Generalized State-Dependent Exploration (gSDE)

백승언

30 August, 2021

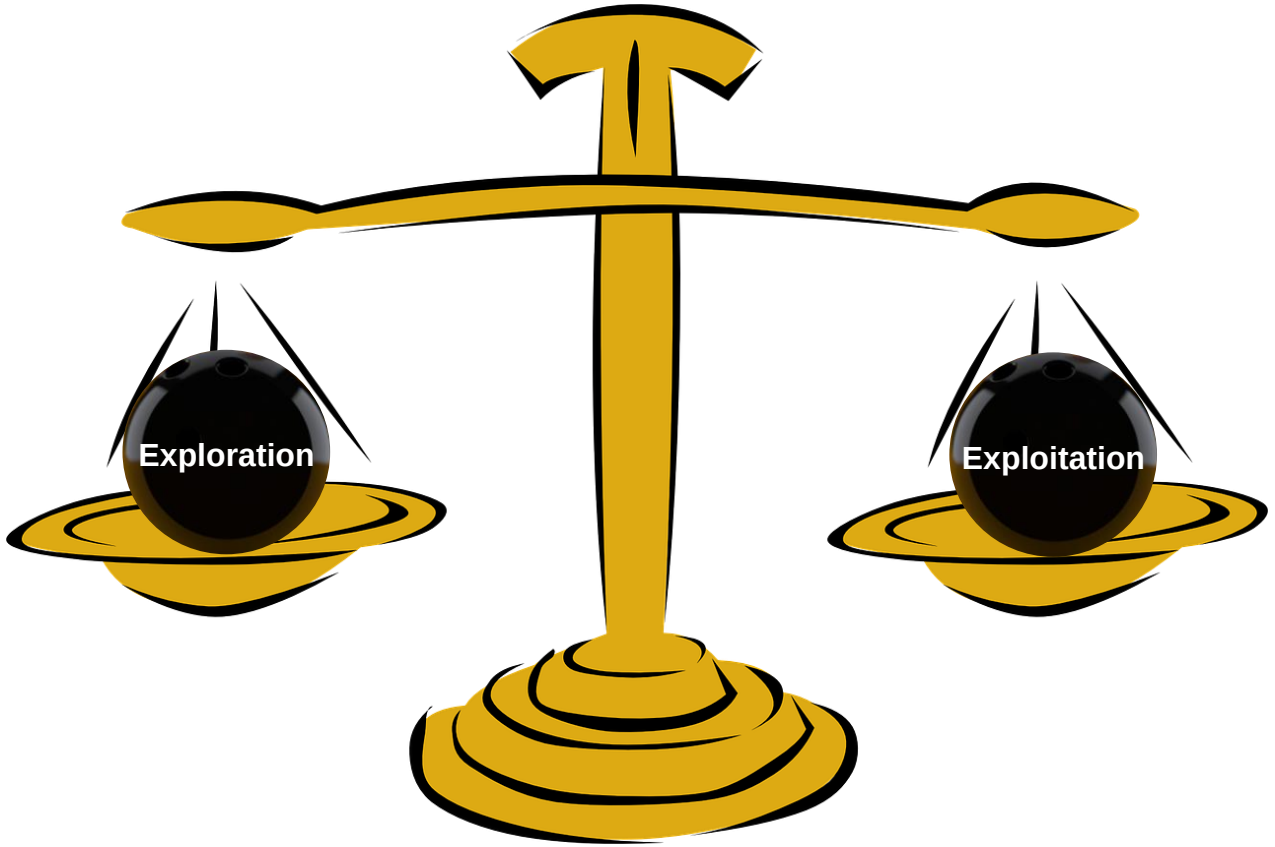
Contents

Introduction

Exploration-Exploitation dilemma

- **Agent learns the optimal policy through exploration and exploitation**
  - The agent need to gather enough information to make the best overall decisions (exploration)
  - The agent need to make the best decision given current information (exploitation)

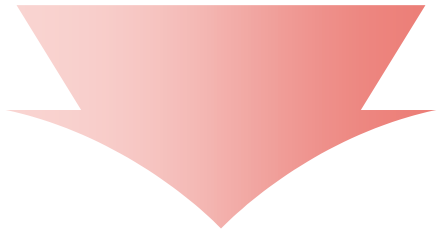
- **What is exploration and exploitation?**
  - Exploration
    - Creating new knowledge
    - New way of thinking
  - Exploitation
    - Reusing the knowledge
    - Best way through experience



Exploration Strategies

- **Naïve approaches in conventional RL**
  - $\epsilon - \text{greedy}$ / decaying $\epsilon - greedy$
  - Count-based exploration
  - Optimism in the face of uncertainty
    - Q-table을 0이 아닌 큰 값으로 초기화 시키는 전략
  - Maximum entropy reinforcement learning

- **Complicated approaches in DRL**
  - Density modeling with entropy regularization
    - Gaussian policy
  - Random noise / stochastic process
    - Gaussian noise
    - OU(Ornstein–Uhlenbeck) noise
  - Curiosity-driven exploration
    - Random network distillation



이러한 탐험 기법들은 로봇, 자율주행 자동차와 같은 실제 시스템에 적용시 예상치 못한 문제를 일으킬 수 있음

Smoothness하고 less variance를 지니는 탐험 전략이 필요!

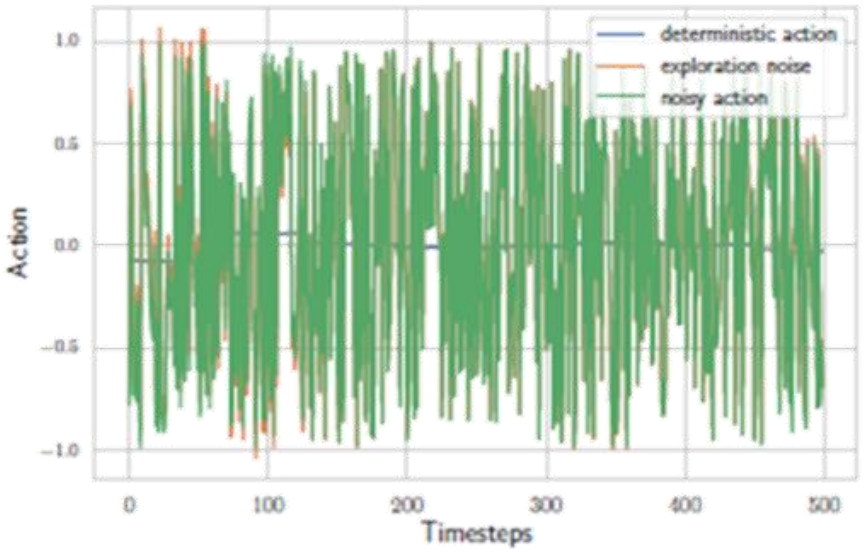
https://www.slideshare.net/DongMinLee32/exploration-strategies-in-reinforcement-learning-179779846
https://www.slideshare.net/YoonhoLee4/parameter-space-noise-for-exploration

Generalized State-Dependent Exploration (gSDE)

https://arxiv.org/abs/2005.05719

Motivations

- **Poor exploration in real robot according to jerky motion pattern with unstructured noise**
  - "When learning robotic skills with deep reinforcement learning(DeepRL), the de factor standard for exploration is to sample a noise vector ϵ_t from a Gaussian distribution independently at each time step t, and then adding it to the policy output"
  - 이러한 구조화 되지 않은(unstructured) 탐험은 실제 로봇으로 실험을 할 때 불안정한 움직임을 야기할 수 있으며, 이는 poor exploration의 결과로 이어질 뿐 아니라, 모터 등의 구동계가 손상을 입을 수 있다고 서술
  - 이러한 한계를 해결하기 위해 여러 연구가 수행되어 왔다고 함
    - Low pass filter, OU noise
  - 저자들은 역시나 이러한 연구의 일환으로 연구되었던, SDE를 DeepRL에 맞게 일반화한 gSDE 연구를 제안
    - SDE: State-Dependent Exploration[1]
    - gSDE: generalized State-Dependent Exploration[2]



Unstructured exploration as typically used in RL

[1]: Rückstieß, Thomas, Martin Felder, and Jürgen Schmidhuber. "State-dependent exploration for policy gradient methods." Joint European Conference on Machine Learning and Knowledge Discovery in Databases. Springer, Berlin, Heidelberg, 2008.
[2]: https://arxiv.org/abs/2005.05719

Backgrounds

- **Exploration in Action or Policy Parameter Space[3]**
  - Continuous action의 경우, exploration은 일반적으로 action space에서 이루어짐: $a_t = \mu(s_t ; \theta_\mu) + \epsilon_t$. 이때, noise vector ϵ_t는 특정 파라미터를 가정한, Gaussian distribution으로부터 sampling됨
  - 이 탐험 전략의 경우, action space가 아닌, (policy) parameter space에서의 noise를 통해 탐험을 종용: $a_t = \mu(s_t ; \theta_\mu + \epsilon)$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$
  - 이는 일반적으로 consistent exploration을 달성할 수 있지만, parameter의 개수가 많아질수록 어려움이 존재

- **State-Dependent Exploration(SDE)**
  - 기존에 이용되던 step-based sampled noise를 episode-based, state-dependent exploration function ϵ을 통해 계산된 noise로 대체한 탐험 전략: $a_t = \mu(s_t ; \theta_\mu) + \epsilon(s_t ; \theta_\epsilon)$, $\theta_\epsilon \sim N(0, \sigma^2)$
  - SDE의 경우 주어진 state에 대해, 동일 episode 내에서 동일한 noise를 계산하는 방식으로 안정적인 탐험을 꾀함(θ_ϵ을 episode 마다 sampling하는 방식 이용)
  - 이는 episode 내에서 smooth하고 less variance한 탐험을 할 수 있는 결과를 만들어 냈다고 함

[3] Rückstiess, T., Sehnke, F., Schaul, T., Wierstra, D., Sun, Y., & Schmidhuber, J. (2010). Exploring parameter space in reinforcement learning. Paladyn, 1(1), 14-24.

gSDE (I)

- **Limitation of original SDE**
  - "The noise does not change during one episode, which is problematic if the episode is long, because the exploration will be limited"
  - "The **variance of policy depends on the state dimension**(it grows with it), which means that the initial σ must be tuned for each problem"
    - In SDE, $\pi_j(a_j \mid s) \sim \mathcal{N}(\mu_j(s),\ \hat{\sigma}_j^2),\ \hat{\sigma}_j = \sqrt{\Sigma_i(\sigma_{ij}\boldsymbol{s_i})^2}$
  - "There is only **a linear dependency between the state and the exploration noise**, which limits the possibilities."
    - In SDE, $a_t = \mu(s_t\,;\theta_\mu) + \epsilon(s_t\,;\theta_\epsilon),\ \ \epsilon = \boldsymbol{\theta_\epsilon s}$
  - The state must be normalized, as the gradient and the noise magnitude depend on the state magnitude

gSDE (II)

- **The Authors proposed two improvements to mitigate the mentioned issues and adapt it to DeepRL algorithms**
  - Sampling the parameters θ_ϵ of the exploration function **every n steps** instead of every episode
    - Long episode, infinite horizon 문제 등에서 한계가 있었던 SDE의 첫 번째 이슈를 해결
  - Using **policy features $z_\mu\left(s\,;\theta_{z_\mu}\right)$** instead of state s as input to the noise function $\epsilon(s\,;\theta_\epsilon) = \theta_\epsilon z_\mu(s)$
    - 이때, policy feature z_μ는 action $\mu(s) = \theta_\mu z_\mu(s\,;\theta_{z_\mu})$ 의 마지막 layer의 output을 의미
    - Variance of policy가 state의 차원과 관계 없이, policy network architecture에만 의존함을 의미 (두 번째 이슈 해결)
    - Exploration noise e와 s가 **non linear한 관계**를 가질 수 있어서 선형 관계만을 가지던 SDE에 비해 표현력 증가 (세 번째 이슈 해결)
    - 학습의 안정화를 위해 SDE에서 수행했던, state normalization등의 추가 엔지니어링이 요구되지 않음 (네 번째 이슈 해결)
  - 저자들은 이러한 개선을 수행한 결과를 generalized State-Dependent Exploration (gSDE)라고 명명

gSDE (III)

- **Pseudo code(Base algorithm: SAC)**
  - Algorithm parameter 초기화 (SAC의 경우, $\theta_\mu, \theta_Q, \alpha$) 및 noise parameter σ를 초기화
    - 알고리즘이 experience memory사용시 replay buffer 초기화 ($\mathcal{D}$)
  - 매 Episode가 시작될 때 마다, exploration noise parameter θ_ϵ을 sampling
    - Pseudo code에 명시되어 있지는 않지만, every n step 마다 θ_ϵ을 추가적으로 sampling
  - 매 step 별로 a_t에 exploration noise $\epsilon(s_t\,;\theta_\epsilon)$을 추가하여 탐험을 종용
  - Update가 될 때마다, reparameterization trick을 통해 σ를 학습시키기 위해 θ_e를 sampling
    - Actor update를 통해 policy $\mu(s)$와 noise variance σ를 업데이트

Algorithm 1 Soft Actor-Critic with gSDE

Initialize parameters θ_μ, θ_Q, σ, α
Initialize replay buffer $\mathcal{D}$
for each iteration **do**
  $\theta_\epsilon \sim \mathcal{N}(0, \sigma^2)$ ▷ Sample noise function parameters
  for each environment step **do**
    $\mathrm{a}_t = \pi(\mathrm{s}_t) = \mu(\mathrm{s}_t; \theta_\mu) + \epsilon(\mathrm{s}_t; \theta_\epsilon)$ ▷ Get the noisy action
    $\mathrm{s}_{t+1} \sim p(\mathrm{s}_{t+1}|\mathrm{s}_t, \mathrm{a}_t)$ ▷ Step in the environment
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathrm{s}_t, \mathrm{a}_t, r(\mathrm{s}_t, \mathrm{a}_t), \mathrm{s}_{t+1})\}$ ▷ Update the replay buffer
  end for
  for each gradient step **do**
    $\theta_\epsilon \sim \mathcal{N}(0, \sigma^2)$ ▷ Sample noise function parameters
    Sample a minibatch from the replay buffer $\mathcal{D}$
    Update the entropy temperature α
    Update parameters using ∇J_Q and ∇J_π ▷ Update actor μ, critic Q and noise variance σ
    Update target networks
  end for
end for

Pseudo code of gSDE

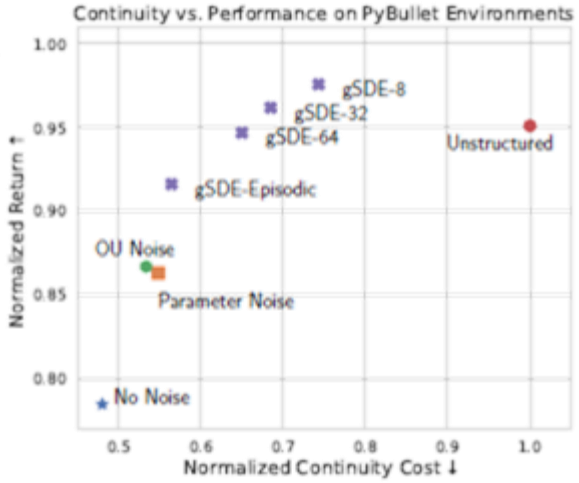
Experiment Results

Experiment results (I)

- **Comparison with different strategies for exploration**
  - 논문에서 제시한 gSDE 기법과, 기존에 사용되던 탐험 전략들에 따른 성능을 Pybullet 환경의 여러 문제에서 비교하는 실험을 수행(Half cheetah 등에서 10번, 1M step 만큼 학습 수행)
    - no exploration noise : 행동에 탐험을 위한 추가적인 noise를 주지 않음
    - Unstructured gaussian noise : 기존에 널리 사용되던, $a_t = \mu(s) + \epsilon, \epsilon \sim \mathcal{N}(0, \sigma^2 \boldsymbol{I})$를 이용
    - OU noise : 비교적 correlated variable을 sampling할 수 있는 stochastic process를 이용
    - parameter space noise : action이 아닌, policy parameter에 noise를 가함 $\pi(s|a\,;\theta),\ \theta \sim \mathcal{N}(\phi, \boldsymbol{\Sigma})$
  - 각 기법들을 비교할 성능 지표로는, Return과 training 중의 continuity cost C_{train}를 선택
    - $C_{tarin} = 100 \times \mathbb{E}_t \left[\frac{a_{t+1} - a_t}{\Delta a_{max}} \right]$ (현재 행동과 다음 행동의 차이가 얼마나 큰지 측정하는 지표)

Algorithm	HalfCheetah		Ant		Hopper		Walker2D	
SAC	Return ↑	C_{train} ↓	Return ↑	C_{train} ↓	Return ↑	C_{train} ↓	Return ↑	C_{train} ↓
w/o noise	2562 +/- 102	**2.6** +/- 0.1	2600 +/- 364	**2.0** +/- 0.2	1661 +/- 270	**1.8** +/- 0.1	2216 +/- 40	**1.8** +/- 0.1
w/ unstructured	**2994** +/- 89	4.8 +/- 0.2	**3394** +/- 64	5.1 +/- 0.1	**2434** +/- 190	3.6 +/- 0.1	2225 +/- 35	3.6 +/- 0.1
w/ OU noise	2692 +/- 68	2.9 +/- 0.1	2849 +/- 267	2.3 +/- 0.0	2200 +/- 53	2.1 +/- 0.1	2089 +/- 25	2.0 +/- 0.0
w/ param noise	2834 +/- 54	2.9 +/- 0.1	3294 +/- 55	**2.1** +/- 0.1	1685 +/- 279	2.2 +/- 0.1	**2294** +/- 40	**1.8** +/- 0.1
w/ gSDE-8	**2850** +/- 73	4.1 +/- 0.2	**3459** +/- 52	3.9 +/- 0.2	**2646** +/- 45	2.4 +/- 0.1	**2341** +/- 45	2.5 +/- 0.1
w/ gSDE-64	**2970** +/- 132	3.5 +/- 0.1	3160 +/- 184	3.5 +/- 0.1	2476 +/- 99	**2.0** +/- 0.1	**2324** +/- 39	2.3 +/- 0.1
w/ gSDE-episodic	2741 +/- 115	3.1 +/- 0.2	3044 +/- 106	2.6 +/- 0.1	2503 +/- 80	**1.8** +/- 0.1	**2267** +/- 34	2.2 +/- 0.1

Return and Continuity cost results for SAC with type of exploration strategies(1M step)

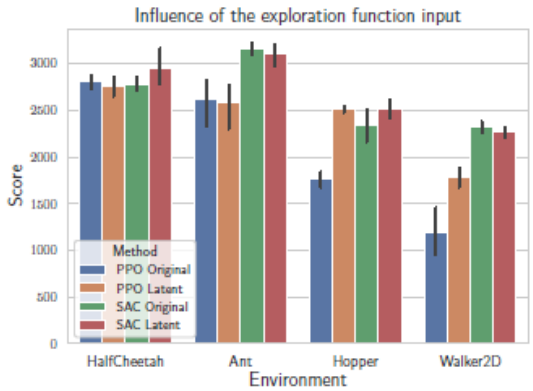


Smoothness vs performance

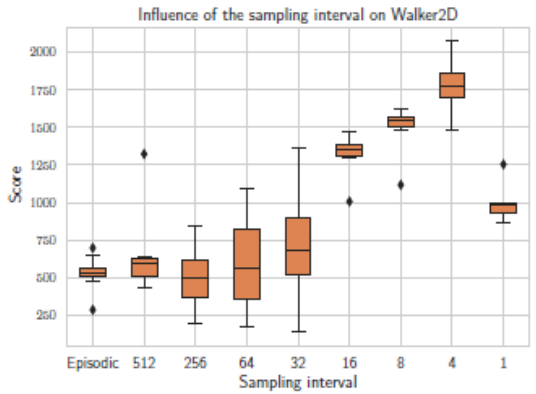
Experiment results (II)

- **Comparison with SDE and gSDE**
  - 앞서 언급한 continuous control 문제에서, original SDE에 비해 gSDE가 더 좋은 성능을 보인다는 것을 확인
  - Sampling interval에 따른 gSDE의 성능을 환경과 알고리즘을 고정한채로 비교 분석 수행
    - SAC의 경우 크게 상관 없었으며, PPO의 경우, sampling frequency를 per 4, 8 step으로 설정할 경우 성능이 가장 좋았음

- **Applying gSDE in real robot system**
  - SAC + gSDE를 직접 적용한 결과가 vanilla SAC에 low-pass filter를 적용한 결과보다 더욱 **안전한**(연속적인) 행동을 고르는 것을 실험으로 확인

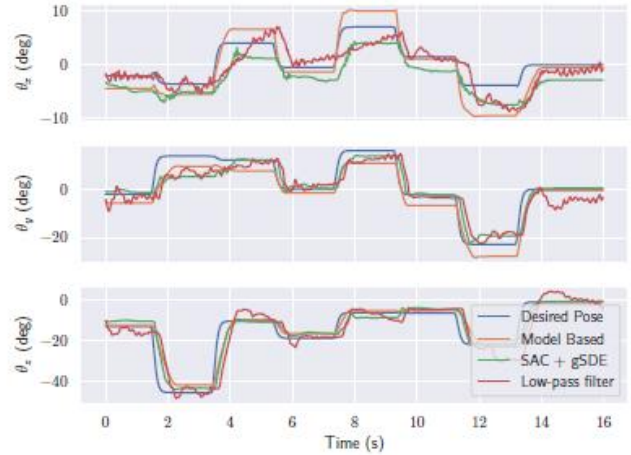


(a) Exploration function input



(b) Sampling interval (PPO on WALKER2D)

(a) Tendon-driven elastic continuum neck in a humanoid robot

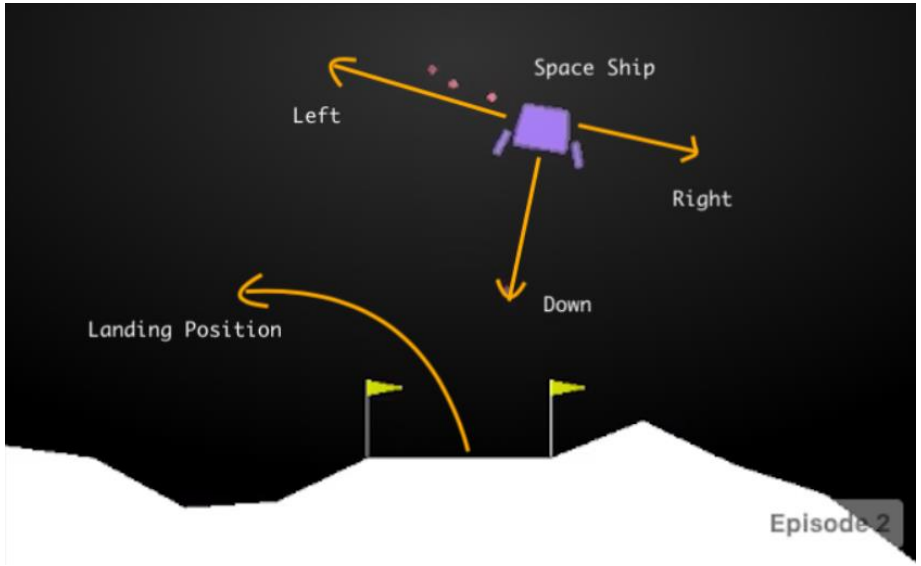


(b) Model-Based controller vs RL controller on the real robot

Implementation for gSDE

Environment setting

- **Environment**
  - LunarLanderContinuous-v2
    - Well-known environment for continuous control in OpenAI's Gym library(Box2D problem)
  - MDP parameters
    - State
      - (8,)
    - Action
      - (2,) Main engine(-1~1), direction engine(-1~1)
    - Reward
      - Arrival with near-zero speed, +100~140
      - Landing with rocket leg, respectively + 10
      - Landing +100, Crash landing +100
      - Fuel consumption penalty, -0.3 per frame, (infinite fuel)



LunarLander environment

- **Algorithm(with Tensorflow 2)**
  - TD3(Twin-Delayed Deep Deterministic policy gradient), PER(Prioritized Experience Replay)
    - TD3 + gaussian noise(decaying std, initial std = 0.2, noise clipping value = 0.5)
    - TD3 + gSDE(episode, every 16 step, every 8 step)

https://gym.openai.com/envs/LunarLanderContinuous-v2/
https://towardsdatascience.com/solving-lunar-lander-openaigym-reinforcement-learning-785675066197

https://arxiv.org/pdf/1802.09477.pdf
https://arxiv.org/abs/1511.05952

Result analysis

- **LunarLanderContinuous-v2**
  - Comparison with Vanilla TD3(**TD3_None**), TD3_gSDE(8, 16, 32, **64, Episode**)
    - Figure 1 shows that TD3_gSDE is better stability than Vanilla TD3
      - gSDE 알고리즘들의 경우, 충돌이 덜 일어나기 때문에 score 그래프가 안정적으로 우 상향
    - Figure 2 shows that TD3_gSDE is better safety than Vanilla TD3 (crash landing less occurred than Vanilla TD3)

Learning environment

Thank you!

Q&A