2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

TTR-Based Reward for Reinforcement Learning with Implicit Model Priors

Xubo Lyu[1] and Mo Chen[1]

***Abstract*— Model-free reinforcement learning (RL) is a powerful approach for learning control policies directly from high-dimensional state and observation. However, it tends to be data-inefficient, which is especially costly in robotic learning tasks. On the other hand, optimal control does not require data if the system model is known, but cannot scale to models with high-dimensional states and observations. To exploit benefits of both model-free RL and optimal control, we propose time-to-reach-based (TTR-based) reward shaping, an optimal control-inspired technique to alleviate data inefficiency while retaining advantages of model-free RL. This is achieved by summarizing key system model information using a TTR function to greatly speed up the RL process, as shown in our simulation results. The TTR function is defined as the minimum time required to move from any state to the goal under assumed system dynamics constraints. Since the TTR function is computationally intractable for systems with high-dimensional states, we compute it for approximate, lower-dimensional system models that still captures key dynamic behaviors. Our approach can be flexibly and easily incorporated into any model-free RL algorithm without altering the original algorithm structure, and is compatible with any other techniques that may facilitate the RL process. We evaluate our approach on two representative robotic learning tasks and three well-known model-free RL algorithms, and show significant improvements in data efficiency and performance.**

I. INTRODUCTION

Model-free RL has been successful in many fields such as games and robotics [1], [2], and allows control policies to be learned directly from high-dimensional inputs by mapping observations to actions. However, model-free RL often requires an impractically large number of trials to learn desired behaviours which is costly and unrealistic especially in the context of robotics [3]. To address such sample efficiency challenge, prior model-free RL methods incorporate techniques such as curiosity-based exploration [4], curriculum learning [5] and hierarchical RL [6], while model-based RL tries to incorporate environment model information into policy learning [7].

Model-based RL uses an internal model (given or learned) that approximates the full system dynamics [8]. A control policy is learned based on this model. This significantly reduces the number of trials in learning and leads to fast convergence [9]. However, model-based methods are heavily dependent on the accuracy of model itself, thus the learning performance can be easily affected by the model bias. This is challenging especially when one aims to map sensor inputs directly to control actions, since the evolution of sensor inputs over time can be very difficult to model.

Optimal control can be considered a classical,analytical method in contrast to more recent learning-based methods, and has been substantially applied to many applications such as mobile robotics and aerospace [10], [11] In general, optimal control does not require any data to generate optimal solution if the system model is known, but cannot scale to models with high-dimensional state space.

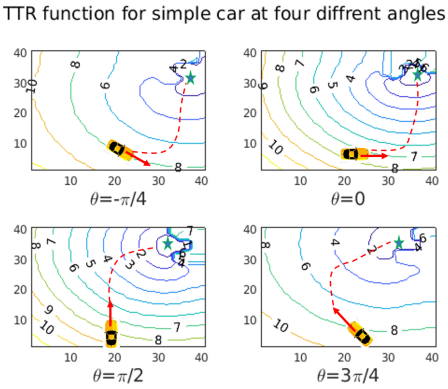

Fig. 1: TTR functions at different heading angles for a simple car model. The TTR function describes the minimum arrival time under assumed system dynamics and is effectively used for reward shaping in robotic RL tasks.

In this paper, we propose Time-To-Reach (TTR) reward shaping, an approach that integrates optimal control into model-free RL. This is accomplished by incorporating into the RL algorithm a TTR-based reward function, which is obtained by solving a Hamilton-Jacobi (HJ) partial differential equation (PDE), a technique that originated in optimal control. A TTR function maps a robot's internal state to the minimum arrival time to the goal, assuming a model of the robot's dynamics. In the context of reward function in RL, intuitively a smaller TTR value indicates a desirable state for many goal-oriented robotic problems.

To accommodate the computational intractability of computing the TTR function for a high-dimensional system such as the one used in the RL problem, an approximate, low-dimensional system model that still captures key dynamic behaviors is selected for the TTR function computation. As we will demonstrate, such approximate system model is sufficient for improving data efficiency of policy learning. Therefore, our method avoids the shortcomings of both model-free RL and optimal control. Unlike model-based RL, our method does not try to learn and use a full model explicitly. Instead, we maintain a looser connection between a known model and the policy improvement process in the form of a TTR reward function. This allows the policy improvement process to take advantage of model information while remaining robust to model bias.

Our approach can be modularly incorporated into any model-free RL algorithm. In particular, by effectively infusing system dynamics in an implicit and compatible manner with RL, we retain the ability to learn policies that map sensor inputs directly to actions. Our approach represents

This work was funded in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grants Program.

[1]School of Computing Science, Simon Fraser University, BC, CA V5A1S6. xlv@sfu.ca, mochen@cs.sfu.ca

a bridge between traditional analytical optimal control approaches and the modern data-driven RL, and inherits benefits of both. We evaluate our approach on two common mobile robotic tasks and obtain significant improvements in learning performance and efficiency. We choose Proximal Policy Optimization (PPO) [3], Trust Region Policy Optimization (TRPO) [12] and Deep Deterministic Policy Gradient (DDPG) [2] as three representative model-free algorithms to illustrate the modularity and compatibility of our approach.

II. PRELIMINARIES

In this section, we introduce key background concepts of this work. Firstly, the Markov Decision Process is described as the fundamental mathematical framework for modeling RL problem. Secondly, model-free RL optimization techniques that are closely related to our work will be presented. Thirdly, the key concepts of approximate system model and the mathematical formulation of TTR function are given.

A. Markov Decision Process

A Markov Decision Process (MDP) is a discrete time stochastic control process. It serves as a framework for modeling decision making in situations where outcomes are partly random and partly under the control of decision maker. Consider a MDP defined by a 4-tuple $M = (S, A, f(\cdot,\cdot,\cdot), r(\cdot,\cdot))$, where S is a finite set of states, and A is a finite set of actions. $f(s,a,s') = \Pr(s_{t+1} = s' | s_t = s, a_t = a)$ is the transition probability that action a in state s at time t will lead to state s' at time $t+1$. The reward function $r(s,a)$ represents the immediate reward received if action a is chosen at state s. In this paper, we employ a slight abuse of notation and write

$$s_{t+1} \sim f(s_t, a_t) \tag{1}$$

to denote that s_{t+1} is drawn from the distribution $\Pr(s_{t+1}|s_t = s, a_t = a)$. This is done to match the notation of the approximate system dynamics presented in Eq. (3).

Given an MDP, one aims to find a "policy" denoted $\pi(\cdot)$ that specifies the action $a = \pi(s)$ that is chosen at state s, such that the expected sum of discounted rewards

$$R_\pi(s_0) = \sum_{t=0}^{T} \gamma^t r(s_t, \pi(s_t)) \tag{2}$$

is maximized over a finite horizon. Here, $\gamma \in [0,1]$ denotes the discount factor and is usually close to 1.

B. Model-free Reinforcement Learning

Model-free RL uses algorithms that do not require explicit knowledge of the transition probability distribution associated with the MDP to optimize RL objective. An obvious advantage of such algorithm is "model-independency" since the MDP model is often inaccessible in the problems with high-dimensional state space.

Policy-based and value-based methods are two main approaches for training agents with model-free RL. Policy-based methods primarily learn a policy by representing it explicitly as $\pi_\theta(a|s)$ and optimizing the parameters θ either directly by gradient ascent on the performance objective $J(\pi_\theta)$ or indirectly by maximizing local approximations of $J(\pi_\theta)$. Policy-based methods sometimes involve "on-policy" updates which means they update policy only using data collected by the most recent version of the policy.

Value-based methods, on the other hand, primarily learn an action-value approximator $Q_\theta(s,a)$. The optimization is sometimes performed in a "off-policy" manner which means it can learn from any trajectory sampled from the same environment. The corresponding policy is obtained via the connection between Q and π: $\pi(s) = \arg\max_a Q_\theta(s,a)$.

Among the three model-free RL algorithms in this work, PPO and TRPO fall into the category of policy-based methods while DDPG belongs to value-based methods.

C. Approximate System Model and Time-to-Reach Function

Consider the following dynamical system in $\mathbb{R}^n$

$$\dot{\tilde{s}}(\tau) = \tilde{f}(\tilde{s}(\tau), \tilde{a}(\tau)) \tag{3}$$

Note that $\tilde{f}(\cdot)$ is used to distinguish this model from $f(\cdot)$ in Eq. (1). Here, $\tilde{s}(\cdot)$ and $\tilde{a}(\cdot)$ are the state and action of an approximate system model. The TTR problem involves finding the minimum time it takes to reach a goal from any initial state $\tilde{s}$, subject to the system dynamics in Eq. (3). We assume that $\tilde{f}(\cdot)$ is Lipschitz continuous. Under these assumptions, the dynamical system has a unique solution. The common approach for tackling TTR problems is to solve a Hamilton-Jacobi (HJ) partial differential equation (PDE) corresponding to system dynamics and this approach is widely applicable to both continuous and hybrid systems [13]–[15]. Mathematically, the time it takes to reach a goal $\Gamma \in \mathbb{R}^n$ using a control policy $\tilde{a}(\cdot)$ is

$$T_{\tilde{s}}[\tilde{a}] = \min\{\tau | \tilde{s}(\tau) \in \Gamma\} \tag{4}$$

and the TTR function is defined as follows:

$$\phi(\tilde{s}) = \min_{\tilde{a} \in \tilde{\mathcal{A}}} T_{\tilde{s}}[\tilde{a}] \tag{5}$$

$\tilde{\mathcal{A}}$ is a set of admissible controls in approximate system. Through dynamic programming, we can obtain ϕ by solving the following stationary HJ PDE:

$$\max_{\tilde{a} \in \tilde{\mathcal{A}}} \{-\nabla\phi(\tilde{s})^\top \tilde{f}(\tilde{s}, \tilde{a}) - 1\} = 0 \tag{6}$$

$$\phi(\tilde{s}) = 0 \ \forall \tilde{s} \in \Gamma \tag{7}$$

Detailed derivations and discussions are presented in [16], [17]. Normally the computational cost of solving the TTR problem is too expensive for systems with higher than five dimensional state. However, model simplification and system decomposition techniques partially alleviate the computational burden in a variety of problem setups [18], [19]. Well-studied level set based numerical techniques [14], [15], [18], [19] have been developed to solve Eq. (6).

III. APPROACH

Model-free RL algorithms have the benefit of being able to learn control policies directly from high-dimensional state and observation; however, the lack of data efficiency is a well-known challenge. Integrating a fully MDP model into RL seems promising but can sometimes be difficult due to model bias. In this work [1], we address this issue by

[1]Code is available at: https://github.com/SFU-MARS/reward_shaping_ttr/tree/dev; videos are available at https://sites.google.com/view/ttrpapersupp

implicitly utilizing a simplified system model to provide a useful "model-informed" reward in an important subspace of the full MDP state. This way we produce policies that are as flexible as those obtained from model-free RL algorithms, and accelerate learning without altering the model-free pattern.

In this section, we explain the concrete steps (shown in Fig. 2) of applying our method. The system under consideration may be represented by an MDP given by $f(\cdot)$, as explained in Section II-A; this MDP is in general unknown. Choosing an approximate system model $\tilde{f}(\cdot)$ that captures key dynamic behavior is the first step; this step is explained in Section III-A. Using this approximate system model, we compute the TTR function $\phi(\cdot)$, and then apply a simple transformation to it to obtain the reward function $r(\cdot)$ that is used in RL; this is fully discussed in Section III-B. Finally, any model-free RL algorithm may be used to obtain a policy that maximizes the expected return in Eq. (2).

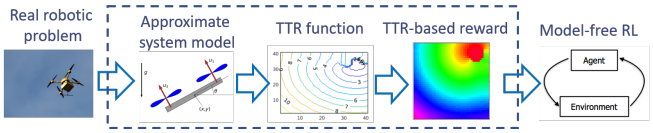

Fig. 2: Sequential steps of TTR-based reward shaping

A. Model Selection

"Model selection"[2] here refers to the fact that we need to pick an (approximate) system model for the robotic task in order to compute the corresponding TTR function. This model should be relatively low-dimensional so that the TTR computation is tractable but still retain key behaviors in the dynamics of the system.

Before the detailed description of model selection, it is necessary to clarify some terminology used in this paper. First, we use the phrase "full MDP model" to refer to $f(\cdot)$, which drives the real state transitions in the RL problem. The full MDP model is often inaccessible since it captures the high-dimensional state inputs including both sensor data and robot internal state. Second, we will use the phrase "approximate system model" to refer to $\tilde{f}(\cdot)$. The tilde indicates that $\tilde{f}$ does not necessarily accurately reflect the real state transitions of the problem we are solving. In fact, the approximate system model should be low-dimensional to simplify the TTR computation while still capturing key robot physical dynamics.

The connection between the full MDP model and the approximate system model is formalized as follows. We assume that the approximate system state is a subset of the full MDP state. Thus, the relation between the full MDP state and the approximate model state is:

$$s = (\tilde{s}, \hat{s}) \tag{8}$$

Here the full state s refers to the entire high-dimensional state in the full MDP model $f(\cdot)$, and $\tilde{s}$ refers to the state of the approximate system model $\tilde{f}(\cdot)$ which evolves according to Eq. (3). For clarity, we also define $\hat{s}$, which are state components in the full MDP model that are not part of $\tilde{s}$.

[2]Note that "model selection" here has a different meaning than that in machine learning.

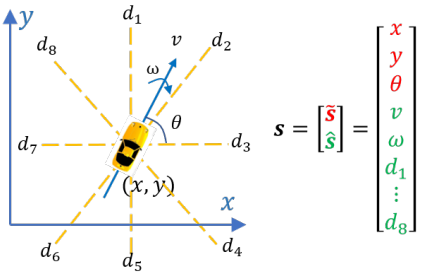

Fig. 3: State definition of the simulated car in Section IV-A

For example, in the simulated car experiment in Section IV-A, the full state contains the internal states of the car, including the position (x, y), heading θ, speed v, and turn rate ω. In addition, eight laser range measurements $d_1, \ldots, d_8$ are also part of the state s. These measurements provide distances from nearby obstacles. As one can imagine, the evolution of s can be very difficult if $f(\cdot)$ is impossible to obtain, especially in a *priori* unknown environments.

The state of the approximate system, denoted $\tilde{s}$, contains a subset of the internal states $(x, y, \theta, v, \omega)$, and evolves according to Eq. (3). In particular, for the simulated results in this paper, we choose the simple Dubins Car model to be the approximate system dynamics. As we show in Section IV-A, such simple dynamics is sufficient for improving data efficiency in model-free RL. With this choice, the remaining states are denoted $\hat{s} = (v, \omega, d_1, \ldots, d_8)$, as shown in Fig. 3.

$$\dot{\tilde{s}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \end{bmatrix} \tag{9}$$

$$\dot{\tilde{s}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v\cos\theta \\ v\sin\theta \\ \omega \\ \alpha_v \\ \alpha_\omega \end{bmatrix} \tag{10}$$

In general, we may choose $\tilde{s}$ such that a reasonable explicit, closed-form ODE model $\tilde{f}(\cdot)$ can be derived. Such a model should capture the evolution of the robotic internal state. One motivation for using an ODE is that the real system operates in continuous time, and computing the TTR function for continuous-time systems is a solved problem for sufficiently low-dimensional systems.

It is worth noting that if a higher-fidelity model of the car is desired, one may also choose the following 5D ODE approximate system model instead as Eq. (10). In this case, we would have $\tilde{s} = (x, y, \theta, v, \omega)$, and $\hat{s} = (d_1, \ldots, d_8)$. Note that the choice of an ODE model representing the real robot may be very flexible, depending on what behavior one wishes to capture. In the 3D car example given in Eq. (9), we focus on modelling the position and heading of car to be consistent with the goal. However, if speed and angular speed is deemed crucial for the task under consideration, one may also choose a more complex approximate system given by Eq. (10). To re-iterate, a good choice of approximate model is computationally tractable for the TTR function computation, and captures the system behaviors that are important for performing the desired task.

B. TTR Function as Approximate Reward

In this section, we discuss how the reward function $r(s, a)$ in the full MDP can be chosen based on the TTR function. For simplicity, we ignore a and denote it as $r(s)$, although a simple modification to the TTR function can be made

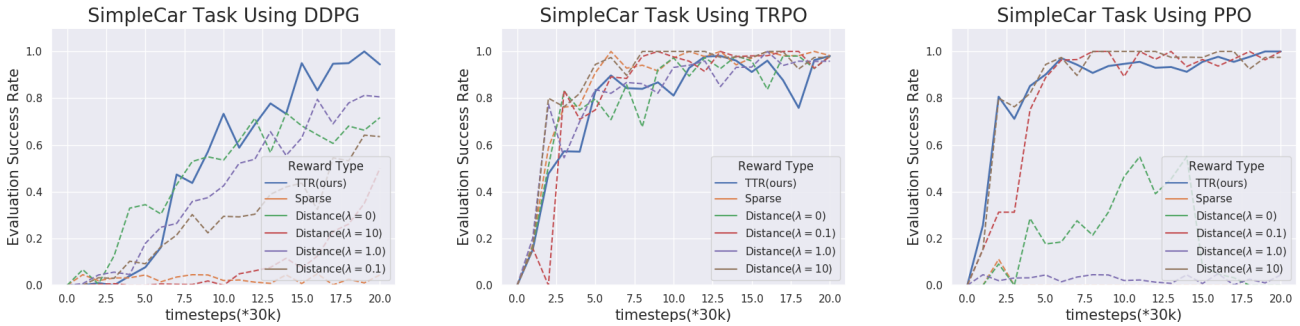

Fig. 4: Performance comparison of three different reward functions on the car example under three model-free RL optimization algorithms: DDPG, TRPO and PPO. All results are based on the mean of five runs.

to incorporate actions into the reward function. Since s is often in high dimensional space with sensor measurements involved, it is often unclear how to determine an proper reward for s. As a result, simple reward functions such as sparse and distance rewards are sometimes used.

However, this can be easily resolved in our approach by viewing $r(\cdot)$ as a function of $\tilde{s}$, the state of approximate system model we have chosen before. As mentioned earlier, $\tilde{s}$ a subset of full state s. In our method, the TTR function $\phi(\tilde{s})$ defined in Eq. (5) is transformed slightly to obtain the reward function for full MDP state s:

$$r(s) = r(\tilde{s}, \hat{s}) = \begin{cases} -\phi(\tilde{s}) & s \in \mathbf{I} \\ 1000 & s \in \mathbf{G} \\ -400 & s \in \mathbf{C} \end{cases} \tag{11}$$

By definition, $\phi(\tilde{s})$ is non-negative and $\phi(\tilde{s}) = 0$ if and only if $\tilde{s} \in \Gamma$. Thus, we use $-\phi(\cdot)$ as the reward because the state with lower goal-arrival time should be given a higher reward.

As shown in Eq. (11), we set positive reward for goal states $\mathbf{G}$ and negative reward for collision states $\mathbf{C}$. Note that the TTR-based reward can also be extended to have obstacles taken into account, or to satisfy any other design choices if required. For *intermediate* states that are neither obstacles or goals, TTR function $\phi(\cdot)$ directly provides a useful reward signal in an important subspace of the full MDP state. This is significant since the associated rewards for these intermediate states are usually quite difficult to manually design, and TTR reward does not require any manual fine-tuning. This way the RL agent learns faster compared to not having a useful reward in the subspace, and can quickly learn to generalize the subspace knowledge to the high-dimensional observations. For example, positions that are near obstacles correspond to small values in LIDAR readings, and thus the agent would quickly learn these observations correspond to bad states.

To further reduce the computational complexity of solving the PDE for more complicated system dynamics (such as a quadrotor), we may apply system decomposition methods established from the optimal control community [18]–[20] to obtain an approximate TTR function without significantly impacting the overall policy training time. Particularly, we first decompose the entire system into several sub-systems potentially with overlapping components of state variables, and then efficiently compute the TTR for each sub-system. We utilize Lax-Friedrichs sweeping-based [14] to compute the TTR function. As shown in the Table I, computation time of TTR functions are negligible compared to the time it takes to train policies.

IV. SIMULATED EXPERIMENTS

In order to illustrate the benefits from our TTR-based reward shaping method, we now present two goal-oriented tasks through two different mobile robotic systems: a simple car and a planar quadrotor. Each system is simulated in Gazebo [21], an open-source 3D physical robot simulator. Also, we utilize the Robot Operating System (ROS) for communication management between robot and simulator. For each task, we compare the performance between our TTR-based reward and two other conventional rewards: sparse and distance-based reward. For each reward function, we use three representative model-free RL algorithms (DDPG, TRPO and PPO) to demonstrate that TTR-based reward can be applied to augment any model-free RL algorithm. Full version analysis can be found at [22].

We select sparse and distance-based rewards for comparison with our proposed TTR-based reward because they are simple, easy to interpret, and easy to apply to any RL problem. These reward functions are consistent across our two simulated environments, shown in Table II. We formulate the distance-based reward as general Euclidean distance involving position and angle because the tasks we consider involve reaching some desired set of positions and angles, shown in Table II. By choosing different weights λ, the angle is assigned different weights. For both examples, we choose four different weights, $\lambda \in \{0, 0.1, 1, 10\}$. The sparse reward is defined to be 0 everywhere except for goal states (1000 reward) or collision states (-400 reward).

Task	Model	Computation Time	Decomposed
Simple Car	Eq. (9)	5 sec	No
Planar Quadrotor	Eq. (12)	90 sec	Yes

TABLE I: TTR function computational load. 'Decomposed' means if we need to decompose approximate model into subsystems in order to reduce computational cost.

A. Simple Car

The car model is widely used as standard testbed in motion planning [21] and RL [23] tasks. Here we use a "turtlebot-2" ground robot to illustrate the performance of the TTR-based

reward. The state and observation of this example are already discussed at III-A.

The car starts with randomly-sampled initial conditions from the starting area and aims to reach the goal region without colliding with any obstacle along the trajectory. Specifically, we set the precise goal state as $G : (x_g = 4\,\mathrm{m}, y_g = 4\,\mathrm{m}, \theta_g = 0.75\,\mathrm{rad})$, and states within $0.3\,\mathrm{m}$ in positional distance and $0.3\,\mathrm{rad}$ in angular distance of G are considered to have reached the goal, denoted as $S_g = \{(x, y, \theta) | 3.7\,\mathrm{m} \leq x \leq 4.3\,\mathrm{m}; 3.7\,\mathrm{m} \leq y \leq 4.3\,\mathrm{m}; 0.45\,\mathrm{rad} \leq \theta \leq 1.05\,\mathrm{rad}\}$. The TTR-based reward for this simple car task is derived from a lower-dimensional approximate car system in Eq. (9) which only considers the 3D vector (x, y, θ) as state and angular velocity ω as control.

Fig. 4 compares the performance of TTR-based reward with sparse and distance-based rewards under three different model-free algorithms. Success rate after every fixed number of training episodes is considered as qualitative assessment. In general, the car system is simpler and more stable thus obtains relatively higher success rate among different reward settings compared to the quadrotor task (shown later in Fig. 5). In particular, TTR-based reward leads to high success rate (mostly over 90%) consistently with all three learning algorithms. In contrast, sparse reward leads to poor performance with PPO, and distance-based reward leads to poor performance with DDPG.

Note that despite of the better performance from certain distance-based reward, the choice of appropriate weight for each variable is non-trivial and not transferable between different tasks. However, TTR-based reward requires little human engineering to design and can be efficiently computed once an low-fidelity model is provided.

The TTR function for the model in Eq. (9) is shown in Fig. 1 to convey the usefulness of TTR-based reward more intuitively. Here, we show the 2D slices of the TTR function at four heading angles, $\theta \in \{-\pi/4, 0, \pi/2, 3\pi/4\}$. The green star located at the upper-right of each plot is the goal area. The car starts moving from lower-middle area. Note that the 2D slices look different for different heading angles according to the system dynamics, with the contours expanding roughly in opposite direction to the heading slice.

Sparse	Distance	TTR
$r(s) = \begin{cases} 0 \\ 1000 \\ -400 \end{cases}$	$r(s) = \begin{cases} -d(\cdot)^* \\ 1000 \\ -400 \end{cases}$	$r(s) = \begin{cases} -\phi(\tilde{s})^* & s \in \mathbf{I} \\ 1000 & s \in \mathbf{G} \\ -400 & s \in \mathbf{C} \end{cases}$
$^*d(\cdot) = \begin{cases} \sqrt{(x-x_g)^2 + (y-y_g)^2 + \lambda(\theta-\theta_g)^2} & \textbf{Simple Car} \\ \sqrt{(x-x_g)^2 + (z-z_g)^2 + \lambda(\psi-\psi_g)^2} & \textbf{Planar Quadrotor} \end{cases}$		
$^*\phi(\tilde{s})$: TTR function defined in a subspace of s		

TABLE II: Reward functions tested in this work. **I**: set of intermediate states; **G**: set of goal states; **C**: set of collision states. $d(\cdot)$: generalized distance function involving angle.

B. Planar Quadrotor Model

A Quadrotor is usually considered difficult to control mainly because of its nonlinear and under-actuated dynamics. In the second experiment, we select a planar quadrotor model [24], [25], a popular test subject in the control literature, as a relatively complex mobile robot to validate that the TTR-based reward shaping method still works well even on highly dynamic and unstable system. "Planar" here means the quadrotor only flies in the vertical (x-z) plane by changing the pitch angle without affecting the roll and yaw angle.

The approximate system model has 6D internal state $\tilde{s} = (x, v_x, z, v_z, \psi, \omega)$, where x, z, ψ denote the planar positional coordinates and pitch angle, and v_x, v_z, ω denote their time derivatives respectively. The dynamics used for computing the TTR function are given in Eq. (12). The quadrotor's movement is controlled by two motor thrusts, T_1 and T_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. Similar to the car example, the full state s contains eight laser readings extracted from the "Hokoyu_utm30lx" ranging sensor for detecting obstacles, in addition to the internal state $\tilde{s}$. The objective of the quadrotor is to learn a policy mapping from states and observations to thrusts that leads it to the goal region.

$$\dot{\tilde{s}} = \begin{bmatrix} \dot{x} \\ \dot{v}_x \\ \dot{z} \\ \dot{v}_z \\ \dot{\psi} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} v_x \\ -\frac{1}{m} C_D^v v_x + \frac{T_1}{m} \sin\psi + \frac{T_2}{m} \sin\psi \\ v_z \\ -\frac{1}{m}\left(mg + C_D^v v_z\right) + \frac{T_1}{m} \cos\psi + \frac{T_2}{m} \cos\psi \\ \omega \\ -\frac{1}{I_{yy}} C_D^\psi \omega + \frac{l}{I_{yy}} T_1 - \frac{l}{I_{yy}} T_2 \end{bmatrix} \quad (12)$$

The environment for this task is shown in Fig. 6. The obstacles are fixed. The goal region is $S_g = \{(x, z, \psi) | 3.5\,\mathrm{m} \leq x \leq 4.5\,\mathrm{m}; 8.5\,\mathrm{m} \leq z \leq 9.5\,\mathrm{m}; 0.45\,\mathrm{rad} \leq \psi \leq 1.05\,\mathrm{rad}\}$. The quadrotor's starting condition is uniformly-randomly sampled from $\{(x, z) | 2.5\,\mathrm{m} \leq x \leq 3.5\,\mathrm{m}; 2.5\,\mathrm{m} \leq z \leq 3.5\,\mathrm{m}\}$ (green area in Fig. 6) and the starting pitch angle is randomly sampled from $\{\psi | -0.17\,\mathrm{rad} \leq \psi \leq 0.17\,\mathrm{rad}\}$.

Fig. 5 shows the performance of TTR-based, distance-based, and sparse reward under optimization from DDPG, TRPO and PPO. With TTR-based reward, performance is consistent over all model-free algorithms. In contrast, sparse and distance-based rewards often do not lead to quadrotor stability and consistency of performance. For example, under sparse and distance-based rewards, performance is relatively good under TRPO, but very poor under DDPG. In terms of learning efficiency, TTR-based reward achieves a success rate of greater than 90% after approximately 90K time steps regardless of the model-free algorithm. This is the best result among the three reward functions we tested.

V. CONCLUSION

In this paper, we propose TTR-based reward shaping to alleviate the data inefficiency of model-free RL on robotic tasks. By using TTR function to provide RL reward, the model-free learning process remains flexible but is endued with global guidance provided by implicit system dynamics priors. In this approach, an approximate system model chosen in a highly flexible way. By computing a TTR function based on the chosen model and integrating it as the RL reward function, the agent receives more dynamics-informed feedback and learns faster and better.

Simple and effective, TTR-based reward shaping is easy to implement and can be used as a wrapper for any model-free RL algorithm since it does not alter the original algorithmic structure. Accordingly, any additional tricks or improvements on model-free algorithms can be attached in a compatible way. From the perspective of reward shaping, our approach

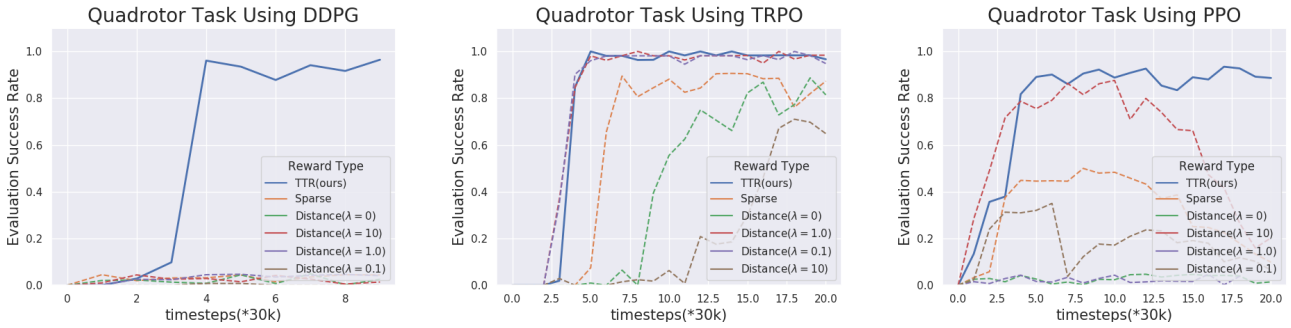

Fig. 5: Performance comparison between TTR, distance and sparse based rewards on quadrotor using three different model-free algorithms. The results are based on identical evaluation setting as car example and are concluded from five runs as well. Our TTR-based reward achieves the best in terms of efficiency and performance. **Left**: success rate comparison under DDPG algorithm **Middle**: success rate comparison under TRPO algorithm **Right**: success rate comparison under PPO algorithm

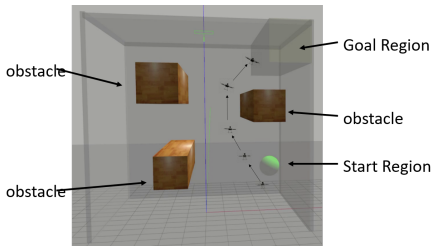

Fig. 6: Visualization of quadrotor's sequential movement after learning from TTR-based reward. The trajectory is connected by a combination of the same quadrotor at a few different time snapshots. As shown in the picture, the quadrotor has learned to make use of physical dynamics (tilt) to reach the target as soon as possible

provides a straight-forward yet distinct shaping option which requires little human engineering.

Our method is effective when an explicit robotic system dynamics are accessible. Data efficiency can be greatly improved even if only low-dimensional approximate system dynamics are available.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] 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," *arXiv preprint arXiv:1509.02971*, 2015.

[3] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust region policy optimization," in *Proc. Annual Int. Conf. Machine Learning*, 2015.

[4] I. Osband, B. Van Roy, D. Russo, and Z. Wen, "Deep exploration via randomized value functions," *arXiv preprint arXiv:1703.07608*, 2017.

[5] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," *http://arxiv.org/abs/1707.05300*, 2017.

[6] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[7] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *J. Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, May 2017.

[8] A. m. Farahmand, A. Shademan, M. Jagersand, and C. Szepesvari, "Model-based and model-free reinforcement learning for visual servoing," in *Proc. IEEE Int, Conf, Robotics and Automation*, 2009.

[9] M. Deisenroth and C. E. Rasmussen, "Pilco: A model-based and data-efficient approach to policy search," in *Proceedings of the 28th International Conference on machine learning (ICML-11)*, 2011, pp. 465–472.

[10] M. Chen, J. F. Fisac, S. Sastry, and C. J. Tomlin, "Safe sequential path planning of multi-vehicle systems via double-obstacle hamilton-jacobi-isaacs variational inequality," in *2015 European Control Conference (ECC)*, 2015, pp. 3304–3309.

[11] M. Chen, J. C. Shih, and C. J. Tomlin, "Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, 2016, pp. 1695–1700.

[12] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017.

[13] Z. Zhou, R. Takei, H. Huang, and C. J. Tomlin, "A general, open-loop formulation for reach-avoid games," in *Proc. IEEE Conf, Decision and Control*, 2012.

[14] I. Yang, S. Becker-Weimann, M. J. Bissell, and C. J. Tomlin, "One-shot computation of reachable sets for differential games," in *Proc. ACM Int. Conf. Hybrid Systems: Computation and Control*, 2013.

[15] R. Takei and R. Tsai, "Optimal trajectories of curvature constrained motion in the hamilton–jacobi formulation," *J. Scientific Computing*, vol. 54, no. 2, pp. 622–644, Feb 2013.

[16] M. Bardi and I. Capuzzo-Dolcetta, *Optimal Control and Viscosity Solutions of Hamilton-Jacobi-Bellman Equations*, ser. Modern Birkhäuser Classics, 2008.

[17] M. Bardi and P. Soravia, "Hamilton-jacobi equations with singular boundary conditions on a free boundary and applications to differential games," *Transactions of the American Mathematical Society*, vol. 325, no. 1, pp. 205–229, 1991.

[18] I. M. Mitchell, "The flexible, extensible and efficient toolbox of level set methods," *J. Scientific Computing*, vol. 35, no. 2, pp. 300–329, Jun 2008.

[19] M. Chen, S. Herbert, and C. J. Tomlin, "Fast reachable set approximations via state decoupling disturbances," in *Proc. IEEE Conf. Decision and Control*, 2016.

[20] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of reachable sets and tubes for a class of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675–3688, Nov 2018.

[21] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2004.

[22] X. Lyu and M. Chen, "Ttr-based reward for reinforcement learning with implicit model priors," *arXiv preprint arXiv:1903.09762*, 2019.

[23] D. J. Webb and J. van den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. IEEE Int.Conf. Robotics and Automation*, 2013.

[24] J. H. Gillula, H. Huang, M. P. Vitus, and C. J. Tomlin, "Design of guaranteed safe maneuvers using reachable sets: Autonomous quadrotor aerobatics in theory and practice," in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 1649–1654.

[25] S. Singh, A. Majumdar, J.-J. Slotine, and M. Pavone, "Robust online motion planning via contraction theory and convex optimization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 5883–5890.