Guiding Reinforcement Learning with Incomplete System Dynamics

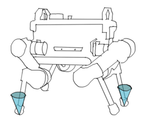
Shuyuan Wang[1], Jingliang Duan[4], Nathan P. Lawrence[2], Philip D. Loewen[2]
Michael G. Forbes[3], R. Bhushan Gopaluni[1], Lixian Zhang[5] *Fellow IEEE*

***Abstract*— Model-free reinforcement learning (RL) is inherently a reactive method, operating under the assumption that it starts with no prior knowledge of the system and entirely depends on trial-and-error for learning. This approach faces several challenges, such as poor sample efficiency, generalization, and the need for well-designed reward functions to guide learning effectively. On the other hand, controllers based on complete system dynamics do not require data. This paper addresses the intermediate situation where there is not enough model information for complete controller design, but there is enough to suggest that a model-free approach is not the best approach either. By carefully decoupling known and unknown information about the system dynamics, we obtain an embedded controller guided by our partial model and thus improve the learning efficiency of an RL-enhanced approach. A modular design allows us to deploy mainstream RL algorithms to refine the policy. Simulation results show that our method significantly improves sample efficiency compared with standard RL methods on continuous control tasks, and also offers enhanced performance over traditional control approaches. Experiments on a real ground vehicle also validate the performance of our method, including generalization and robustness.**

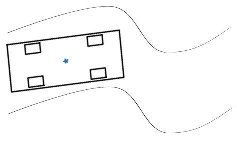
I. Introduction

Human learning, such as riding a bicycle, is often accelerated by leveraging accumulated prior knowledge about the dynamics of the world. This ability allows adults to learn new tasks through a relatively small number of trials, emphasizing the value of prior knowledge in efficient learning [1]. Humans' dynamic process of trial and error bears a striking resemblance to reinforcement learning (RL) [2]. However, despite these similarities, RL agents typically learn from scratch, which necessitates a large number of interactions with the environment. This limitation blocks RL's application toward real-world continuous control tasks.

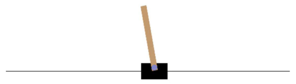
The current work seeks to narrow this gap by enhancing RL algorithms with partial system knowledge. In particular, we focus on incorporating prior knowledge related to model structure and some parameters into RL algorithms. To illustrate, consider the inverted pendulum system in Fig. 1, a classic control problem. In this system, certain parameters related to the mechanical properties of the system (like length and inertia) are unknown, while the model structure and other parameters are known exactly, such as the 0 and 1 elements in the matrices. Most RL methods do not take such information into account.



$$\tau = M(\theta)\cdot\ddot{\theta} + C(\theta,\dot{\theta})\cdot\dot{\theta} + G(\theta) + J_c^T \cdot F_{\text{ground}}$$



$$\begin{aligned} m\ddot{x} &= m\dot{y}\dot{\varphi} + F_{xf} + F_{xr} \\ m\ddot{y} &= m\dot{x}\dot{\varphi} + F_{yf} + F_{yr} \\ I_z\ddot{\varphi} &= aF_{yf} - bF_{yr} \end{aligned}$$



$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & \frac{-(I+ml^2)b}{I(M+m)+Mml^2} & \frac{m^2g^2}{I(M+m)+Mml^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{-mlb}{I(M+m)+Mml^2} & \frac{mgl(M+m)}{I(M+m)+Ml^2} & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ \phi \\ \dot{\phi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{I+ml^2}{I(M+m)+Mml^2} \\ 0 \\ \frac{ml}{I(M+m)+Mml^2} \end{bmatrix} u$$

Fig. 1: Dynamics model and partial model knowledge for different tasks: quadruped robot; self-driving vehicle; inverted pendulum. Green elements represent known parameters and structure; Red elements represent unknown parameters.

In this work, we introduce a control module into RL policies that harnesses partial model knowledge, thereby aiming to improve both the sample efficiency and generalizability of RL algorithms. Our proposed method stands out by effectively decoupling known and unknown information. This results in a more efficient use of model information and an expected improvement in sample efficiency. Our approach integrates partial model knowledge into RL while preserving the core RL structure. Thus, it can be readily implemented with mainstream RL methods. Empirical results from simulations and real-world robot experiments show that our approach outperforms several prominent RL algorithms.

We highlight the following contributions:

1) We introduce a novel framework that brings partial model knowledge into RL in a decoupled way, bridging the RL and control frameworks without disrupting the RL structure.
2) Our method enhances RL's sample efficiency, consistency, and generalization while maintaining its capacity

*This work was supported by NSERC and Honeywell Connected Plant. Video available at `https://youtu.be/xGNNiuYJh98`.

[1]Department of Chemical and Biological Engineering, University of British Columbia, Vancouver, Canada, `wshuyuan@student.ubc.ca`, `bhushan.gopaluni@ubc.ca`

[2]Department of Mathematics, University of British Columbia, Vancouver, Canada, `lawrence@math.ubc.ca`, `loew@math.ubc.ca`

[3]Honeywell Process Solutions, North Vancouver, Canada, `michael.forbes@honeywell.com`

[4]School of Mechanical Engineering, University of Science and Technology Beijing, Beijing, China, `duanjl@ustb.edu.cn`

[5]Department of Astronautics, Harbin Institute of Technology, Harbin, China, `lixianzhang@hit.edu.cn`

for optimality.

II. Related works

A. Reinforcement learning and adaptive control

RL algorithms have proven to be highly effective in tackling complex decision-making and control tasks through trial-and-error. However, such data collection usually entails high sample complexity, which limits the feasibility of RL algorithms when applied in real-world applications [3], [4].

Various enhancements have been developed to bolster RL performance, such as double Q-learning [5] for mitigating overestimation, proximal policy optimization [6] for safe updates, and soft iterations [7] for exploration. Nonetheless, RL's nature of learning from scratch remains unchanged, impeding faster learning and generalization. Our approach breaks the policy network 'black box' by integrating a control module that utilizes partial model knowledge, thus enhancing learning without altering the existing framework. This strategy maintains existing frameworks' strengths and also capitalizes on the benefits of control and partial models for more efficient learning.

Our method is distinct from model-based RL [8], [9], which constructs a model from scratch to generate synthetic data or estimate cost gradients relative to policy parameters. Our approach diverges by leveraging known structures and parameters a priori, rather than building from zero. We aim to incorporate this knowledge into a model-free framework, merging model-free RL's flexibility with control theory's precision. Notably, model-based RL methods such as ME-TRPO [10], SLBO [11], and MBPO [12] still utilize model-free RL for policy optimization after model creation, indicating that improvements in model-free techniques could benefit model-based strategies. Thus, our method complements rather than conflicts with existing paradigms.

In the context of adaptive control, [13], [14] explored policy iteration with partial knowledge. These works assume the control matrix B is fully known and the state transition matrix A is fully unknown for linear systems $x_{k+1} = Ax_k + Bu_k$. Although the titles include 'partial model', these assumptions narrow the scope of partial knowledge to a specific subset, limiting its breadth in the domain of partial models.

B. End-to-end learning

In the context of control and planning, end-to-end learning optimizes any parameters (such as policy parameters, and model parameters) directly from the overall performance metric. As a milestone, [15] shows that embedding the planning module into an RL policy network can augment the policy's generalizability, leading to acceptable performance even in unexplored domains. However, [15] assumes discrete action and state spaces. Differentiable model predictive control (MPC) [16], [17], [18] provides insights to incorporate a continuous-action control module into a network. However, these approaches assume a linear model structure and quadratic cost. These issues make the approaches fall short in handling model biases and flexible rewards, thereby limiting their application to RL tasks. Our method is built upon these methods but bypasses these limitations by introducing a compensation structure, enabling control to dance better with RL.

III. Preliminary: Differentiable MPC

In order to incorporate partial knowledge-based control within the reinforcement learning framework, we introduce a differentiable control module as an integrated layer within the RL policy network. At the core of this module lies a linear MPC strategy, formulated as follows:

$$x(t+dt) - x_d(t+dt) = A(x(t) - x_d(t)) + B(u(t) - u_d(t)), \tag{1}$$

where $x(t)$ represents the system state at time t, $u(t)$ denotes the control input, and $x_d(t)$ and $u_d(t)$ correspond to the desired state and control input, respectively. When x_d and u_d are set to zero, the problem reduces to the well-known Linear Quadratic Regulator (LQR). The objective is to minimize a quadratic stage cost:

$$\begin{aligned} J = \sum_{t=0}^{\infty} &(x(t) - x_d(t))^\top Q (x(t) - x_d(t)) \\ &+ (u(t) - u_d(t))^\top R (u(t) - u_d(t)) \end{aligned} \tag{2}$$

with weight matrices $Q \geq 0$ and $R > 0$ specifying the importance of state and control input deviations in the cost function. In our framework, the weights can be aligned with the rewards from the RL environment or predefined based on common practices. As shown in the experiment section, our proposed framework can adapt well to various kinds of objective functions from the environment.

The action u that optimizes the accumulated cost is obtained by solving the Discrete Algebraic Riccati Equation (DARE)

$$P = Q + A^\top PA - A^\top PB(R + B^\top PB)^{-1} B^\top PA, \tag{3}$$

and the optimal controller is given by

$$u = -(R + B^\top PB)^{-1} B^\top PA(x - x_d) + u_d. \tag{4}$$

To integrate linear MPC as a differentiable layer within the RL framework, it we need to know the sensitivity of the solution matrix P with respect to the dynamics A and B. As shown in [18], this can be achieved by differentiating through (3) and solving for a closed-form solution. Here is the result.

Proposition 1 ([18]): Let P be the stabilizing solution of DARE, and assume that Z_1^{-1} and $(R + B^\top PB)^{-1}$ exists. Then the Jacobians of the implicit function defined by DARE are given by

$$\frac{\partial \operatorname{vec} P}{\partial \operatorname{vec} A} = Z_1^{-1} Z_2, \quad \frac{\partial \operatorname{vec} P}{\partial \operatorname{vec} B} = Z_1^{-1} Z_3, \tag{5}$$

where Z_1, Z_2, Z_3 are defined by

$$\begin{aligned} Z_1 &= \mathbf{I}_{n^2} - \left(A^\top \otimes A^\top\right)\left[\mathbf{I}_{n^2} - \left(PBM_2B^\top \otimes \mathbf{I}_n\right)\right. \\ &\quad +(PB \otimes PB)\left(M_2 \otimes M_2\right)\left(B^\top \otimes B^\top\right) \\ &\quad \left.-\left(\mathbf{I}_n \otimes PBM_2B^\top\right)\right] \\ Z_2 &= \left(\mathbf{V}_{n,n} + \mathbf{I}_{n^2}\right)\left(\mathbf{I}_n \otimes A^\top M_1\right) \\ Z_3 &= \left(A^\top \otimes A^\top\right)\left[(PB \otimes PB)\left(M_2 \otimes M_2\right)\right. \\ &\quad \left.\left(\mathbf{I}_{m^2} + \mathbf{V}_{m,m}\right)\left(\mathbf{I}_m \otimes B^\top P\right)\right] \end{aligned} \tag{6}$$

and M_1, M_2, M_3 are defined by

$$M_1 := P - PBM_2B^\top P, M_2 := M_3^{-1}, M_3 := R + B^\top PB.$$

IV. Main Method

A. Partial Knowledge Representation

We consider the dynamical system model in continuous time, given by:

$$\dot{x} = f(x, u). \tag{7}$$

We adopt a continuous-time model, as it often captures the physical properties of the system and thus contains inherent prior information. Note that this does not limit our method's ability to adapt to discrete-time systems. We will elaborate on this point later in this section.

In order to formulate known information and unknown information, we decompose the model as follows:

$$f(x, u) = f_{app}(f_1(x, u), f_2(x, u)) + f_{bias}(x, u). \tag{8}$$

Here, f_{app} represents the approximated model with known structure, while $f_{bias}(x, u)$ accounts for the bias of the approximate model. Within $f_{app}(x, u)$, we have both known information $f_1(x, u)$ and unknown information $f_2(x, u)$, and we also know how f_1 and f_2 combine to form f_{app}. In essence, $f_2(x, u)$ and $f_{bias}(x, u)$ make up the unknown part of the overall model. Taking the inverted pendulum example from Fig.1, f_{app} represents the linearized model of the system. f_{bias} captures the error introduced by this linearization, which is not explicitly represented or learned within our framework. f_1 corresponds to the known effects of the input, represented by zeros and ones in the matrix, while f_2 deals with the impact of unknown elements in the matrix on the input. The unknown parameters within f_2 are denoted by ψ, making f_{app} rewritten as $f_{app}(\psi)$. Our framework focuses on $f_{app}(\psi)$, leveraging the capabilities of RL to learn control strategies that effectively address the uncertainties and biases represented by f_2 and f_{bias}.

B. Framework

We construct a novel policy network that incorporates a differential control module. This module enables planning based on partial knowledge and facilitates training within an RL framework. To address the action errors stemming from the model's bias and uncertainties associated with unknown parameters, an additive, separate neural network is introduced for compensation, thereby enhancing performance. Together, the policy with partial system knowledge can be deployed in a standard model-free RL pipeline. Fig. 2 illustrates our framework.

In the module, f_{app} is built as a differentiable computational graph, with ψ representing its unknown but trainable parameters. The module first linearizes the nonlinear model around the reference trajectory or setpoint, x_d and u_d. This is done by taking the ***in-graph gradient*** with respect to x_d and u_d. The output is a linear model with coefficient matrices given by

$$A(\psi) = \left.\frac{\partial f_{app}}{\partial x}\right|_{x_d}, B(\psi) = \left.\frac{\partial f_{app}}{\partial u}\right|_{u_d}. \tag{9}$$

Given step size τ, we discretize the continuous model with the Euler method:

$$A_{\text{dis}}(\psi) = \tau A(\psi) + I, B_{\text{dis}}(\psi) = \tau B(\psi) \tag{10}$$

where I is the identity matrix with the same dimension as $A(\psi)$. If the system (8) originally operates in discrete time, the linearization directly applies to the model with partial knowledge, bypassing the need for discretization

$$A_{\text{dis}}(\psi) = \left.\frac{\partial f_{app}}{\partial x}\right|_{x_d}, B_{\text{dis}}(\psi) = \left.\frac{\partial f_{app}}{\partial u}\right|_{u_d}. \tag{11}$$

This direct linearization simplifies the adaptation of our method to inherently discrete systems.

With $A_{\text{dis}}(\psi)$ and $B_{\text{dis}}(\psi)$, the parameterized value matrix $P(\psi)$ can be obtained by solving the DARE (3). The gradient of P with respect to ψ will be introduced in the section IV-C. With $P(\psi)$, the module calculates a sub-optimal baseline action by

$$\hat{u}(\psi) = -K(\psi)(x - x_d) + u_d, \tag{12}$$

where $K(\psi)$ is given by

$$(R + B_{\text{dis}}(\psi)^\top P(\psi) B_{\text{dis}}(\psi))^{-1} B_{\text{dis}}(\psi)^\top P(\psi) A_{\text{dis}}(\psi), \tag{13}$$

which follows exactly the scheme of linear MPC.

In contrast to existing differentiable LQR and MPC methods [18], [16], our framework can mitigate the model bias. This is achieved by introducing the corrective structure

$$u = \hat{u} + \delta u. \tag{14}$$

Here δu is generated from a trainable neural network, with input directly as the state. The δu is used to address the suboptimality introduced by the linear control with unknown parameters, and the model bias term $f_{bias}(x, u)$ from the original nonlinear model and the linearization error from (9). We hypothesize that this approach will enjoy a lower learning burden and hence faster learning speed. For deterministic policy in DDPG and TD3, δu is output from a network directly. For stochastic policy in SAC and PPO, the output of the network is the mean and standard deviations of a Gaussian distribution. We admit that the initial value and magnitude of δu may impact the training outcomes. At this early stage, we have not yet explored theoretical guidance for these parameters. In our experiments, we set the initial value of δu to 0 in order to emphasize the dominant role of $\hat{u}$.

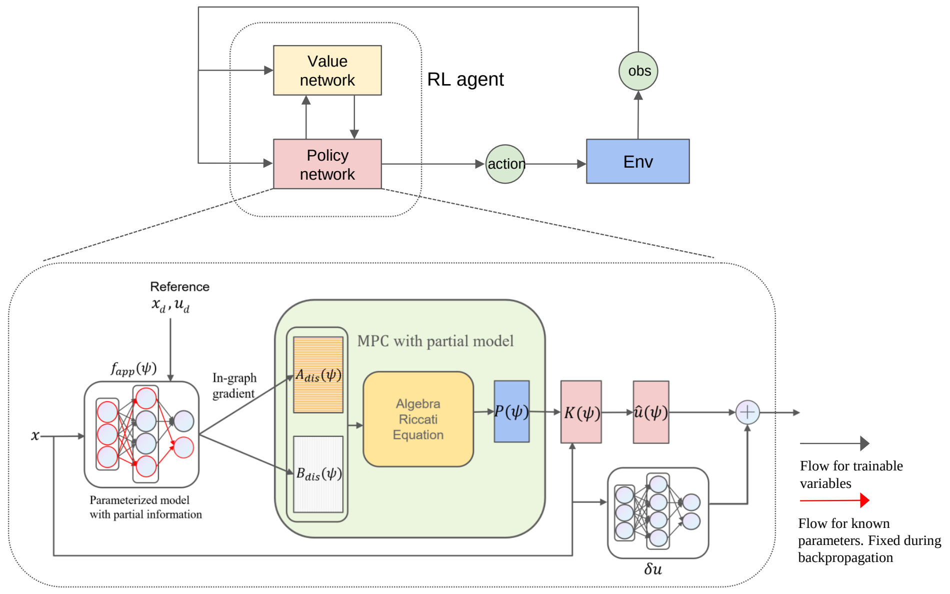


Fig. 2: Schematic diagram for our policy network with partial knowledge control module inside.

C. Training

All the operations in the proposed framework are differentiable, and consequently, the whole process can be trained through backpropagation. To separate the contribution from known parameters and unknown parameters, ***our framework keeps the known parameters fixed and only backpropagates the gradient through the unknown parameters***. In this way, the contributions from different information are decoupled. Therefore, RL only learns the policy with respect to unknown information.

For the backpropagation within MPC, we use the chain rule to calculate the gradient:

$$\frac{\partial P}{\partial \psi} = \frac{\partial P}{\partial A}\frac{\partial A}{\partial \psi} + \frac{\partial P}{\partial B}\frac{\partial B}{\partial \psi}. \tag{15}$$

By keeping the known model parameters constant, the gradients of P, A, B w.r.t. parameters outside of ψ are eliminated. The $\frac{\partial A}{\partial \psi}$ and $\frac{\partial B}{\partial \psi}$ can be obtained using Automatic Differentiation (AD) tools provided in standard packages such as PyTorch [19] and TensorFlow [20]. Combined with the analytical solutions of $\frac{\partial P}{\partial A}$ and $\frac{\partial P}{\partial B}$ from Prop. 1, the gradient of P with respect to unknown parameters is calculated.

V. Experiments

A. Simulation results

We evaluate our method using tasks from OpenAI gym [21] and the MuJoCo physics engine [22]: CartPole, and Inverted Double Pendulum, which are widely recognized benchmarks in control and RL. The goal of the tasks can be summarized as achieving an expected configuration.

We demonstrate the following features of our method:

1) Faster learning speed than its corresponding base RL method.
2) Performance improvement over LQR.
3) Adaptation to different reward functions.

To better showcase the strength of our method, we have modified the original Gym environments. For CartPole, we retained the original environment's alive bonus of 1 but changed the action space from discrete to continuous. For the Inverted Double Pendulum (IDP), we removed the environment's original alive bonus of 10, relying solely on the quadratic cost of configuration and velocity to motivate the RL agent. Specifically, the reward is calculated as the negative sum of the distance penalty, $x^2 + 2(y - 1.2)^2$, where x and y are the coordinates of the pendulum tip, and the velocity penalty, $10v_1^2 + 20v_2^2$, where v_1 and v_2 are the angular velocities of the two pendulum joints. This encourages minimizing deviations from the upright position and penalizes excessive joint velocities. Experiments demonstrate that traditional RL methods struggle to learn from quadratic costs effectively.

For our baseline algorithms, we selected SAC [7], TD3 [23], and Policy Gradient (PG) [24]. When our method is applied to these baselines, we denote them as PK-SAC, PK-TD3, and PK-PG, respectively. (PK stands for Partial Knowledge.)

For our partial knowledge-based method, we represent partial knowledge using the model structure as shown in Table I, where all mechanical properties, such as mass and length, are treated as unknown parameters. Specifically, for the CartPole task, to demonstrate the limits of our approach, we intentionally initialize the model's unknown parameters with values that render the system unstable under a classic LQR controller. For the IDP task, we broaden the range of initial states in the environment to include configurations that are difficult for LQR to control effectively. The baseline methods

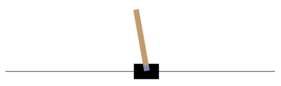
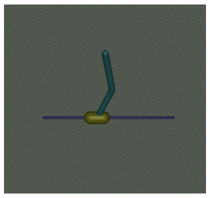
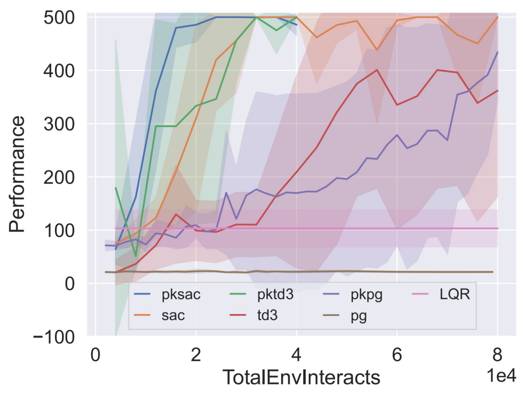
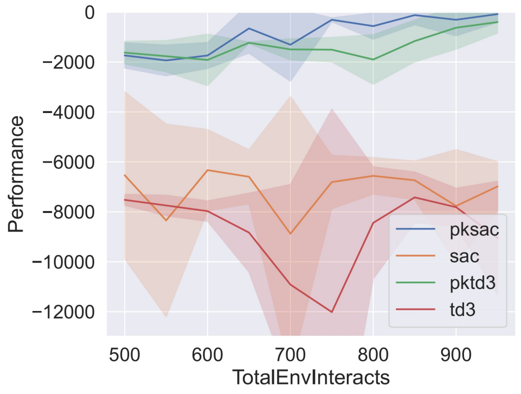
CartPole	Inverted Double Pendulum
	
$\ddot{\theta} = \frac{g\sin(\theta) - \cos(\theta)\left(\frac{u - m_p l\dot{\theta}^2 \sin(\theta)}{m_p + m_c}\right)}{\frac{1}{10}\left(\frac{4}{3} - \frac{m_p \cos(\theta)^2}{m_p + m_c}\right)}$ $\ddot{x} = \frac{u - m_p l\dot{\theta}^2\sin(\theta)}{m_p + m_c} - \frac{m_p l\ddot{\theta}\cos(\theta)}{m_p + m_c}$	$(m_0 + m_1 + m_2)\ddot{x} + (0.5m_1L_1 + m_2L_1)\cos\theta_1\ddot{\theta}_1 + 0.5m_2L_2\cos\theta_2\ddot{\theta}_2 - (0.5m_1L_1 + m_2L_1)\sin\theta_1\dot{\theta}_1^2 - 0.5m_2L_2\sin\theta_2\dot{\theta}_2^2 = u$ $\left(0.25m_1L_1^2 + m_2L_1^2 + m_1L_1^2/12\right)\ddot{\theta}_1 + (0.5m_1L_1 + m_2L_1)\cos\theta_1\ddot{\theta}_0 + 0.5m_2L_1L_2\cos(\theta_1 - \theta_2)\ddot{\theta}_2 + 0.5m_2L_1L_2\sin(\theta_1 - \theta_2)\dot{\theta}_2^2 - g(0.5m_1L_1 + m_2L_1)\sin\theta_1 = 0$ $0.5m_2L_2\cos\theta_2\ddot{\theta}_0 + 0.5m_2L_1L_2\cos(\theta_1 - \theta_2)\ddot{\theta}_1 + \left(0.25m_2L_2^2 + m_2L_2^2/12\right)\ddot{\theta}_2 - 0.5m_2L_1L_2\sin(\theta_1 - \theta_2)\dot{\theta}_1^2 - 0.5m_2gL_2\sin\theta_2 = 0$

TABLE I: Visualizations and dynamics models of CartPole and Inverted Double Pendulum (IDP): Displacement x and angle θ for the cart with cart mass m_c and pole mass m_p; Displacement x, angles θ_1 and θ_2, and masses m_0 (the cart), m_1 (the pole below), and m_2 (the pole above) for Inverted Double Pendulum. All mechanical properties, such as mass and length, are unknown parameters. The models are derived from Lagrangian mechanics.



(a) CartPole with only alive bonus.



(b) IDP with only quadratic cost.

Fig. 3: Training curves on the control benchmarks. Solid lines show the mean; shaded regions show the standard deviations over five runs.

TABLE II: Cost comparison between our method and LQR on IDP.

Initial velocities	PKSAC	PKTD3	LQR
$[-0.13, -0.13, 0.17]$	-1012.54	**-731.33**	-10444.72
$[-0.11, -0.13, 0.17]$	-793.89	**-720.17**	-11635.68
$[-0.13, -0.13, 0.18]$	-864.21	**-709.86**	-10973.45

are adopted directly from OpenAI Spinning Up (`https://github.com/openai/spinningup`), a renowned library recognized for its implementations of RL algorithms that consistently achieve state-of-the-art performance as reported in the literature.

We conducted five separate tests for each algorithm, using distinct random seeds across all algorithms. All the experiments are conducted on a platform with AMD 3700X, 8 core, 3.6GHZ, and 16G memory.

1) Enhanced Learning Speed: Figure 3 compares all algorithms in terms of the performance. This comparison demonstrates that our PK-based method can achieve superior performance with significantly less data than the baselines, demonstrating the enhanced sample efficiency of our approach. Notably, our framework improves the performance of algorithms that previously failed to learn in more complex settings. We applied Policy Gradient (PG) to our CartPole task, which is traditionally used as a fundamental benchmark within discrete action spaces. As shown in Fig. 3a, PG was unsuccessful when adapted to continuous action spaces, demonstrating a significant gap in its applicability. However, by integrating our partial model framework into PG, thereby creating PKPG, we significantly enhance its learning capability and stimulate it to control the system successfully.

Similarly, in the IDP experiments that relied solely on quadratic cost, baseline algorithms such as SAC and TD3 were also unable to successfully learn to control the system. Our method, however, not only achieves exceptionally high returns from the outset but also continues to enhance performance within just a few hundred steps. Note that our curves only display the learning process within the first 1,000 steps, however for SAC and TD3, training extended beyond 80,000 steps, yet these algorithms consistently failed to acquire a successful control strategy for the IDP.

In addition, the results show that upon integration with our approach, every baseline improves sample efficiency and

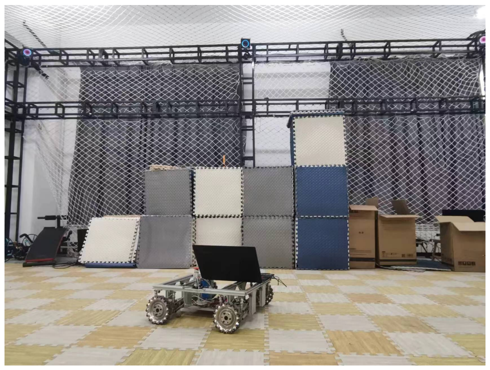


Fig. 4: Experiment overview. The field is $6 \times 8\,\mathrm{m}$, with 8 cameras on top of the field capturing the pose and position of the robot. A target for the capturing system is fixed on the side of the robot.

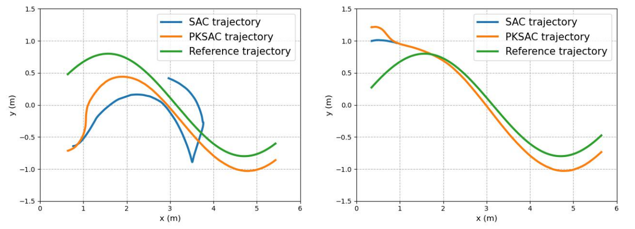
displays a reduced variance, indicating a more consistent learning process.

2) Adaptability to Various Reward Functions: The control module optimizes based on a presumed quadratic cost. However, as observed with the CartPole task, which uses a simple reward structure consisting only of a binary alive bonus (0 or 1), our method augments the performance of baselines, irrespective of the alignment between the environment's reward and the control module's cost. This indicates our method's adaptability in learning the optimal policy specific to the environment's reward structure.

3) Comparisons with LQR: In the CartPole task, we initialized our partial model with settings that led to an unstable system configuration. We subjected LQR to the same initial conditions for a fair comparison, as illustrated in Figure 3a. Under these settings, LQR could only achieve a performance score of around 100, significantly lower than our method. In the IDP task, we benchmarked our learned policy against LQR with accurately defined model parameters. While LQR is generally capable of stabilizing the system, it struggles to maintain control under certain challenging initial conditions, failing to keep the system within the environmental constraints. Leveraging the adaptive capabilities inherent in RL, our method demonstrates the ability to learn and perform under these difficult conditions. We present the accumulated costs encountered in such scenarios in Table II, where the vectors in the first column include the cart's linear velocity and the angular velocities of the first and second poles.

B. Real-world experiments

In our real-world experiments, we used a four-wheeled Mecanum robot to follow a sinusoidal path, $y = 0.8\sin(x)$, aiming to achieve a consistent travel speed of $0.35\,\mathrm{m/s}$ along the desired path, from various initial positions far from the reference trajectory. We conducted two sets of experiments: one with the robot starting above the sine wave, and the other starting below, testing its ability to converge to the target speed along the trajectory. The tracking performance was defined as a quadratic cost function including the squared sum of the orientation error, deviation in the y direction,



(a) Trajectories starting below the sine wave (b) Trajectories starting above the sine wave

Fig. 5: Trajectories following our method, SAC, and the reference.

and the difference from the desired tracking speed.

We use the OptiTrack Motion Capture System to capture the robot's positioning and orientation. An onboard computer running ROS processed the real-time data. The sampling interval for the controller was $0.1\,\mathrm{s}$. The floor is covered with a foam mat, bringing hard-to-model divergence between simulation and reality.

The robot's state variables included its position x and y, yaw angle θ, linear velocity v, and angular velocity ω. The controller generated outputs in terms of δv and $\mathrm{d}\delta\omega$, which denote the incremental changes in linear and angular velocities within the sampling interval. In this experiment, we implemented our approach with SAC as the foundational framework and compared its performance directly against the baseline SAC. The policies were pre-trained offline and deployed to the real world without any fine-tuning.

TABLE III: Tracking error under different scenarios and different methods.

Scenario	PKSAC	SAC	Improvement (over SAC)
Upper start	**126.27**	173.68	27.3%
Lower start	**72.55**	120.72	39.9%

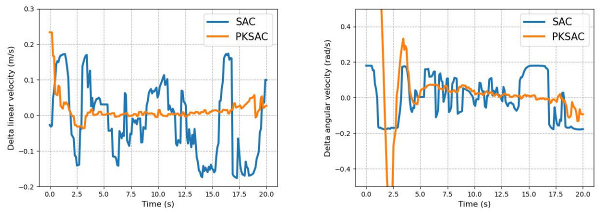


Fig. 6: Actions δv and $\delta\omega$ with time. Left is δv, right is $\delta\omega$.

In Fig. 5, it is clear that our method closely follows the desired sine curve trajectory, while the baseline SAC exhibits significant deviations due to real-world complexities such as wheel slippage and inaccuracies in lower-level controllers. Moreover, as illustrated in Fig. 5b, SAC struggles with early termination; in certain start positions, it prematurely halts the trajectory to avoid accumulating further errors. This issue is observed in both simulations and real-world experiments. In contrast, our method demonstrates superior tracking precision. The incorporation of the control module enhances the robustness of our approach, effectively bridging the sim-to-real gap and guiding the robot along a higher-reward trajectory. Specifically, Table III illustrates the reduction in accumulated tracking error achieved by our method.

A detailed examination of the robot's actions, as illustrated in Fig. 6, shows that our method produces actions with less fluctuation compared to SAC. This figure shows the actions for the case where the robot starts below the sine wave. Stable actions are essential for comfort and the health of the platform. This clear distinction accentuates the added value and efficacy of our control module.

In summary, the inherent challenges posed by the simulation-to-reality gap were more pronounced in the baseline SAC. In stark contrast, our extended method consistently demonstrated superior robustness and adaptability.

VI. Discussion

One advantage of our method is that it doesn't break the framework of RL. Consequently, other methods dealing with safety and stability can be extended to our method, which is important for RL-based controller design. It is worth noting that if the system information is perfectly known, meaning that there is no model bias and all the parameters inside the dynamics are accurate, our framework reduces to a control framework.

Our method is currently restricted to regulation and tracking style problems and requires a differentiable model. Such tasks are often fit for an LQR controller. For more complex tasks such as the locomotion of HalfCheetah, and Humanoid that contain discontinuous contact force, our method currently has limitations. We leave the adaption towards more complex tasks as future work.

VII. Conclusion

In control tasks, some partial parametric model information is often known, but seemingly under-utilized in model-free RL. We have developed a flexible framework that allows RL to leverage such information solely by adapting the policy network. Experiments show that our framework bridges RL and control theory, significantly improving sample efficiency and sim-to-real transfer compared to RL alone, and enhancing performance over conventional control methods. Future work will explore incorporating such information into the value network as well and extending our method to more complex tasks, such as locomotion and vision-based control.

References

[1] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman, "Building machines that learn and think like people," *Behavioral and brain sciences*, vol. 40, p. e253, 2017.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[3] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *International conference on machine learning*, pp. 1282–1289, PMLR, 2019.

[4] Y. Yao, L. Xiao, Z. An, W. Zhang, and D. Luo, "Sample efficient reinforcement learning via model-ensemble exploration and exploitation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4202–4208, IEEE, 2021.

[5] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proceedings of the 30th AAAI conference on artificial intelligence*, vol. 30, pp. 2094–2100, 2016.

[6] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[7] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[8] 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)*, pp. 465–472, 2011.

[9] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez, and Y. Tassa, "Learning continuous control policies by stochastic value gradients," in *Advances in Neural Information Processing Systems*, vol. 28, Curran Associates, Inc., 2015.

[10] T. Kurutach, I. Clavera, Y. Duan, A. Tamar, and P. Abbeel, "Model-ensemble trust-region policy optimization," in *International Conference on Learning Representations*, 2018.

[11] Y. Luo, H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma, "Algorithmic framework for model-based deep reinforcement learning with theoretical guarantees," in *International Conference on Learning Representations*, 2019.

[12] I. Clavera, J. Rothfuss, J. Schulman, Y. Fujita, T. Asfour, and P. Abbeel, "Model-based reinforcement learning via meta-policy optimization," in *Proceedings of The 2nd Conference on Robot Learning*, vol. 87 of *Proceedings of Machine Learning Research*, pp. 617–629, PMLR, 29–31 Oct 2018.

[13] H. Modares, F. L. Lewis, and M.-B. Naghibi-Sistani, "Integral reinforcement learning and experience replay for adaptive optimal control of partially-unknown constrained-input continuous-time systems," *Automatica*, vol. 50, no. 1, pp. 193–202, 2014.

[14] D. Vrabie, O. Pastravanu, M. Abu-Khalaf, and F. Lewis, "Adaptive optimal control for continuous-time linear systems based on policy iteration," *Automatica*, vol. 45, no. 2, pp. 477–484, 2009.

[15] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," *Advances in neural information processing systems*, vol. 29, 2016.

[16] B. Amos, I. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," *Advances in neural information processing systems*, vol. 31, 2018.

[17] B. Amos and J. Z. Kolter, "Optnet: Differentiable optimization as a layer in neural networks," in *International Conference on Machine Learning*, pp. 136–145, PMLR, 2017.

[18] S. East, M. Gallieri, J. Masci, J. Koutnik, and M. Cannon, "Infinite-horizon differentiable model predictive control," *Proceedings of ICLR 2020*, 2020.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[21] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," *OpenAI*, 2016.

[22] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*, pp. 5026–5033, IEEE, 2012.

[23] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*, pp. 1587–1596, 2018.

[24] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.