APPENDIX

A. System Dynamics of Planar Quadrotor

We use the following 6D ordinary differential equation to describe the evolution of the planar quadrotor:

$$\dot{\tilde{s}} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{z} \\ \dot{v}_z \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m} C_D^v v_x + \frac{F_1}{m} \sin\psi + \frac{F_2}{m} \sin\psi \\ v_z \\ -\frac{1}{m}\left(mg + C_D^v v_z\right) + \frac{F_1}{m}\cos\psi + \frac{F_2}{m}\cos\psi \\ \omega \\ -\frac{1}{I_{yy}} C_D^{\psi}\omega + \frac{l}{I_{yy}} F_1 - \frac{l}{I_{yy}} F_2 \end{bmatrix}, \tag{10}$$

where the quadrotor's movement is controlled by two motor thrusts, F_1 and F_2. The quadrotor has mass m, moment of inertia I_{yy}, and half-length l. Furthermore, g denotes the gravity acceleration, C_D^v the translation drag coefficient, and C_D^{ψ} the rotational drag coefficient.

B. Reward Functions

We use Eq. (11) as the reward function for the two examples in Section V, where $\mathbf{G}$ refers to the set of goal states, $\mathbf{C}$ the set of collision states and $\mathbf{I}$ the set of intermediate states which involves neither collision nor goal. Note that here s is the full MDP state, which includes both robotic internal states and sensor readings, for the target RL problem Ω.

Eq. (12) is used in Section VI for the Trap-Goal environment. Here, $\mathbf{Tr}$ represents the states in the trap area.

$$r(s) = \begin{cases} 0 & s \in \mathbf{I} \\ +1000 & s \in \mathbf{G} \\ -400 & s \in \mathbf{C} \end{cases} \quad (11) \qquad r(s) = \begin{cases} 0 & s \in \mathbf{I} \\ +1000 & s \in \mathbf{G} \\ +100 & s \in \mathbf{Tr} \\ -400 & s \in \mathbf{C} \end{cases} \quad (12)$$

C. Data Generation using MPC

1) General MPC Formulation: The goal of MPC for both examples is to generate trajectories from certain initial states to a desired goal region whilst avoiding obstacles. Using the point-mass optimization-based collision avoidance approach, the obstacle avoidance condition is:

$$\text{dist}(p, \mathbb{O}) > d_{min}, \tag{13}$$

where $\text{dist}(p, \mathbb{O}) := \min_t \{||t||_2 : (p+t) \cap \mathbb{O} \neq \emptyset\}$. Then, p is the position of the point-mass, $d_{min} \geq 0$ is a desired safety margin with the size of real robot taken into account, and $\mathbb{O}$ denotes a polyhedral obstacle which is a convex compact set with non-empty relative interior, represented as

$$\mathbb{O} = \{p \in \mathbb{R}^n : Ap \leq b\}, \tag{14}$$

where $A \in \mathbb{R}^{l \times n}$, $b \in \mathbb{R}^l$, l is the number of sides of the obstacle (in our example, all obstacles are shaped as cubes) and n is the dimensionality of positional states. Since (13) is non-differentiable, we reformulate the condition into an equivalent differentiable condition: $\text{dist}(p, \mathbb{O}) > d_{min} \iff \exists \lambda \geq 0 : (Ap - b)^\top \lambda > d_{min},\ ||A^\top \lambda||_2 \leq 1$, where λ is the dual variable. We now formulate the general MPC optimization problem as follows:

$$\begin{aligned} \min_{\tilde{\boldsymbol{s}}, \tilde{\boldsymbol{a}}, \boldsymbol{\lambda}} \ & \sum_{k=0}^{H-1} \left((p_k - p_{ref})^\top \gamma^{k+1} Q (p_k - p_{ref}) + \tilde{a}_k^\top R \tilde{a}_k \right) \\ & + (p_H - p_{ref})^\top \gamma^{H+1} Q (p_H - p_{ref}) \\ \text{s.t.} \ & \tilde{s}_0 = \tilde{s}_{initial} \\ & \tilde{s}_{k+1} = \tilde{s}_k + T_s \tilde{f}(\tilde{s}_k, \tilde{a}_k), \quad \tilde{a}_k \in \Delta, \quad \forall k \in \{0, \dots, H-1\} \\ & \tilde{s}_k \in \tilde{S}, \quad \forall k \in \{0, \dots, H\} \\ & \tilde{s}_H \in \Gamma_3 \\ & \lambda_k^{(m)} \geq 0, \quad \left(A^{(m)} p_k - b^{(m)}\right)^\top \lambda_k^{(m)} > d_{min}, \quad ||A^{(m)\top} \lambda_k^{(m)}||_2 \leq 1, \\ & \forall m \in \{1, \dots, M\}, \quad \forall k \in \{0, \dots, H\} \end{aligned} \tag{15}$$

To match the notations in the main paper, here we employ $\tilde{s}$ and $\tilde{a}$ as the state and action of a low-dimensional model (as in Eq. (10)) used in MPC. Tilde indicates it is an approximate model $\tilde{f}$ in contrast to the full MDP model f of the target RL problem. Specifically, H is the horizon length, $\tilde{\boldsymbol{s}} = [\tilde{s}_0, \dots, \tilde{s}_H]$ is all the states over the horizon, $\tilde{\boldsymbol{a}} = [\tilde{a}_0, \dots, \tilde{a}_{H-1}]$ is the actions over the horizon, $\boldsymbol{\lambda} = [\lambda_0^{(1)}, \dots, \lambda_0^{(M)}, \lambda_1^{(1)}, \dots, \lambda_H^{(M)}]$ is the dual variables associated with obstacles $\mathbb{O}^{(1)} \dots \mathbb{O}^{(M)}$ over the horizon, M is the number of obstacles, $\tilde{s}_k$ is the states at time k, p_k is the positional states at time k, p_{ref} is the reference position, $\tilde{a}_k$ is the control inputs at time step k, γ is the scaling factor for the position error, Q is the weight matrix for the position error, R is the weight matrix for the control effort, $\tilde{s}_{initial}$ is the initial state, T_s is the sampling period, $\tilde{f}$ is the continuous-time system dynamics, Δ is the collection of possible control inputs, $\tilde{S}$ is the state constraints, Γ_3 is the collection of goal states, and $A^{(m)} \in \mathbb{R}^{l \times n}$ and $b^{(m)} \in \mathbb{R}^l$ define the obstacle $\mathbb{O}^{(m)}$ based on (14).

2) Car Example MPC Formulation: In the car scenario, the continuous-time system dynamics $\tilde{f}$ is:

$$\dot{\tilde{s}} = \tilde{f}(\tilde{s}, \tilde{a}) = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos(\theta) \\ v\sin(\theta) \\ \omega \end{bmatrix} \tag{16}$$

where $\tilde{s} = [x, y, \theta]^\top$ and $\tilde{a} = [v, \omega]^\top$. In the optimization problem we have:

$H = 80; \gamma = 1.1; T_s = 0.05; d_{min} = 0.277; M = 6$
$p = [x, y]^\top; p_{ref} = [3.5, 3.5]^\top$
$Q = \text{diag}([10, 10]^\top); R = \text{diag}([1, 1]^\top)$
$\Delta = \{[v, \omega]^\top : -2 \leq v \leq 2,\ -2 \leq \omega \leq 2\}$
$\tilde{S} = \{[x, y, \theta]^\top : |x| \leq 4.723, |y| \leq 4.723, -\infty \leq \theta \leq \infty\}$
$\Gamma_3 = \{[x, y, \theta]^\top : ||[x, y]^\top - [3.5, 3.5]^\top||_2 \leq 1, \cos(\theta - 0.75) \leq 0.3\}$

The matrices for specifying obstacles $\mathbb{O}^{(1)} \dots \mathbb{O}^{(6)}$ are:

$$A^{(m)} = \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & -1 \\ 0 & 1 \end{bmatrix}, \quad \forall m \in \{1, \dots, M\} \tag{17}$$

$$\begin{aligned} b^{(1)} &= \begin{bmatrix} 0.35 \\ 0.35 \\ 0.35 \\ 0.35 \end{bmatrix}, b^{(2)} = \begin{bmatrix} -2.65 \\ 3.35 \\ 1.35 \\ -0.65 \end{bmatrix}, b^{(3)} = \begin{bmatrix} -1.65 \\ 2.35 \\ -0.65 \\ 1.35 \end{bmatrix}, \\ b^{(4)} &= \begin{bmatrix} 2.35 \\ -1.65 \\ 2.35 \\ -1.65 \end{bmatrix}, b^{(5)} = \begin{bmatrix} 0.35 \\ 0.35 \\ -3.65 \\ 4.35 \end{bmatrix}, b^{(6)} = \begin{bmatrix} 3.35 \\ -2.65 \\ -1.65 \\ 2.35 \end{bmatrix}. \end{aligned} \tag{18}$$

3) Quadrotor Example MPC Formulation: In the quadrotor scenario, the continuous-time system dynamics $\tilde{f}$ is denoted as Eq. (10). In the optimization problem we have:

$H = 140; \gamma = 1.1; T_s = 0.05; d_{min} = 0.1; M = 4$
$p = [x, z]^\top; p_{ref} = [4, 9]^\top$
$Q = \text{diag}([10000, 10000]^\top); R = \text{diag}([1, 1]^\top)$
$\Delta = \{[M_1, M_2]^\top : 5 \leq M_1 \leq 11,\ 5 \leq M_2 \leq 11\}$
$\tilde{S} = \{[x, v_x, z, v_z, \phi, \omega]^\top : |x| \leq 4.75, 0.25 \leq z \leq 9.75, |\phi| \leq \pi/2\}$
$\Gamma_3 = \{[x, z, \phi] : 3 \leq x \leq 5, 8 \leq z \leq 10, -\pi/3 \leq \phi \leq \pi/3\}$

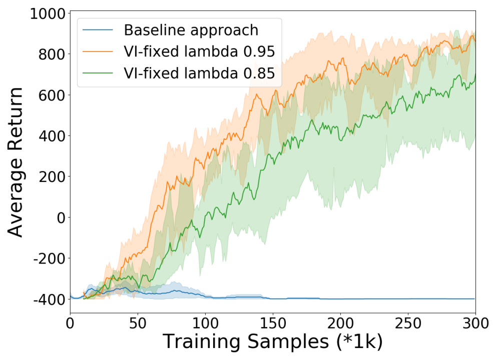
In the quadrotor scenario, $A^{(m)}$ is the same as the car scenario in (17). However, $b^{(1)}, \dots, b^{(4)}$ are:

$$b^{(1)} = \begin{bmatrix} 3.25 \\ -0.75 \\ -3.75 \\ 6.25 \end{bmatrix}, \quad b^{(2)} = \begin{bmatrix} 1.25 \\ 1.25 \\ -7.5 \\ 9.5 \end{bmatrix},$$
$$b^{(3)} = \begin{bmatrix} -1.75 \\ 5.75 \\ -3.75 \\ 6.25 \end{bmatrix}, \quad b^{(4)} = \begin{bmatrix} 0.75 \\ 0.75 \\ 0.5 \\ 2.5 \end{bmatrix}. \tag{19}$$

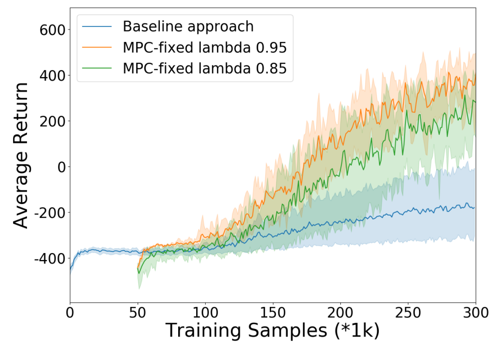
D. More results using TD(λ) return

As discussed in Sec. IV-C, the TD(λ) estimation of G_t^λ can include an on-policy value function $V^\pi(s_t)$ if $\lambda \in [0, 1)$. However, we demonstrate that our model-based baseline function $V^{\mathrm{M}}(s_t)$ can not only act as a choice of the baseline function $b(s_t)$, but also a special, model-based value function for more general TD(λ) return to replace $V^\pi(s_t)$. Without the loss of generality, we test the cases where $\lambda = 0.95, 0.85$.

Fig. 4 illustrates the learning performance on the car and quadrotor examples respectively using our MBB approach[5] with TD(λ) return, where $V^\pi(\cdot)$ is replaced by $V^{\mathrm{M}}(\cdot)$. We observe similar data efficiency improvements as using Monte-Carlo return in Sec. V, which indicates the benefits of MBB approach under general TD(λ) return despite the potential bias. We also note that as λ deceases, the averaged policy performance decreases correspondingly in certain extent. This is potentially a possible negative effect of using $V^{\mathrm{M}}(s_t)$ for TD(λ) estimation. Since $V^{\mathrm{M}}(s_t)$ is calculated and approximated from a lower-dimensional problem, it introduces more bias than V^π in G^λ estimation. As λ deceases, such bias becomes more significant and can be more harmful to the learning process as the value function occupies larger proportion in the TD(λ) return.



(a) Car example



(b) Quadrotor example

Fig. 4: Performance comparison between MBB and the baseline approach, where MBB approach uses TD(λ) return ($\lambda = 0.95, 0.85$) for policy gradient calculation.

E. More Discussions

*1) **Correctness of Policy Gradient Estimation**:* Following the procedure in Section VI, we now further investigate the correctness of gradient estimation in the Trap-Goal environment. We take policies at certain iterations in the earlier learning stage and employ these policies to collect sample transitions from the Trap-Goal environment. Particularly, we denote π_{G}^{20}, π_{G}^{40},π_{G}^{60} as polices belong to the goal-reaching trials at iteration 20, 40, 60; π_{Tr}^{20}, π_{Tr}^{40},π_{Tr}^{60} as polices belong to the trap-reaching trials at iteration 20, 40, 60.

We first take Fig. 5a, 5b and 5c as a group to analyze the results since they are all using trap-reaching polices. In the three sub-figures, MBB approach always provide completely distinct gradient estimation from the baseline approach regardless of sample size or ground truth type (comparing blue, solid line to red, solid line, or blue, dashed line to red, dashed line). This suggests that MBB is capable of distinguishing the wrong policy optimization direction for the Trap-Goal task.

On the other hand, Fig. 5d, 5e and 5f show the cosine similarity for different sample sizes with the samples collected from goal-reaching policies. In this case, MBB and the baseline approach have consistent tendencies on gradient estimation. However, even in this case, our MBB approach still provides more accurate gradient estimation than the baseline approach (blue solid and dashed lines are mostly above red solid and dashed lines respectively), especially at relatively lower sample sizes.

[5]Specifically, we use VI-fixed as a representative variant of the MBB approach for the car and MPC-fixed for the quadrotor.

*2) **Advantage Estimation**:* We also investigate the advantage estimation to empirically show the efficacy of MBB algorithm. The advantage function $A^\pi(s, a)$ measures how much better an action a is than others on average, and has been extensively used in policy gradient algorithms to drive the policy update [22]. In this paper, we assume the advantage is defined by $A^\pi(s, a) = G_t^\lambda - b(s_t)$ as shown in Eq. (1) where $b(s_t)$ can be the on-policy value function $V^\pi(s_t)$ or our model-based baseline function $V^{\mathrm{M}}(s_t)$.

Fig. 6a shows the advantage estimation comparison between MBB and the baseline approach. The total number of training iterations are 150 and we take an average of advantage estimation every 10 iterations. It is clearly observed that MBB approach maintains a relatively larger range of advantage estimation over the entire learning process. In contrast, the advantage estimation of the baseline approach gradually shrink to a very narrow range around zero. This may indicate that MBB involves more exploration thus more likely to escape local optima. On the other hand, as we already observed in the Fig. 1c, the baseline approach can often drive the robot get stuck in an undesired area for a long time due to the negligible advantage estimation.

*3) **Variance of Policy Gradient Estimation**:* We also conduct empirical analysis on the variance of gradient estimation. More specifically, at each training iteration, we

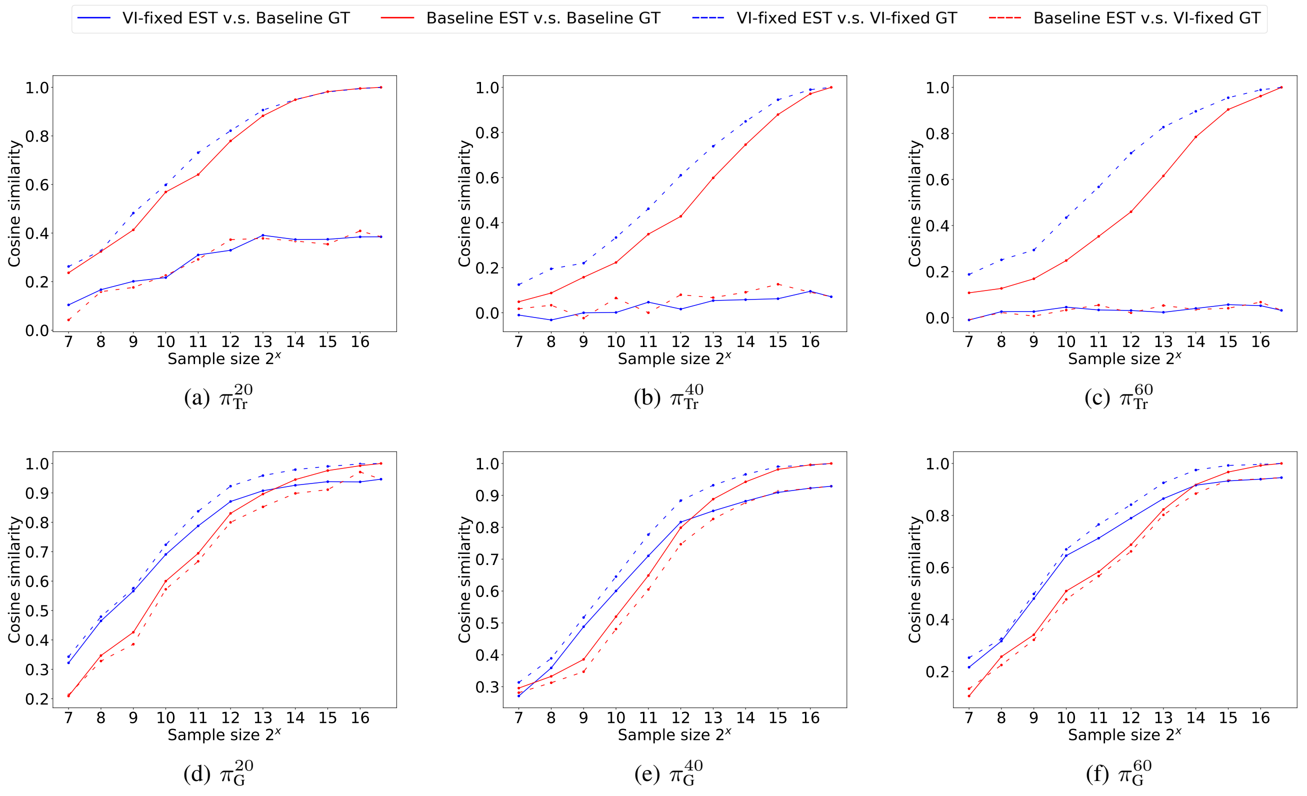
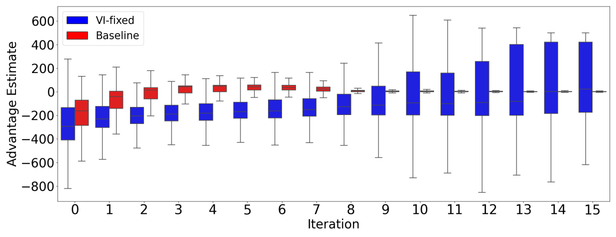
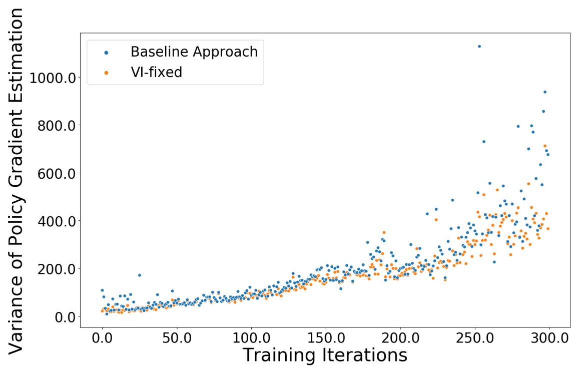


Fig. 5: The comparison on the correctness of policy gradient estimation between MBB and the baseline approach under varied sampling policies. We take (a), (b), (c) as a group, and (d), (e), (f) as another group to analyze. In each sub-figure, blue lines (solid or dashed) represent MBB-based gradient estimation with regard to different types of ground truths while red lines (solid or dashed) represent that of the baseline approach. A good comparison group would be "blue solid v.s. red solid" or "blue dashed v.s. red dashed".



(a) Advantage Estimation



(b) Variance of Policy Gradient Estimation

Fig. 6: **(a)** Advantage estimation versus training iterations using MBB and the baseline approach. Larger colored box refers to wider range of advantage estimation. **(b)** Policy gradient variance computed via MBB and the baseline approach. Each dot corresponds to the magnitude of variance at certain iteration.

collect K[6] samples with each of them denoted as $D_i = (s_i, a_i, r_i, s_{i+1}); i \in \{0, 1, 2, ..., K\}$ using MBB and the baseline approach respectively and calculate K policy gradients $\nabla_\theta J(\theta|D_i)$ based on each sample. Then we compute the largest singular value of co-variance matrix of each set of gradients $\{\nabla_\theta J(\theta|D_i) \mid i \in \{0, 1, ..., K-1\}\}$.

Fig. 6b shows the gradient variance magnitude versus training iterations. MBB (orange dots) and the baseline (blue dots) approach share comparable variance of gradient estimation over the entire training process. This fully shows the benefits of our MBB approach since our model-based baseline are facilitating the learning performance by providing necessary exploration and better gradient direction whilst not introducing too much variance.

F. Results on Q-learning based Algorithm

In contrast to direct policy gradient based methods (like PPO), Q-learning based methods are another important type of model-free RL algorithms. Deep Deterministic Policy Gradient (DDPG) algorithm is a well-known example. Unlike PPO which directly executes policy gradient optimization, DDPG first learns an on-policy Q function $Q^\pi(s, a)$ and then derives a policy based on $Q^\pi(s, a)$ [39].

We test our MBB approach on a refined DDPG algorithm (known as TD3 [40]) to demonstrate its compatibility and efficacy on different types of model-free RL algorithms.

*1) **Important Procedure of DDPG Algorithm:*** In DDPG, the Q function $Q^\pi(s, a)$ is often represented by a nonlinear function approximator (e.g. neural network) with parameters ϕ, denoted as $Q^\pi_\phi(s, a)$. Here we simplify $Q^\pi_\phi(s, a)$ to $Q_\phi(s, a)$ for conciseness. In addition, a Q target function $Q_{\phi_{\text{targ}}}(s, a)$ is introduced in order to alleviate the optimization instability [39]. At each iteration, the Q target function $Q_{\phi_{\text{targ}}}(s, a)$ is also responsible for providing the Q function approximation target $y(r, s', d)$:

$$y(r, s', d) = r(s) + \gamma(1-d) Q_{\phi_{\text{targ}}}\left(s', \mu_{\theta_{\text{targ}}}(s')\right) \tag{20}$$

where $\mu_{\theta_{\text{targ}}}$ is the parameterized target policy. Then Q function $Q_\phi(s, a)$ is updated towards $y(r, s', d)$ by one-step gradient descent using Eq. (21) where $\mathcal{B}$ is a batch of transition data sampled from original RL problem Ω:

$$\phi \leftarrow \phi - \alpha \nabla_\phi \frac{1}{|\mathcal{B}|} \sum_{(s,a,r,s',d) \in \mathcal{B}} \left(Q_\phi(s, a) - y(r, s', d)\right)^2 \tag{21}$$

The policy μ_θ is updated subsequently via one-step gradient ascent towards $Q_\phi(s, a)$ via:

$$\theta \leftarrow \theta + \beta \nabla_\theta \frac{1}{|\mathcal{B}|} \sum_{s \in \mathcal{B}} Q_\phi\left(s, \mu_\theta(s)\right) \tag{22}$$

α and β are step sizes. The Q target $Q_{\phi_{\text{targ}}}(s, a)$ and policy target $\mu_{\theta_{\text{targ}}}$ then need to be updated in certain way such as polyak averaging:

$$\begin{aligned} \phi_{\text{targ}} &\leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi \\ \theta_{\text{targ}} &\leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta \end{aligned} \tag{23}$$

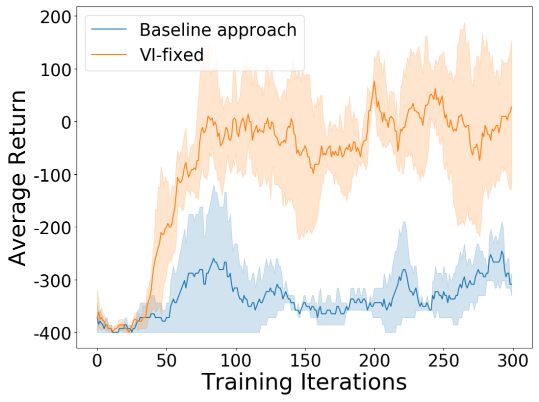
[6]We use $K = 1024$ at each training iteration

*2) **MBB-based DDPG Algorithm:*** We take "VI-fixed" as the representative variant of our MBB algorithm and integrate it into DDPG via the following two steps:

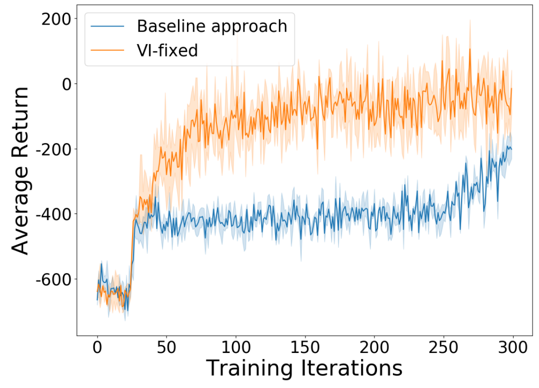
1) Our model-based baseline function $V^{\text{M}}(\cdot)$ can be considered as a special value function which contains the model information from a simplified, low-dimensional problem. Therefore, we can use $V^{\text{M}}(\cdot)$ and one-step simulation to replace $Q_{\phi_{\text{targ}}}(s, a)$ in Eq. (20), where the one-step simulation is shown as Eq. (24) assuming a deterministic full MDP model.

$$Q_{\phi_{\text{targ}}}(s, a) = r(s) + \gamma V^{\text{M}}(s') \tag{24}$$

2) The variant "VI-fixed" does not enable the subsequent update of $V^{\text{M}}(\cdot)$. Correspondingly, we can disable the update of $Q_{\phi_{\text{targ}}}(s, a)$ in Eq. (23).



(a) Car Example



(b) Quadrotor Example

Fig. 7: Overall learning performance comparison using MBB-based and the original TD3 algorithms on two robotic examples. We take the averaged training return versus training iterations as performance measurement. Results are summarized across 5 different trials. The curves show the mean return and the shaded area represents the standard deviation of 5 trials.

Particularly in TD3 algorithm, there are two Q target functions: $Q_{\phi_{\text{targ},1}}(s, a)$ and $Q_{\phi_{\text{targ},2}}(s, a)$, and $y(r, s', d)$ is computed by:

$$y(r, s', d) = r(s) + \gamma(1-d) \min_{i=1,2} Q_{\phi_{\text{targ}},i}\left(s', a'(s')\right) \tag{25}$$

Therefore, with MBB, we simply use $V^{\mathrm{M}}(\cdot)$ and Eq. (24) to replace both Q target functions $Q_{\phi_{\text{targ},1}}(s,a)$ and $Q_{\phi_{\text{targ},2}}(s,a)$ and disable the update of them.

Fig. 7 illustrates how the learning progresses with MBB-based and the original TD3 algorithm (noted as the baseline approach) on two robotic problems in Section V. Note that the baseline approach uses random initialized Q target function and regular Q target update following Eq. (23). MBB approach refers to a new TD3 algorithm that involves the refinement described in this section. As evident from Fig. 7, MBB approach continues to improve the policy with tolerable variance, and outperforms the baseline approach at around 50 iterations on both examples. For the quadrotor case, even though the baseline approach eventually improves at around 280 iteration, it largely lags behind MBB approach and shows poor data efficiency.

G. *Code base and hyper-parameters of neural networks*

For the PPO algorithm, we follow the implementations from OpenAI Baselines [41]. For the TD3 algorithm, we follow the original implementations from the author. Table. II summarizes the neural network structure as well as the main hyper-parameters of PPO and TD3 algorithm.

	PPO	TD3
Type of layers	FC	FC
Structure of layers	(64,64)	(256,256)
Total timesteps	300K	300K
Policy clipping	0.2	×
Learning rate	$3e^{-4}$	$1e^{-3}$
SGD Optimizer	Adam	Adam
Discount γ	0.998	0.99
λ for TD(λ) return	[1, 0.95, 0.85]	×
Gradient clipping	0.5	×
Policy noise	×	0.2
Policy update frequency	×	2
Target network update rate	×	0.005

TABLE II: Main hyper-parameters and neural network structures for PPO and TD3 algorithm.