization"(TRPO) algorithm[11], which also works for continuous action space, was shown to be effective for optimizing large nonlinear policies such as neural network and to tend to give monotonic improvement, with little tuning of hyperparameters. Furthermore, the "Proximal Policy Optimization"(PPO) algorithm[12], suggests to have some of the benefits of TRPO. But it is simpler to implement, more general, and has better sample complexity empirically.

1.3 Task Statement

For the implementation of deep reinforcement learning, We have defined our first task as making the robot-arm in SL successfully learn to find a ball. Moreover, our task was later extended to a more complex scene including multiple balls with different rewards, and the scene was further extended to include multiple obstacles as well, though minor adjustment for collision check for each link is still to be done in the future. The comprehensive environment in SL look like Figure 1.

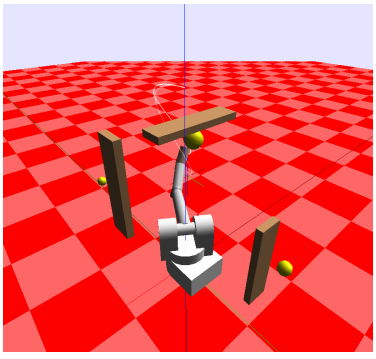


Figure 1: A complex sample scene in SL

Though our ultimate goal in the future is to implement a deep reinforcement learning algorithm on a real robot with high dimensional raw pixels as state information so that it can find the ball with maximal reward in such a complex scene, as a first step, we only chose to use low dimensional states, which are joint degrees, joint velocities, the cartesian position and velocity of the end-effector, to test our implementation of deep learning algorithms in SL simulation. Besides, we defined the joint acceleration as our action space. So, both state space and action space in our task are continuous,

which motivates us to use DDPG and PPO as our algorithms and to compare them with each other. TRPO is dumped due to its computational cost and complexity.

The organization of this paper is as followings: we introduced our simulation environment including its communication with Python in Subsection 2.1. Then in Subsection 2.2, theories regarding to DDPG, PPO and curiosity exploration with ICM were explained. In Section 3, experiments setup including definition for different reward settings, the comparison result of DDPG and PPO in a dense reward setting for finding simple one ball, the result of enhancing curiosity-driven exploration with PPO-based ICM in sparse reward setting for finding optimal ball among multiple balls and the result of achieving the optimal ball in the most complex scene including obstacles were respectively explained. Finally, we made conclusions in Section 4.

2 Methods

The goal of SL is to have a common platform that allows for easy implementation, evaluation and comparison of different robot evaluation platforms, experiments and learning algorithms. We have chosen Barrett WAM in SL, which is a robot-arm with 7 degrees of freedom.

2.1 Structure of the System

The overall system has 2 main parts. One is the SL simulation environment programmed in C, another one is the learning algorithm in Python. SL processes task management, robotics kinematic and dynamic calculation, motor control, visualization and etc. All the data will be saved into and read from the shared memory, which is used to interact with the algorithm in python. The deep reinforcement learning algorithm is written in Python. As the world's fastest growing programming language, Python is suitable for implementing the algorithm of machine learning because of its readability, versatility, easiness and possession of lots of packages, such as tensorflow, numpy, matplotlib and etc.

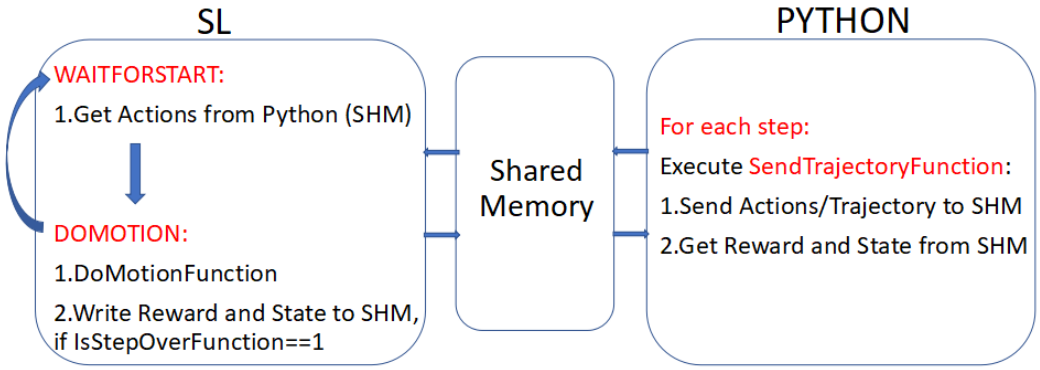


Figure 2: Communication flow between C and Python

2.2 Theory

The normal reinforcement learning considers an agent that interacts with an environment E in an infinite horizon. At time step t, it sees an observation $\mathbf{s}_t$, takes an action $\mathbf{a}_t$ which comes from a stochastic policy distribution $\pi(\mathbf{a}_t|\mathbf{s}_t) = \mathbb{P}(\mathbf{a}_t|\mathbf{s}_t)$, receives then a reward r_t with discount factor γ, and finally goes to the next state $\mathbf{s}_t + 1$. We further define the state value function $V(\mathbf{s}_t)$, the state-action value function $Q(\mathbf{s}_t, \mathbf{a}_t)$ and the advantage function $A(\mathbf{s}_t, \mathbf{a}_t)$ as followings[3]:

$$V(\mathbf{s}_t) = \mathbb{E}_{\mathbf{s}_{t+1:\infty},\mathbf{a}_{t:\infty}}\left[\sum_{l=0}^{\infty}\gamma^l r^{t+l}\right] \tag{1}$$

$$Q(\mathbf{s}_t,\mathbf{a}_t) = \mathbb{E}_{\mathbf{s}_{t+1:\infty},\mathbf{a}_{t+1:\infty}}\left[\sum_{l=0}^{\infty}\gamma^l r^{t+l}\right] \tag{2}$$

$$A(\mathbf{s}_t,\mathbf{a}_t) = Q(\mathbf{s}_t,\mathbf{a}_t) - V(\mathbf{s}_t) \tag{3}$$

2.2.1 DDPG Algorithm

The crux of the DDPG algorithm is the architecture of actor and critic. There are two actors and two critics, respectively for evaluation and target. The evaluation actor network $\pi(\mathbf{s}|\theta^\pi)$ learns the action for the current state $\mathbf{s}$, while the target actor network $\pi'(\mathbf{s}'|\theta^{\pi'})$ learns the action for the next state $\mathbf{s}'$. The evaluation critic network $Q(\mathbf{s},\mathbf{a}|\theta^Q)$ learns the state-action value for the current state $\mathbf{s}$ and action $\mathbf{a}$, while the target critic network $Q'(\mathbf{s}',\mathbf{a}'|\theta^{Q'})$ learns the action for the next state $\mathbf{s}'$ and action $\mathbf{a}'$, which comes directly from target actor $\pi'(\mathbf{s}'|\theta^{\pi'})$. The DDPG algorithm also uses an exploration strategy, which is to add an action noise $\mathcal{N}$ to the output of evaluation actor $\pi(\mathbf{s}|\theta^\pi)$ to explore more. The noise usually can be set as *OrnsteinUhlenbeckActionNoise*.

To optimize all networks, the DDPG uses the following strategy: fisrtly to update the evaluation critic by minimizing the loss L in equation 4, where N is the batch size. Then, update the evaluation actor using the policy gradient $\nabla_{\theta^\mu}J$ in equation 5, where J is the score function. Finally, $\theta^{Q'}$ and $\theta^{\mu'}$ will be softly updated as equations 6 and 7.

$$L = \frac{1}{N}\sum_i (r_i + \gamma Q'(\mathbf{s}_{i+1}, \pi'(\mathbf{s}_{t+1}|\theta^{\pi'})|\theta^{Q'}) - Q(\mathbf{s}_i,\mathbf{a}_i|\theta^Q))^2 \tag{4}$$

$$\nabla_{\theta^\mu}J = \frac{1}{N}\sum_i \nabla_{\mathbf{a}}Q(\mathbf{s},\mathbf{a}|\theta^Q)|_{\mathbf{s}=\mathbf{s}_i,\mathbf{a}=\mu(\mathbf{s}_i)}\nabla_{\theta^\mu}\mu(\mathbf{s}|\theta^\mu)|\mathbf{s}_i \tag{5}$$

$$\theta^Q \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'} \tag{6}$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'} \tag{7}$$

2.2.2 PPO Algorithm

The PPO algorithm has also two networks. One is the state value function network $V(\mathbf{s})$ which outputs the state value, the other one is policy network $\pi(\mathbf{s})$ which determines the action. An action will be drawn from the Gaussian distribution $\pi_\theta(\mathbf{a}|\mathbf{s})$ with the mean of the output of $\pi(\mathbf{s})$ and the standard deviation of a noise also to be learned. Besides, the TD residual δ_t at time step t and the generalized advantage estimator $A_t^{\texttt{GAE}}$ [13] are as followings, where λ stands for eligibility trace:

$$\delta_t = r_t + \gamma V(\mathbf{s}_{t+1}) - V(\mathbf{s}_t) \tag{8}$$

$$A_t^{\texttt{GAE}} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l} \tag{9}$$

In fact, if we have the correct value function V, then an unbiased estimator of advantage A_t will be just $\mathbb{E}_{\mathbf{s}_{t+1}}[\delta_t]$. So, the first update is to minimize the loss of network $V(\mathbf{s})$, which equals to $A_t^{\texttt{GAE}}$. Then, the policy network $\pi(\mathbf{s})$ will be learned through maximizing the following objective function $L(\theta)$, where ϵ is a hyperparamter and $r_t(\theta)$ is the ratio between old and new policy:

$$r_t(\theta) = \frac{\pi_\theta(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\theta_{\texttt{old}}}(\mathbf{a}_t|\mathbf{s}_t)} \tag{10}$$

$$L(\theta) = \mathbb{E}_t\left[\min(r_t(\theta)A_t^{\texttt{GAE}}, \texttt{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t^{\texttt{GAE}})\right] \tag{11}$$

2.2.3 Curiosity-driven Exploration with ICM

ICM also has two parts. One is the inverse dynamics model, which is to learn the general feature $\phi(\mathbf{s})$ of a state $\mathbf{s}$. Inverse model takes the current state $\mathbf{s}_t$ and the next state $\mathbf{s}_{t+1}$ as inputs, converts them to their feature representation and finally outputs an action $\hat{\mathbf{a}}_t$. The model will be updated through the minimization of the loss function L_I between the real action $\mathbf{a}_t$ and the output action $\hat{\mathbf{a}}_t$ of itself, the inverse dynamics model. Another one is the forward dynamics model. Forward model takes the current state $\mathbf{s}_t$ and the real action $\mathbf{a}_t$ as inputs and outputs the which is to learn the prediction $\hat{\phi}(\mathbf{s}_{t+1})$ of the feature of the next state$\mathbf{s}_{t+1}$. The model will be updated through the minimization of the loss function L_I between the real feature $\phi(\mathbf{s}_{t+1})$ of the next state $\mathbf{s}_{t+1}$ and the predicted feature $\hat{\phi}(\mathbf{s}_{t+1})$. Based on this, the intrinsic reward r_t^i is defined as the difference between these two features. The following equations show the definition of the intrinsic reward, where η is a scaling factor, and the minimization of the total loss function. Finally, the output of ICM will be added to the extrinsic reward as the total reward. The flow is shown in Figure 3.

$$r_t^i = \frac{\eta}{2}||\hat{\phi}(\mathbf{s}_{t+1}) - \phi(\mathbf{s}_{t+1})||_2^2 \tag{12}$$

$$\min_{\theta_P,\theta_I,\theta_F} \left[-\lambda \mathbb{E}_{\pi(\mathbf{s}_t;\theta_P)}[\sum_t r_t] + (1-\beta)L_I + \beta L_F \right] \tag{13}$$

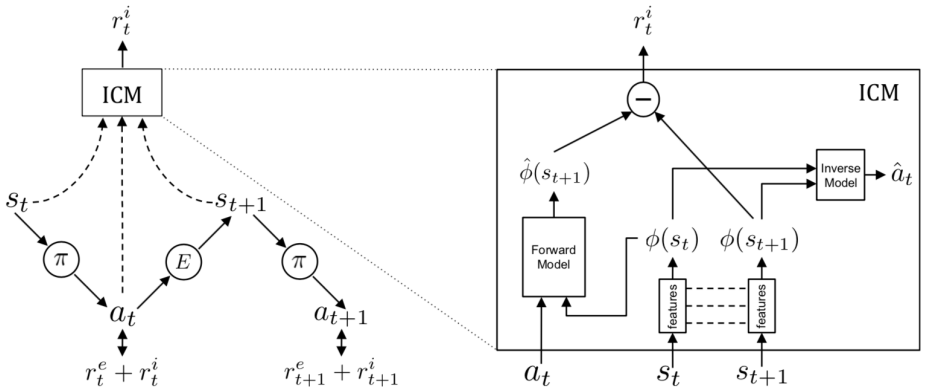


Figure 3: Curiosity driven exploration flow with ICM

3 Experiments

To evaluate our implementation of deep reinforcement learning algorithms in SL and furthermore the curiosity model's ability to improve exploration, we have designed experiments as followings. The following section will describe the details of the experimental setup, the reward setup of environments, and the hyperparameters' setup.

3.1 Experimental Setup

We evaluated the performances of the DDPG and PPO policy with and without the Intrinsic Curiosity Module (ICM) qualitatively and quantitatively. For all experiments, an average result was taken by 10 evaluation episodes for each iteration, each of which trained networks for 5 episodes(2500 training steps). Afterwards we calculate the average total reward and count the time of reaching the ball as the evaluation of the results.

In Subsection 3.2, the minus-defined dense reward is the addition of the distance, velocity and acceleration with different weights, respectively 15, 0.3 and 0.001. In Subsection 3.3, the respective sparse reward is only be provided when the the robot arm reaches any of the balls in the environment. In this case, there is no terminal if the robot arm reaches the goal, because we want to control the end-effector finally stays at the optimal ball's position as long as possible. In the case of Subsection

3.4, the reward is also set to be sparse, only be positive when it reaches a ball or negative when touches an obstacle.

Besides, β was set to be 1 in the case of ICM exploration. Because we have already used information sufficient and low dimensional states, the inverse dynamics model for learning the feature space of states was not necessary, i.e. the feature space could be the states itself. All common hyperparameters of algorithms during the experiments are the same, which can be read from the following Table 1.

Table 1: Hyperparameters in DDPG, PPO and Curious ICM-PPO

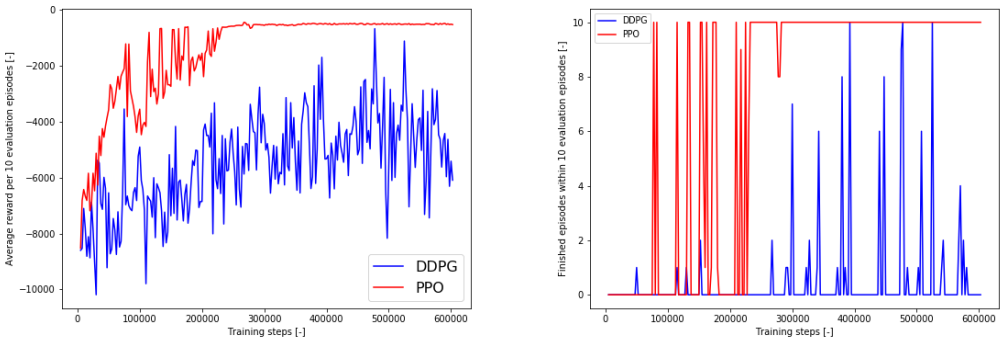
Parameters	DDPG	PPO	Curious ICM-PPO
Warmup steps	10000	-	-
Maximal learning steps	600,000	600,000	600,000
Maximal steps per episode	500	500	500
Batch size	64	64	64
Action bound	20	20	20
Reward discount factor γ	0.99	0.99	0.99
Network Size of $\pi(s)$	[200 200]	[200 200]	[200 200]
Network Size of $Q(s, a)$	[200 200]	-	-
Network Size of Value Function	-	[200 200]	[200 200]
Network Size of ICM Network	-	-	[200 200]
Optimizer of $\pi(s)$	RMSProp	RMSProp	RMSProp
Optimizer of $Q(s, a)$	RMSProp	-	-
Optimizer of Value Function	-	RMSProp	RMSProp
Optimizer of ICM Network	-	-	RMSProp
Learning rate of $\pi(s)$	0.0003	0.0003	0.0003
Learning rate of $Q(s, a)$	0.0003	-	-
Learning rate of Value Function	-	0.0003	0.0003
Learning rate of ICM Network	-	-	0.0003
Update coefficient $\tau_Q$of $Q'(s', a')$	0.001	-	-
Update coefficient τ_π of $\pi'(s')$	0.001	-	-
Scaling factor η	-	-	2
Weight factor β	-	-	1
Clip factor ϵ	-	-	0.2

In Subsection 3.2, we compared the performance of DDPG and PPO for reaching a ball. The extrinsic reward is dense reward. In Subsection 3.3, the PPO algorithm will face a bigger challenge, the sparse extrinsic reward provided only by multiple balls with different rewards, which is expected to be improved by ICM curiosity-driven exploration. In Subsection 3.4 we will use the curious ICM-PPO agent to finish the most complex task, which aims to find the biggest ball among the multiple balls and avoid the obstacles at the same time.

3.2 Performance in Dense Reward Setting

The DDPG and PPO are the two important reinforcement learning algorithms in the continuous action domain. First of all, we validated the algorithms with understandable dense extrinsic reward, to confirm if the algorithms can be used in a similar or more complex robot task. In comparison with the performance of the DDPG agent against the PPO agent, we have plotted the results of average total rewards with regard to the training steps in the Figure 4a below, and the results of the finished times in each 10 episodes with regard to the training steps in the Figure 4b. Due to the time urgency, each experiment was executed twice.

As the results are shown, all the groups are able to learn the skill as showing an increasing average rewards. However, the PPO agent learned obviously much faster than DDPG and its reward plot in Figure 4a had much less fluctuations. Moreover, we can see from Figure 4b that DDPG can not even find the optimal trajectory within 600,000 training steps, while PPO finds the optimal solution around 230,000 training steps showing a convergence speed more than twice faster than DDPG. This could be explained as it has been proposed in [12] that PPO has a characteristic of continuous improvement, and in [14] that DDPG suffers from inefficiency in the case of RL problems with multidimensional



(a) Comparison of average total reward

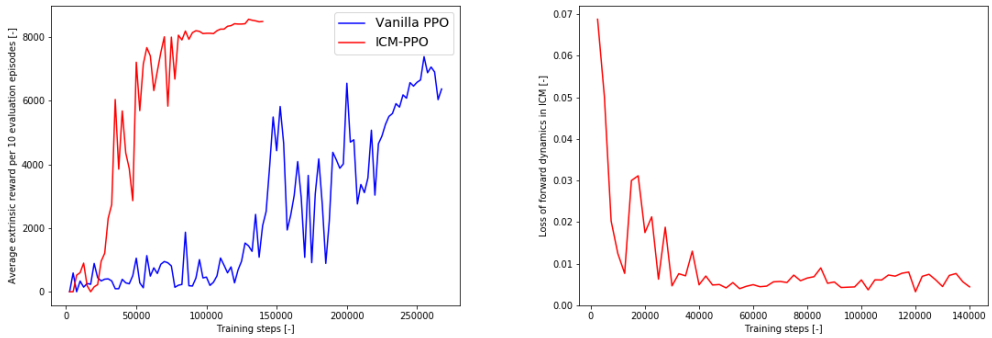
(b) Comparison of finished times

Figure 4: Comparison between DDPG and PPO for finding simple one ball

continuous actions, where exploration can be particularly challenging, e.g. in robotics. Therefore, we have attested this and prefer PPO algorithm for our further works.

3.3 Exploration in Sparse Reward Setting

Varying to a much higher degree of reward sparsity can verify the effect of curious ICM model. Therefore, we performed the sparse extrinsic reward experiments with and without ICM for finding the optimal ball among 3 balls to test the effect of adding curiosities. The results based on PPO are shown in Figure 5a and Figure 5b, respectively for comparison of average extrinsic reward and ICM forward dynamics' loss L_F. Due to the time urgency, each experiment was executed once, the goal range for each ball was set to be 6 times wider than in Subsection 3.2, and the observed evaluation was confined to the training steps at which the end-effector converged to the second biggest ball. For more curiosity, only in this case we have set η=6.



(a) Comparison of average extrinsic reward

(b) Loss function L_F of ICM forward dynamics

Figure 5: Results of PPO and ICM-PPO in a scene including multiple balls

The Figure 5a demonstrated that ICM-PPO, which converged to the second biggest ball after 150,000 training steps, explored the space much more efficiently and faster than vanilla PPO, which performed worse even after 250,000 steps. We indeed observed the end-effector tended to explore more during the whole training process with ICM-PPO than with vanilla PPO. As seen from the Figure 5a, there was an obvious fluctuation of L_F in ICM-PPO between 20,000 steps and 40,000 steps, which led to a better exploration at that time than vanilla PPO, also shown in the Figure 5a. Thus, combining ICM as motivation for exploration had a better performance than PPO also in our continuous action space.

Definitely, due to the not yet investigated uncertainty, we need run more experiments, more iterations and in a more strict goal setting to finally affirm the superiority of ICM-PPO. But according to the original ICM paper [7], ICM-integrated methods works surely better than the algorithms using only sparse extrinsic rewards. So, our evaluation extending to the continuous action space, which shows a similar tendency as in [7], is also likely to be reliable, though further experiments are necessary.

3.4 Exploration within Complex Scenes

After attesting the effectiveness of ICM-PPO, we have finally made a pretest on our ultimate task which is the most complex scene as in the Figure 1 containing 3 balls and 3 obstacles. Because of not having built a collision check function for each robot link in SL, we just simply used the collision check only between the obstacle and end-effector, which must be improved and finished in the future. Still, we got a satisfactory result of average total rewards as shown in the Figure 6.

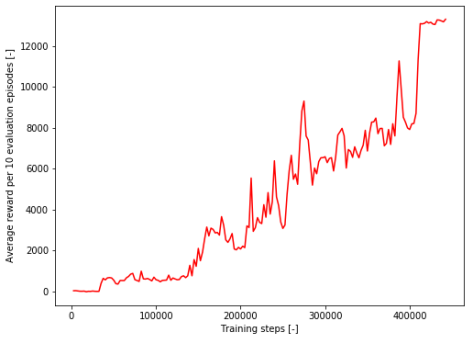


Figure 6: Average total reward in a scene including multiple balls and obstacles

From the result of Figure 6, we can see that after 500,000 training steps the average reward has climbed to over 12000, which shows the end-effector getting close to the ball with the maximal reward. Also, we observed end-effector's collisions in the first several training episodes in the SL simulator, while later it became less and less. Furthermore, we actually indeed saw in the simulator that the robot-arm guided itself safely to the neighbour of the ball with the maximal reward, while the arm gradually abandoned the states near sub-optimal balls. We confirmed that the arm explored novel and better states when the exploration loss of forward dynamics model changed much during the evaluation. Overall, we can also say that the ICM also works well for the task of continuous action space.

4 Conclusions

In this work, after building a step-wise communication interface between SL environment and Python, we successfully implemented and compared the DDPG algorithm and the PPO algorithm on the task of finding a desired ball. The PPO significantly outperformed the and DDPG in the case of a robot arm with 7 degrees of freedom. Moreover, we demonstrated that the curiosity-driven exploration ICM-integrated method enhanced the exploration of the robot arm also in a continuous action space on the task of finding the optimal ball among multiple balls, and the curious ICM-PPO could be applied further in the most complex environment including obstacles.

An interesting direction of future research is to implement the algorithms to the real robot. Then somes aspects as followings need to be further considered: firstly, with the same low dimensional state information as in our simulation, a collision check program in SL for all links of robot-arm need to be fully developed. Secondly, a high-dimensional input will be the real case, which has to be dealt with. Thirdly, the inverse dynamics model of the high dimensional space must be learned to acquire a sufficiently good feature space for implementing ICM-exploration. Fourthly, the safety of the real robot-arm during training and evaluation should be seriously considered as well.

References

[1] Daniel Christian, Neumann Gerhard, and Peters Jan. *CompLACS Robot Arm Evaluation Scenarios*, 2018.

[2] Pierre OM Scokaert and James B Rawlings. Constrained linear quadratic regulation. *IEEE Transactions on automatic control*, 43(8):1163–1169, 1998.

[3] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 1998.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[5] Georg Ostrovski, Marc G Bellemare, Aaron van den Oord, and Rémi Munos. Count-based exploration with neural density models. *arXiv preprint arXiv:1703.01310*, 2017.

[6] Bradly C Stadie, Sergey Levine, and Pieter Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[7] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 488–489. IEEE, 2017.

[8] Yuri Burda, Harri Edwards, Deepak Pathak, Amos Storkey, Trevor Darrell, and Alexei A. Efros. Large-scale study of curiosity-driven learning. In *arXiv:1808.04355*, 2018.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[10] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[11] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

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

[13] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[14] Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. Gep-pg: Decoupling exploration and exploitation in deep reinforcement learning algorithms. *arXiv preprint arXiv:1802.05054*, 2018.