ME491: Final Term Project Report

20213680 Jungwoo Han

---

1. Introduction

Designing a framework to control a robot in a complex environment is a challenging task. Nowadays, many researchers and engineers have come up with deep Reinforcement Learning (RL) frameworks that outperform traditional robotic control methods. Considering such trend, focus of this project is to develop an algorithm that controls a 4-wheeled robot, husky, to the designated destination (center of the environment) based on the deep RL. The environment is full of obstacles, and the robot has no prior knowledge regarding the environment. As a metric to measure performance of the algorithm is the time taken to reach the goal, the robot should learn from scratch how to efficiently avoid the obstacles based on the received observations to minimize the travelling time. Most importantly, total number of samples that could be used for the training is limited to 20M, which is equivalent to the sampling from 250k different environments (maximum of 80 steps for each episode).

In section 2, I will briefly explain the RL algorithms that are used in the project. In section 3, I will introduce how I designed my observation, reward, and action spaces to train the robot. In section 4 and 5, I will present methods that I used to improve the performance, and the corresponding results, respectively.

2. Related Works

Proximal Policy Optimization (PPO)

PPO is a state-of-the-art on-policy algorithm for learning a continuous or discrete policy. PPO updates the policy using action-advantages, $A_t = A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t)$, and the objective function shown below.

$$\mathcal{L}_\pi(\theta) = -\mathbb{E}_{\tau\sim\pi}\left[\min\left(\rho_t(\theta)A_t,\ \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)A_t\right)\right], \quad \rho_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}$$

Value function which is required to calculate the action-advantages is also optimized using the below objective function.

$$\mathcal{L}_V(\phi) = \mathbb{E}_{\tau\sim\pi}\left[\left(V_\phi(o_t) - V_t^{targ}\right)^2\right]$$

Soft Actor Critic (SAC)

SAC is a state-of-the-art off-policy algorithm for continuous controls. SAC learns a policy and a critic by maximizing a weighted objective of the reward and entropy as shown below.

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\rho_\pi}\left[r(\mathbf{s}_t,\mathbf{a}_t) + \alpha\mathcal{H}(\pi(\cdot|\mathbf{s}_t))\right]$$

Critic parameters are learned by minimizing the below squared residual error where Q-function is also parameterized and optimized through MSE with the one-step TD error.

$$J_V(\psi) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{D}}\left[\frac{1}{2}\left(V_\psi(\mathbf{s}_t) - \mathbb{E}_{\mathbf{a}_t\sim\pi_\phi}\left[Q_\theta(\mathbf{s}_t,\mathbf{a}_t) - \log\pi_\phi(\mathbf{a}_t|\mathbf{s}_t)\right]\right)^2\right]$$

Actor parameters are optimized through the reparametrized version of KL-divergence where $\boldsymbol{a}_t = f_\emptyset(\epsilon_t; s_t)$ and ϵ_t is an input noise vector.

$$J_\pi(\phi) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \epsilon_t \sim \mathcal{N}} \left[\log \pi_\phi(f_\phi(\epsilon_t; \mathbf{s}_t) | \mathbf{s}_t) - Q_\theta(\mathbf{s}_t, f_\phi(\epsilon_t; \mathbf{s}_t)) \right]$$

I used an open source for the SAC implementation, which is available at https://github.com/pranz24/pytorch-soft-actor-critic.

3. Observation, Reward, and Action

37-dimensional Observation Space:

1. 7-dimensional generalized coordinate observation is defined with respective to the world coordinate, and it includes the following:
   - x, y, z coordinates
   - w, x, y, z quaternion coordinates

2. 10-dimensional generalized velocity observation is defined with respective to the world coordinate, and it includes the following:
   - x, y, z linear velocities
   - w_x, w_y, w_z angular velocities
   - s_1, s_2, s_3, s_4 wheel velocities

3. 7-dimensional LiDAR observation is defined with respective to body frame of the robot, and the LiDAR points are uniformly distributed over 30° in front of the robot. Maximum range of the LiDAR points is $20m$.

Reward:

1. Consistent negative penalty term is given to the robot which is the norm of its x, y coordinates with respect to the environment center, multiplied by -0.3.

2. Gradual positive reward term for closer approach towards the goal. If the distance between the robot and the goal are $10m$, $5m$, and $2m$, it receives 0.3, 0.2, and 0.1, respectively.

3. Also, to prevent the husky from passing by the goal, consistent penalty term of -0.05 is given. In other words, if the husky reaches the goal and goes farther, -0.06 penalty is given every control step.

Action Space:

1. 4-dimensional action space is defined as torque values for the 4 wheels.

4. Methodologies

8 different methodologies are conducted. For some of them, initial position curriculum learning and hyperparameter scheduling are applied.

Initial Position Curriculum Learning

Default environment makes the robot to be spawned within a range $5 < x, y < 30$ from the origin of the environment. With such condition, it is hard to collect good samples (reaching the goal) during initial training phase. Collecting good samples is especially efficient for off-policy algorithm like SAC as it can use stored past samples for the training.

Therefore, I decided to spawn the robot near the goal for the initial training phase to increase the probability of obtaining good samples. Then, as the training progresses, distance between the spawning location and the goal point is gradually increased.

Hyperparameter Scheduling

This is mainly for PPO. Within the provided code, there is a hyperparameter, num_envs, which determines the number of environments from which the agent collects the samples, in parallel. The default setting is 200; however, the problem is that there would be large number of bad samples (far from the origin) in the initial training phase. As total number of the allowed samples is limited to 20M, it is important to minimize the number of bad samples for efficient training. One way to achieve this is to improve the policy as quickly as possible so that it could start collecting good samples in early phase. If num_envs is decreased, the policy would be updated more frequently, but it compensates robustness of the policy.

Therefore, I decided to start the training with $num\_envs = 50$, and increase it to $num\_envs = 100$ when the robot starts to succeed in reaching the goal.

The conducted methodologies are as follows:

1) **Vanilla PPO**
   A. **Algorithm:** Vanilla PPO algorithm that is provided as a baseline
   B. **Reward:** Only reward of the consistent distance penalty term
   C. **Hyperparameter:** $num\_envs = 200$
   D. Actor network with 2 hidden layers [128, 128] with the activation function of LeakyReLU
   E. Critic network with 2 hidden layers [128, 128] with the activation function of LeakyReLU

2) **Vanilla SAC**
   A. **Algorithm:** Vanilla SAC algorithm with a deterministic policy instead of stochastic policy
   B. **Reward:** Only reward of the consistent distance penalty term
   C. **Hyperparameter:** $num\_envs = 1$
   D. Actor network with 2 hidden layers [128, 128] with the activation function of ReLU
   E. 2 critic networks (parameterized Q-functions) with 2 hidden layers [128, 128] with the activation function of ReLU

3) **Vanilla SAC with initial position curriculum learning**
   A. **Algorithm:** Vanilla SAC algorithm with a deterministic policy instead of stochastic policy
   B. **Reward:** Only reward of the consistent distance penalty term
   C. **Hyperparameter:** $num\_envs = 1$
   D. Initial position curriculum learning:
      i. The spawn range is set as $2 < x, y < 5$, initially.
      ii. After 2000 episodes, lower bound and upper bound are increased by 1 for every 2000 episodes and 200 episodes until they reach 5 and 30, respectively.

4) **PPO with the reward 1, 2, and the initial position curriculum learning**
   A. **Algorithm:** Vanilla PPO algorithm that is provided as a baseline
   B. **Reward:** 1. Consistent distance penalty term 2. Gradual positive reward for the closer approach
   C. **Hyperparameter:** $num\_envs = 100$
   D. Initial position curriculum learning:
      i. The spawn range is set as $2 < x, y < 15$, initially.
      ii. After 2000 episodes, lower bound and upper bound are increased by 1 for every 500 episodes and 100 episodes until they reach 5 and 30, respectively.

5) **PPO with the reward 1, 2, and the hyperparameter of $num\_envs = 50$**
   A. **Algorithm:** Vanilla PPO algorithm that is provided as a baseline
   B. **Reward:** 1. Consistent distance penalty term 2. Gradual positive reward for the closer approach
   C. **Hyperparameter:** $num\_envs = 50$

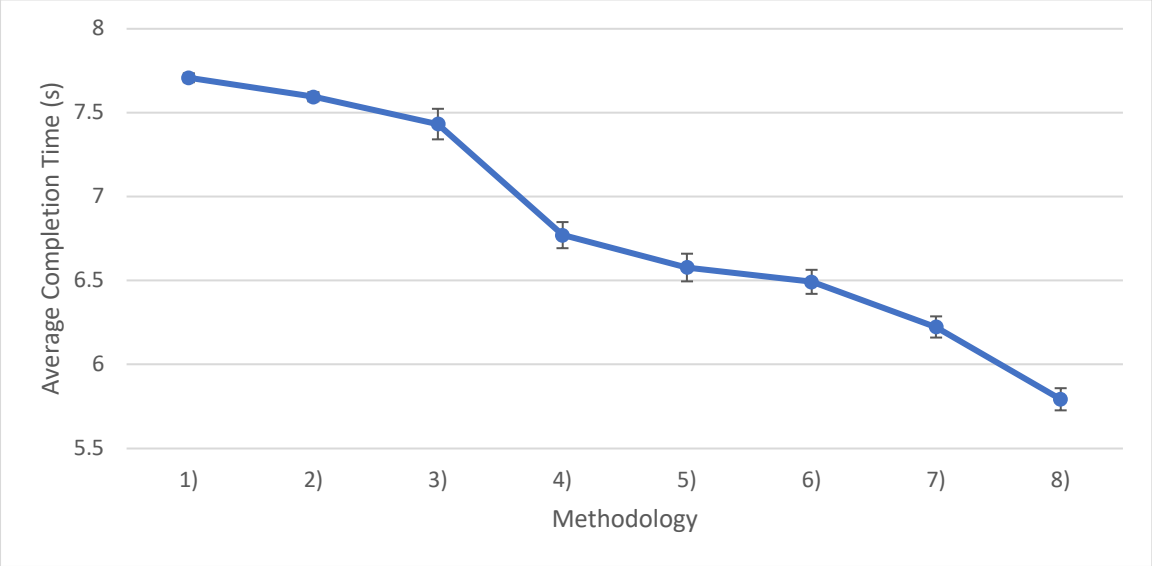
6) **PPO with the reward 1, 2, and the hyperparameter scheduling 1**
   A. **Algorithm:** Vanilla PPO algorithm that is provided as a baseline
   B. **Reward:** 1. Consistent distance penalty term 2. Gradual positive reward for the closer approach
   C.
   D. **Hyperparameter:** $num\_envs = 50$ is increased to 100 when 10M samples are collected. In other words, half of the training is done with 50 and the other half is done with 100.

7) **PPO with the reward 1, 2, and hyperparameter scheduling 2**
   A. **Algorithm:** Vanilla PPO algorithm that is provided as a baseline
   B. **Reward:** 1. Consistent distance penalty term 2. Gradual positive reward for the closer approach
   C. **Hyperparameter:** $num\_envs = 50$ is increased to 100 when the reward exceeds -2.5, which is approximately after 6.4M samples are collected. The number -2.5 is tuned through a trial and error.

8) **PPO with the reward 1, 2, 3**
   A. **Algorithm:** Vanilla PPO algorithm that is provided as a baseline
   B. **Reward:** 1. Consistent distance penalty term 2. Gradual positive reward for the closer approach 3. Consistent penalty term for going farther after reaching the goal
   C. **Hyperparameter:** $num\_envs = 50$

Other than the above methodologies, rewards regarding vehicle heading and vehicle speed are also tried, but they were not successful. Also, the introduced methodologies above are the most successful ones among the various combinations.

5. Results



(a) Performance graph of the 7 methodologies

Methodology	Average Completion Time (s)	Minimum Completion Time (s)	Improvement (%)
1)	7.708300 $\pm$ 0.0265	7.661500	0
2)	7.595250 $\pm$ 0.0272	7.548500	1.48
3)	7.432450 $\pm$ 0.0910	7.282000	4.95
4)	6.771000 $\pm$ 0.0778	6.663000	13.0
5)	6.578000 $\pm$ 0.0824	6.409000	16.3
6)	6.492800 $\pm$ 0.0716	6.372000	16.8
7)	6.224000 $\pm$ 0.0631	6.135500	20.0
8)	**5.793300** $\pm$ 0.0660	**5.685000**	**25.8**

(b) Detailed statistics on the performances

Figure 1: (a) Performance graph of the 7 methodologies that are described in section 4. Average of the completion time is taken over 200 environments. (b) Detailed statistics on the performances. Minimum average completion time is the smallest average completion time among 10 different trials with the same setup. Percentage improvement is based on the minimum completion time of the methodology 1, which is the given default setting.

Figure 2: Learning curves for PPO based methodologies which are 1), 4), 5), 6), 7), 8)

The reason why the methodology 8 has lower rewards is that it uses the third reward component, which is consistent negative reward when the husky tries to move farther from the goal after reaching it.

Videos are available in https://github.com/jungwoohan72/learning_based_control.

6. Discussion

* Research questions are in **bold font.**

According to Figure1(b), vanilla SAC and the SAC with initial position curriculum learning show better performance than vanilla PPO. The results were obtained after collecting only 0.456M and 0.7M samples, respectively. One could conclude that the off-policy learning is indeed sample-efficient than the on-policy learning. However, even though the result suggests that the SAC based algorithms are better than the vanilla PPO algorithm, it is superficial. For my case, the reason why they have shorter completion times is that they perform better only for initial positions below 15m; the SAC based algorithms fail to move fast with correct heading in long range. As the average completion time is calculated over 200 environments with randomly sampled initial position from range of $5 < x, y < 30$, performing well in short distance could lower the overall average value. I conclude that my off-policy setting fails to learn successful long-term policy, and this leads to one interesting research question.

1. What are possible factors that could make the off-policy learning unstable in learning long-term policy in the project?

The initial positions are generated randomly in range $5 < x, y < 30$, and as far as know, they are not uniformly sampled across the range. Then, there could be much more near-the-goal samples than far-away-from-the goal samples in the replay buffer. This would explain why my off-policy algorithm works better in the near-the-goal region.

Also, as on-policy algorithm updates the policy after it finishes one episode, it could evaluate whether the specific undergone trajectory is good or not. This is meaningful for long-term policy, because it is important to evaluate whether or not the agent successfully achieves the 'final' goal. The samples used for off-policy update are relatively uncorrelated compared to the on-policy trajectory samples, and such fact could make the agent difficult to evaluate whether it is doing good or not in term of achieving the long-term final goal.

PPO with the initial position curriculum learning and the hyperparameter scheduling result in dramatic improvement (13% and 20% at most, respectively). Also, the reward function shaping also has contributed to the result. Both auxiliary approaches are to collect good samples in earlier phase of the training so that the policy could converge faster. However, as expected, the training procedures were unstable for both cases. For the initial position curriculum, it works well until the upper bound of the spawning range reaches 27m; the completion rate drops considerably, and the learned policy is not successfully scalable to the range beyond 27m. Also, for the hyperparameter scheduling, reproducibility is the issue. In other words, results vary in large range even with the same setting and code. This is because using the hyperparameter, $num\_envs = 50$, itself results in large variance – while it could update the policy more frequently. Therefore, depending on quality of the policy obtained before changing num_envs to 100, the final performance varies abruptly. The methodology 7) is the most successful attempt among many trials.

Finally, considering the weaknesses of the methodologies explained above, methodology 8) is conducted without the initial position curriculum learning and the hyperparameter scheduling. Most importantly, for the methodology 8), the consistent penalty term is given when the husky tries to escape from 2m range after reaching the goal. I believe this is the key factor of the reward shaping. After trying various reward functions including heading and speed components, I conclude that the algorithm works well if the only distance-based metrics are used. This leads to my next research question.

2. Why does including heading and speed component actually make the training worse?

Intuitively, it seems reasonable to include heading and speed in the reward function; I originally included the heading of the husky to be within range of $\pm 30^{\circ}$ with respect to the vector toward the destination if the distance is above 10m. If the heading deviates from that range, the husky receives proportional negative penalty. Also, for the speed penalty term, if the distance to the goal is above 10m and the current velocity is smaller than the target velocity, the husky receives $-(velocity_{target} - velocity_{current}) \times 0.01$. Here, target velocity was $\frac{distance\ to\ the\ goal}{5}$, as the target completion time was 5 seconds.

I thought adding these two reward term would catalyze the training by making the husky move faster near the goal with right heading; however, it actually made the training worse. The most probable reason is that balance with the distance-based reward terms was not properly set. For example, there could be a situation where the husky receives higher episodic reward by moving fast in wrong direction than moving slowly in right direction. Then the husky would think that the former case is better, and update its policy in an unexpected way. Therefore, I decided to concentrate on the distance-based reward terms, which are the most successful ones.

3. How can we improve performance of initial phase of the training?

I think this question is closely related to exploration and exploitation dilemma. For robust and better policy, many RL algorithms allow more exploration (i.e. ε-greedy) in initial training phase. However, does this count for my project where there exists sample limitation?

What I observed for the training is quality of the final policy highly depends on how well the initial training phase has been done. If the agent succeeds to stably reach the goal before collecting 6M samples, the final average completion times were generally around 6s. In this regard, it is important to obtain high-quality policy in the initial training phase, especially for the case where there exists maximum sample limit. It would be worth trying greedy selection of the action for faster convergence of the policy in the initial phase.

7. Conclusion

In conclusion, it is observed that using the given PPO code with the hyperparameter, $num\_envs = 50$, and all the three types of the reward explained in section 3 results in the best performance. The initial position curriculum learning and the hyperparameter scheduling indeed improve the performance, but they are unstable and not reproducible.