Hybrid reinforcement learning control for a micro quadrotor flight

Jaehyun Yoo[1], Dohyun Jang[2], H. Jin Kim[2], and Karl H. Johansson[3]

***Abstract*—This paper presents a combination of reinforcement learning (RL) and deterministic controllers to learn a quadrotor control. Learning the quadrotor flight in a standard RL approach requires many iterations of trial and error, which may bring about risky exploration and battery consumption. In this paper, we integrate a classical controller such as PD (proportional and derivative) or LQR (linear quadratic regulator) with a RL policy using their linear combination. The proposed method is not only simple to use, but also fast in learning convergence. When the algorithm is evaluated for a quadrotor trajectory tracking by means of a velocity control for both simulation and experiment, it demonstrates the faster convergence rate and better control performance in comparison with an existing rapid model-based RL method.**

***Index Terms*—Reinforcement learning; model-based learning; micro quadrotor.**

I. INTRODUCTION

REINFORCEMENT learning (RL) is getting attention for autonomous control [1], [2] since its data-driven approach can reduce demands on engineering knowledge. Model-based RL methods have been developed to shorten the learning iterations [3], [4]. The probabilistic inference for learning control (PILCO) [5] is known as one of the most rapid model-based RL approaches, which has data-efficient structure based on Bayesian inference with Gaussian process.

Nevertheless, PILCO may still require many online-learning iterations in practice because of the inaccurate policy search during the first few iterations. Given that the initial policy is confined to random action, a minor or a meaningless update on the policy search is made, which may cause the failure of control during initial iterations. To address this problem, we design the initial policy supported by deterministic controllers. It can not only boost the learning convergence, but also stabilize the balance between exploration and exploitation.

The proposed RL algorithm is made by linear combination of a deterministic controller with PILCO framework. The main advantage is the simplicity and the rapid learning convergence. In this paper, two different combinations are introduced according to availability of system model information. First, PD-RL in which a PD controller is combined with PILCO that can be used for the case when a system model is unknown because both of PD and PILCO do not require the model information. On the other hand, similar to some RL methods to use a system model [6]–[8], LQR-RL can be used if a system model is given, where LQR is an optimal control method. In case of a quadrotor, its dynamic model can be approximated by linearization [9].

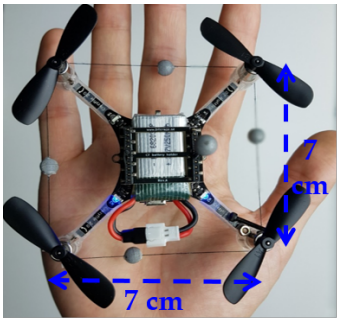


Fig. 1: Learning 32 grams micro quadrotor flight

Linear combination of multiple controllers within RL approaches can be found from some existing works. In [10], reinforcement learning is combined with model predictive control to compensate for a model-plant mismatch. In [11], PID is merged to PILCO. Different from the proposed PD-RL in this paper, its policy is replaced with PID control without a fusion mechanism. In [12], deep neural network is employed to learn a dynamics although the sample-expensive learning can be its limitations. In [13], a PD controller is added to the RL policy only for the first few iterations so that the controller does not aid the final policy. Different from these existing methods, the proposed method is neutral between the deterministic controller and the policy in the sense that the final controller does not rely on only the learned outcome. Rather, the developed method lets the classic controller and the RL policy complement each other in a way that the RL policy reinforces the deterministic controller's insufficiency.

The proposed algorithm is evaluated to learn a control of the micro quadrotor control shown in Fig. 1. Compared to relatively stable applications such as robotic manipulation [14], [15] and ground robot [16], [17], the flight learning is challenging because the quadrotor becomes easily unstable when even small erroneous control is executed. For simulation study, we compare the original PILCO, PD-RL and LQR-RL, and evaluate difference of applying a linear policy and a nonlinear policy for different trajectory-tracking references. The experiment validates practicality of the proposed methods by showing successful learning for the safe flight in only a few

[1]Jaehyun Yoo is with the School of Electrical and Electronic Engineering, Hankyong National University, South Korea, jhyoo@hknu.ac.kr

[2]Dohyun Jang and H. Jin Kim are with the Department of Mechanical and Aerospace Engineering, Seoul National University, South Korea, hjinkim@snu.ac.kr

[3]Karl H. Johansson is with the School of Electrical Engineering, KTH Royal Institute of Technology, Stockholm, Sweden, kallej@kth.se

This work was supported by the Agency for Defense Development under the contract UD 190026RD.

iterations.

The rest of this paper is organized as follows. Section II represents the proposed control methods. Section III and IV demonstrate the performances of the proposed method with simulation and experiment results, respectively. Section V devotes to concluding remarks.

II. Hybrid reinforcement learning control

Section II-A describes the problem formulation of the policy search in model-based RL framework with description of PILCO algorithm. Section II-B introduces the proposed method. 'Policy' refers to the controller learned by RL, which is updated per learning iteration. In this paper, formulation of the policy is made by the combination with PD or LQR controller.

A. Problem formulation

Consider a discrete dynamical system with state $x_t \in R^d$ and control input $u_t \in R^m$, given by

$$x_{t+1} = f(x_t, u_t) + w_t, \tag{1}$$

where $f(\cdot)$ is an unknown function and w_t is a Gaussian noise distributed as $\mathcal{N}(0, \sigma_w)$. Model-based RL learns the dynamics $f(\cdot)$ by using the training set $\{(x_{t_i}, u_{t_i}), x_{t_i+1}\}$, where t_i represent all the previous recorded time stamps in training procedure. The training set is continuously updated from control executions with the assumption that all the states are measurable. In PILCO, the dynamics is assumed to follow a Gaussian process (GP) model. Given the current state and control input $\tilde{x}_t = (x_t, u_t) \in R^{d+m}$, the prediction for the next state x_{t+1} is as follows:

$$p(x_{t+1}|x_t, u_t) = \mathcal{N}(x_{t+1}|\mu_{t+1}, \Sigma_{t+1}) \tag{2}$$

$$\mu_{t+1} = x_t + E_f[\Delta_t] \tag{3}$$

$$\Sigma_{t+1} = \mathrm{var}_f[\Delta_t], \tag{4}$$

where $\Delta_t = x_{t+1} - x_t$ and $E_f[\cdot]$ and $\mathrm{var}_f[\cdot]$ are mean and variance of f, respectively, which are represented by GP regression:

$$E_f[\Delta_t] = k_*^{\mathrm{T}} \left(K(\tilde{X}, \tilde{X}) + \sigma_w^2 I \right)^{-1} Y \tag{5}$$

$$\mathrm{var}_f[\Delta_t] = k(\tilde{x}_t, \tilde{x}_t) - k_*^{\mathrm{T}} \left(K(\tilde{X}, \tilde{X}) + \sigma_w^2 I \right)^{-1} k_*. \tag{6}$$

In (5), the training input is $\tilde{X} = [\tilde{x}_1, \cdots, \tilde{x}_n]$ and training output is $Y = [\Delta_1, \cdots, \Delta_n]$. A kernel function $k(\cdot, \cdot)$ is a Gaussian kernel in this paper, and corresponding kernel vector $k_* = k(\tilde{X}, \tilde{x}_t) \in R^{n \times 1}$ and kernel matrix $K(\tilde{X}, \tilde{X}) \in R^{n \times n}$ are also Gaussian. More details of the GP training are explained in [18].

The policy $\pi(x, \theta)$ to control a system toward a desired state should be learned. RL aims to find the optimal parameter θ by the optimization to reduce an expected long-term cost $J^\pi(\theta)$ such that

$$\min J^\pi(\theta) = \sum_{t=0}^{T} E[l(x_t, u_t)], \tag{7}$$

$$x_0 \sim \mathcal{N}(\mu_0, \Sigma_0), \tag{8}$$

where $l(x_t, u_t)$ is the immediate cost function and the initial state x_0 is given as Gaussian distribution. In this paper, the quadratic cost function is used such that

$$l(x, u) = (x - r)^{\mathrm{T}} Q (x - r) + u^{\mathrm{T}} R u, \tag{9}$$

where Q and R are weights and r is the reference. The advantage of PILCO algorithm is the analytic calculation for the gradient of expected cost $\partial J / \partial \theta$. The gradient calculation with respect to the cost function in (9) with the proposed RL policy in this paper is described in APPENDIX.

In PILCO, the parameters of policy are randomly initialized to have small values. In our experience, PILCO can be easily biased to exploration so that a minor or a meaningless policy update is made. Since many iterations for training are inevitable, it is restrictive to apply the original PILCO in practice.

B. Proposed policy

Let the policy of the original PILCO be π_{PILCO}, and the proposed two polices $\pi_{\text{PD-RL}}$ and $\pi_{\text{LQR-RL}}$, respectively. Two different types of linear combination of PILCO and deterministic controllers are as follows:

$$\pi_{\text{PD-RL}} = \pi_{\text{PILCO}} + u_{\text{PD}}, \tag{10}$$

$$\pi_{\text{LQR-RL}} = \pi_{\text{PILCO}} + u_{\text{LQR}}. \tag{11}$$

The parameter θ of π_{PILCO} is updated online, while the parameters of PD controller u_{PD} and LQR controllers u_{LQR} are deterministic. **Algorithm 1** overviews the proposed algorithm.

1) PD-RL: Because PD control does not need a system dynamics, PD-RL does not also require any additional assumption. When the system state includes derivative components such as $x = [x_1, \dot{x}_1, x_2, \dot{x}_2, \cdots] \in R^d$, the PD control can be defined as

$$u_{\text{PD}}(x) = K_{\text{PD}}(x - r), \tag{12}$$

with

$$K_{\text{PD}} = \begin{bmatrix} K_p & K_d & 0 & 0 & \cdots & 0 \\ 0 & 0 & K_p & K_d & \cdots & 0 \\ & & \vdots & & & \\ 0 & \cdots & \cdots & 0 & K_p & K_d \end{bmatrix} \in R^{m \times d}, \tag{13}$$

Algorithm 1 Hybrid reinforcement learning control

```
Initialize θ randomly in π_PILCO(x; θ)
if dynamic model is available then
    Choose LQR-RL policy in (23) or (24).
    Calculate gain K_LQR using (21).
else
    Select PD-RL policy in (15) or (17).
    Set gain K_PD manually.
end if
repeat
    Implement a hybrid policy and collect training data
    Train Gaussian Process model to learn a system model
    Evaluate the policy by predicting J^π(θ)
    Update the policy based on the gradient ∂J^π(θ)/∂θ
until θ converges
```

where K_p, K_d are tuning parameters corresponding to the state error and the derivative of the state error, respectively. The merit of the PD controller is the intuitive gain-tuning. The gain K_p responds to the convergence rate to approach the reference, and K_d increases the stability by relaxing oscillatory response of the system. From our experience, PD-RL outperforms PILCO as long as we set K_p and K_d not too big, because the insufficiency of the PD controller is covered by the RL policy as the learning repeats. When the linear policy is defined as

$$\pi_{\text{PILCO}}(x, \theta) = Ax + b, \quad \theta = \{A \in R^{m \times d}, b \in R^m\}, \tag{14}$$

the combination is as follows:

$$\pi_{\text{PD-RL}} = (A + K_{\text{PD}})x - K_{\text{PD}}r + b, \tag{15}$$

where A, b become the learning parameter as θ described in the first step of **Algorithm 1**. In (15), continuously updated matrix A can compensate the inaccurately tuned K_{PD}, and the feed-forward gain b helps approach the reference fast in a tracking scenario [19].

PILCO allows to use any differentiable polices including a nonlinear policy such as a Gaussian policy. Without loss of generality, the proposed hybrid approach can combine a nonlinear policy. When the Gaussian policy is defined as

$$\pi_{\text{RBF}}(x, \theta) = \sum_{i=1}^{l} w_i \phi_i(x), \tag{16}$$
$$\phi_i(x) = \exp\left(-\frac{1}{2}(x - \mu_i)^{\text{T}} \Sigma_i^{-1} (x - \mu_i)\right),$$

with the learning parameter $\theta = \{w_i, \Sigma_i, \mu_i\}$, the PD-RL with the Gaussian policy policy is as follows:

$$\pi_{\text{PD-RBF}} = \pi_{\text{RBF}}(x, \theta) + K_{\text{PD}}(x - r). \tag{17}$$

The dimensionality of the learning parameter of the Gaussian policy is larger than the dimensionality of the linear policy's. Selecting a policy between the linear and nonlinear ones may depend on complexity of a learning objective.

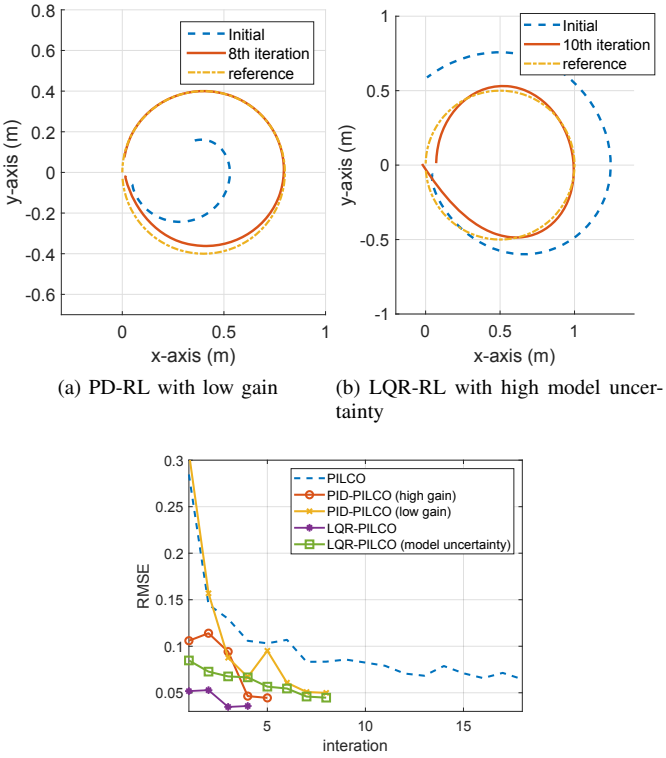
2) LQR-RL: LQR is one of optimal control methods, which derives the control input to minimize a penalty cost. Different to PD-RL, the LQR-RL method can be used when a linear or linearized system model is available. Since linear or linearized models are preferred by many engineers, there are many possible applications of the LQR-RL such as a quadrotor control. When a point mass model and a 1st order dynamics are considered in 2D space, i.e., $x = [\text{x}, \text{y}, \dot{\text{x}}, \dot{\text{y}}]^{\text{T}}$, the quadrotor system can be linearized as follows:

$$\dot{x} \approx Fx + Gu, \tag{18}$$

where

$$F = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\frac{1}{\tau_x} & 0 \\ 0 & 0 & 0 & -\frac{1}{\tau_y} \end{bmatrix}, G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \frac{1}{\tau_x} & 0 \\ 0 & \frac{1}{\tau_y} \end{bmatrix}, \tag{19}$$

and the time constants τ_x and τ_y can be experimentally obtained by [9]. Approximation error by incorrectly tuned τ_x and τ_y could be considered as model uncertainty.



(a) PD-RL with low gain

(b) LQR-RL with high model uncertainty

(c) RMSE of tracking with respect to learning iterations

Fig. 2: Comparison of tracking results given a circular reference; the number of the final learning iteration differs to each algorithm, and all the hybrid algorithms outperform the benchmarked PILCO in terms of the learning convergence rate and the control performance.

Similar to the PD controller, u_{LQR} is also defined as the feedback form:

$$u_{\text{LQR}} = K_{\text{LQR}}(x - r). \tag{20}$$

Given the quadratic cost function in (9), the gain K_{LQR} is obtained by

$$K_{\text{LQR}} = -R^{-1}GP, \tag{21}$$

and P is calculated by the algebraic Riccati equation given by

$$P = PF + F^{\text{T}}P - PGR^{-1}G^{\text{T}}P + Q. \tag{22}$$

The tuning parameters are the weights Q and R in (9). Because the LQR tends to generate conservative control input, LQR-RL can yield a stable initial exploration. When a linear RL policy is used, the hybrid policy is defined as

$$\pi_{\text{LQR-RL}} = (A + K_{\text{LQR}})x - K_{\text{LQR}}r + b. \tag{23}$$

LQR-RL with the Gaussian policy defined in (16) is as follows

$$\pi_{\text{LQR-RBF}} = \pi_{\text{RBF}}(x, \theta) + K_{\text{LQR}}(x - r). \tag{24}$$

For simulation study, we compare performances of two LQR-RLs using each linear and Gaussian policy in Sec. III.

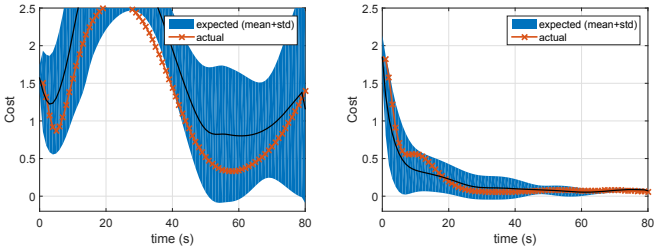
III. Simulation

The proposed algorithm is evaluated for a quadrotor velocity control when position and velocity references $r = [r_{\mathrm{x}}, r_{\mathrm{y}}, \dot{r}_{\mathrm{x}}, \dot{r}_{\mathrm{y}}]^{\mathrm{T}}$ are given . The initial position is estimated with addictive noise Σ_0, i.e, $\hat{x}_o \sim \mathcal{N}(\mathbf{0}, \Sigma_0)$. The control frequency is 10 Hz and the quadrotor is given 8 sec to track a reference trajectory, so 80 training samples are collected from one rollout. The attitude control is assumed stabilized and the height control is fixed. We focus on learning the velocity control $u_{\dot{\mathrm{x}}}, u_{\dot{\mathrm{y}}}$ on 2D space. Although the low-level controllers are predefined, a careful velocity control is required to prevent the quadrotor from crashing.

In the first simulation, the executed methods are 1) the original PILCO, 2) PD-RL with high gain, 3) PD-RL with low gain, 4) LQR-RL, and 5) LQR-RL with model uncertainty, and a circular trajectory is given as the tracking reference. Herein, the high gain in PD-RL meets the condition for largely initialized K_p and K_d; it tries to approach the reference hasty. The low gain leads less aggressive and delayed tracking results. For the LQR-RL evaluation, we add model uncertainty represented by the time constant parameters τ_x and τ_y defined in (19). Incorrectly tuned parameters make the LQR controller misunderstand dynamics, and this causes negative initial tracking performance. However, as the learning repeats, the LQR-RL policy is expected to remedy inaccuracy of the LQR controller. We set the weight parameters $K_p = 3$ and $K_d = 0.1$ for PD-RL with high gain, and $K_p = 0.2$ and $K_d = 0.2$ for PD-RL with low gain. The well-tuned linearization parameters for LQR are set as $\tau_x = \tau_y = 0.105$ and the perturbed parameters to include model uncertainty are set as $\tau_x = \tau_y = 0.001$. The LQR gain K_{LQR} is calculated by (21) with the cost $Q = 15 \cdot I_{4\times4}$ and $R = I_{2\times2}$, where $I_{p\times p}$ indicates the p-by-p identity matrix.

Fig. 2 illustrates the tracking results of the compared methods, where the initial and the final trajectories on 2D view as well as the reference trajectory are shown in (a) and (b), and the tracking results represented by root mean square error (RMSE) according to the number of learning iterations are shown in (c). The initial tracking result is made without using any pre-trained policy so that all hybrid polices at the initial iteration are almost same to the pure PID or LQR controller. The original PILCO takes 18 iterations to converge and the converged policy produces a bias error after the learning termination. The PD-RL with the low gain shows the slow tracking to reach the reference at the beginning and takes 8 iterations to complete the learning as shown in Fig. 2(a). The PD-RL with the high gain presents the overshoot trajectory firstly, and then converges to the stable tracking trajectory quickly. When an accurate system model is given without model uncertainty, LQR-RL gives the best performance in terms of the smallest error and the shortest iteration. In Fig. 2(b), although the model uncertainty is added, the LQR-RL still presents the successful learning performance.

To see more detail of the difference between the PD-RL and the LQR-RL, Fig. 3 presents the expected cost and the actual cost after the first learning. The expected cost is obtained by the policy evaluation at the 12th step in **Algorithm 1** and the actual cost is obtained by implementing the learned policy. Because the modelling algorithm is based on the probabilistic Gaussian process, the expected cost is represented by mean and standard deviation. The large deviation indicates unreliability of the learned model, which refers to the requirement for more policy search. To complete the learning, the deviation should be as small as possible and the expected cost should fit to the actual cost. Therefore, the expected cost history is an empirical evidence to validate a convergence rate. In comparison to the LQR-RL in which the actual cost fits in the expected cost deviation as shown in Fig. 3(b), the PD-RL produces large deviation in Fig. 3(a) at the first iteration. Although both methods finally converge in the simulation study as shown in Fig. 2(c), the large deviation problem in the PD-RL causes failure of the flight learning experiments in our experience.



(a) PD-RL with low gain (b) LQR-RL

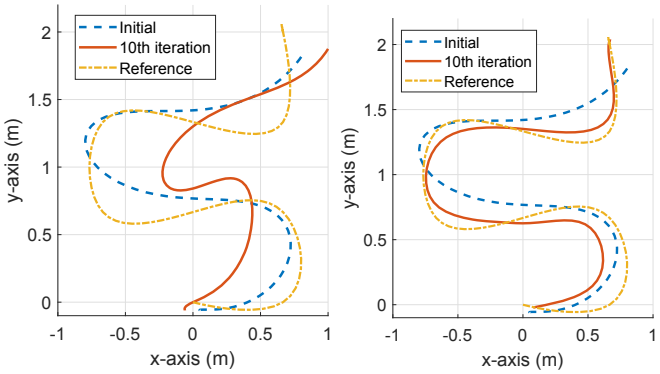
Fig. 3: Expected cost $J^{\pi}(\theta)$ in (28) at the first policy evaluation and the actual cost are compared. The expected cost is represented as a Gaussian random variable with mean and variance. Large variance indicates unreliability of the learned model and requirement for more policy search.

Lastly, we compare two LQR-RLs using each the linear policy in (23) and the Gaussian nonlinear policy in (24). The Gaussian policy has more learning parameters than the linear one, which has more potential for a complex tracking. As a challenging tracking problem, we test a spline trajectory reference in which position and velocity references are changing dynamically rather than the circular reference. For both polices, we tune the parameters to have the best performance. Fig. 4 shows the tracking results and the RMSE plot from which the Gaussian policy outperforms the linear policy.

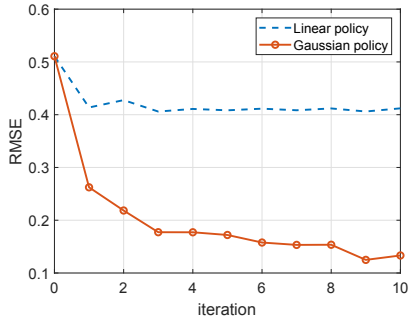
IV. Experiment

The circular trajectory tracking is given to the flight learning experiment and Crazyflie 2.0 quadrotor is used. The state of the quadrotor is measured by the motion capture system (VICON) and the computer having i7 6700K 4.0GHz CPU implements the learning and control on Robot operating system (ROS) environment. Fig. 5(a) shows the experimental view.

For experiment, we implement the LQR-RL excluding the original PILCO and the PD-RL due to the significantly slow convergence and the risky flight as described in the previous Section III. The model parameters $\tau_x = 0.105$ and $\tau_y = 0.105$



(a) LQR-RL with linear policy (b) LQR-RL with Gaussian policy



(c) RMSE of tracking with respect to learning iterations

Fig. 4: Gaussian policy v.s linear policy of LQR-RL for a spline trajectory tracking.

in (19), the cost weights $Q = 15 \cdot I_{4\times4}$ and $R = 0.001 \cdot I_{2\times2}$ in (9) are set, and the LQR gain K_{LQR} in (21) is calculated

$$K_{\text{LQR}} = \begin{bmatrix} -1.5340 & 0 & -0.6862 & 0 \\ 0 & -1.5340 & 0 & -0.6862 \end{bmatrix}.$$

Fig. 5 shows the learning and tracking results and the learning terminates by the second iteration. Initially the LQR-RL yields the delayed tracking on both position and velocity as shown in Fig. 5(b). After, the position error is reduced by producing more active control in Fig. 5(c). After only two iterations, the learning result shown in Fig. 5(d) is improved with the reduced velocity error. The video of the experimental flight can be found at HTTPS://YOUTU.BE/ZKWAPQPSNIU.

V. CONCLUSIONS

This paper presented a new hybrid RL control algorithm by combining PD and LQR control methods to PILCO algorithm. Both the methods improved the convergence rate to complete the learning much than the original PILCO algorithm. They were evaluated by the quadrotor tracking control and the experimental results validated the outstanding performance with respect to the rapid convergence and control performance.

APPENDIX

The state x follows Gaussian probability $p(x)$ with mean μ_x and covariance Σ_x, represented by

$$p(x) = \mathcal{N}(\mu_x, \Sigma_x), \tag{25}$$

and the derivative with respect to the probabilistic state is given by

$$\frac{dE_x[l(x)]}{dp(x)} = \left\{ \frac{dE_x[l(x)]}{d\mu_x}, \frac{dE_x[l(x)]}{d\Sigma_x} \right\}. \tag{26}$$

When K is defined as K_{PD} in (13) or K_{LQR} in (21), the cost function in (9) is as follows:

$$\begin{aligned} l(x) = (x-r)^{\text{T}} \left(Q + K^{\text{T}}RK\right)(x-r) \\ + x^{\text{T}} \left(2A^{\text{T}}RK^{\text{T}} + A^{\text{T}}RA\right) x \\ + \left(2b^{\text{T}}RA + 2b^{\text{T}}RK - 2r^{\text{T}}K^{\text{T}}RA\right) x \\ + b^{\text{T}}Rb - 2b^{\text{T}}RKr. \end{aligned} \tag{27}$$

The expectation of the immediate cost $l(x)$ is in the following

$$E_x[l(x)] = \int l(x)p(x)dx, \tag{28}$$

which can be determined by (25) and (27). The random variable $x^{\text{T}}Cx$ for any $C \in R^{d\times d}$ follows non central chi square whose mean is $\text{tr}(W\Sigma_x)$ and variance is $2\left(\text{tr}(W\Sigma_x) + 2\mu_x^{\text{T}}W\mu_x\right)$ such that

$$x^{\text{T}}Wx \sim \chi(\text{tr}(W\Sigma_x), \mu_x^{\text{T}}W\mu_x), \tag{29}$$

where $\text{tr}(\cdot)$ is matrix trace. According to (25) and (29), the expectation of the cost in (28) is as follows:

$$\begin{aligned} E_x[l(x)] = \text{tr}\left((Q + K^{\text{T}}RK + 2A^{\text{T}}RK^{\text{T}} + A^{\text{T}}RA)\cdot\Sigma_x\right) \\ + (\mu_x - r)^{\text{T}}(Q + K^{\text{T}}RK)(\mu_x - r) \\ + \mu_x^{\text{T}}(2A^{\text{T}}RK^{\text{T}} + A^{\text{T}}RA)\mu_x \\ + \left(2b^{\text{T}}RA + 2b^{\text{T}}RK - 2r^{\text{T}}K^{\text{T}}RA\right)\mu_x \\ + b^{\text{T}}Rb - 2b^{\text{T}}RKr. \end{aligned} \tag{30}$$

Finally, the gradient of the expectation of the cost in (26) is calculated as

$$\frac{dE_x[l(x)]}{d\mu_x} = 2(\mu_x - r)^{\text{T}}(Q + K^{\text{T}}RK) \tag{31}$$

$$+ 2\mu_x^{\text{T}}(2A^{\text{T}}RK^{\text{T}} + A^{\text{T}}RA) \tag{32}$$

$$+ \left(2b^{\text{T}}RA + 2b^{\text{T}}RK - 2r^{\text{T}}K^{\text{T}}RA\right), \tag{33}$$

$$\frac{dE_x[l(x)]}{d\Sigma_x} = Q + K^{\text{T}}RK + 2A^{\text{T}}RK^{\text{T}} + A^{\text{T}}RA. \tag{34}$$

REFERENCES

[1] B. Demirel, A. Ramaswamy, D. E. Quevedo, and H. Karl, "DeepCAS: A deep reinforcement learning algorithm for control-aware scheduling," *IEEE Control Systems Letters*, vol. 2, no. 4, pp. 737–742, 2018.

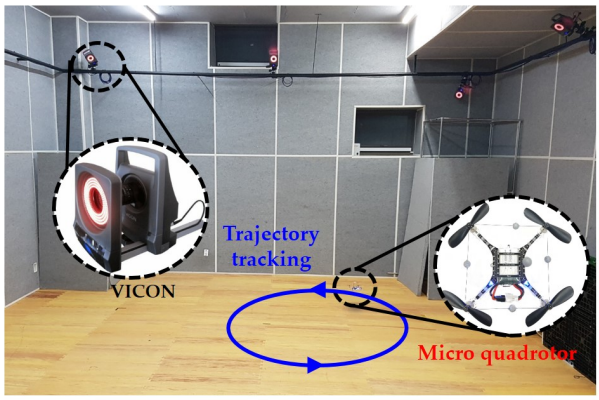
[2] J. Xie, Y. Wan, K. Mills, J. J. Filliben, and F. L. Lewis, "A scalable sampling method to high-dimensional uncertainties for optimal and reinforcement learning-based controls," *IEEE Control Systems Letters*, vol. 1, no. 1, pp. 98–103, 2017.

[3] X. Liang, M. Zheng, and F. Zhang, "A scalable model-based learning algorithm with application to uavs," *IEEE Control Systems Letters*, vol. 2, no. 4, pp. 839–844, 2018.

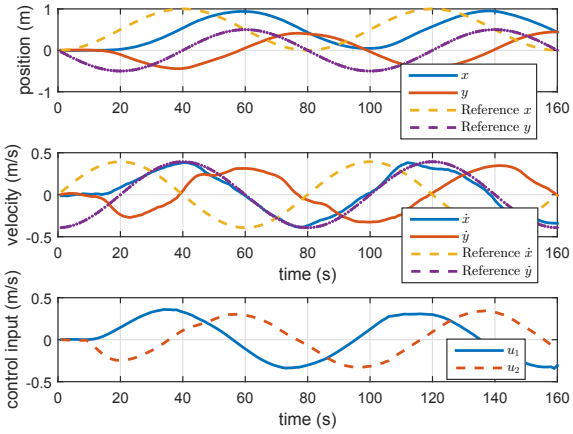
[4] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[5] M. Deisenroth and C. E. Rasmussen, "PILCO: A model-based and data-efficient approach to policy search," in *Proceedings of the International Conference on machine learning*, 2011, pp. 465–472.

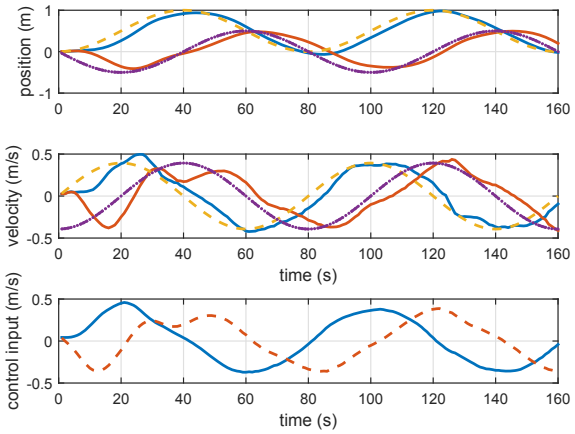
[6] S. Levine and V. Koltun, "Guided policy search," in *International Conference on Machine Learning*, 2013, pp. 1–9.



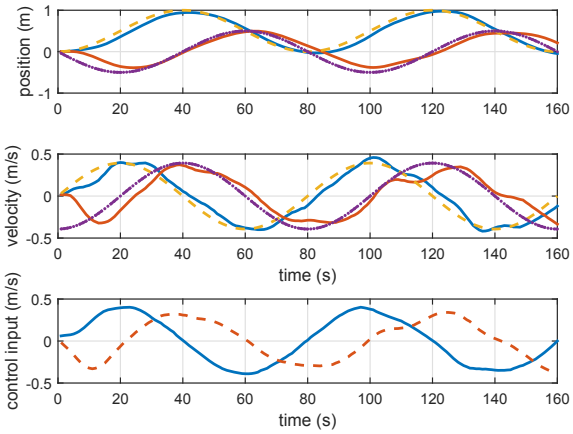
(a) Experimental environment



(b) Initial iteration, RMSE=0.3668



(c) 1st iteration, RMSE=0.1358



(d) 2nd iteration (learning completion), RMSE=0.0940

Fig. 5: The trajectory tracking results of LQR-RL until the second reinforcement learning iteration.

[7] S. Levine and P. Abbeel, "Learning neural network policies with guided policy search under unknown dynamics," in *Advances in Neural Information Processing Systems*, 2014, pp. 1071–1079.

[8] T. Zhang, G. Kahn, S. Levine, and P. Abbeel, "Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search," in *IEEE International Conference on Robotics and Automation*, 2016, pp. 528–535.

[9] M. Kamel, T. Stastny, K. Alexis, and R. Siegwart, "Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system," in *Robot Operating System*. Springer, 2017, pp. 3–39.

[10] I. Koryakovskiy, M. Kudruss, H. Vallery, R. Babuška, and W. Caarls, "Model-plant mismatch compensation using reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2471–2477, 2018.

[11] A. Doerr, D. Nguyen-Tuong, A. Marco, S. Schaal, and S. Trimpe, "Model-based policy search for automatic tuning of multivariate pid controllers," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 5295–5301.

[12] Y. Gal, R. McAllister, and C. E. Rasmussen, "Improving PILCO with bayesian neural network dynamics models," in *Data-Efficient Machine Learning workshop, International Conference on Machine Learning*, vol. 4, 2016.

[13] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[14] G. Sebastian, Y. Tan, D. Oetomo, and I. Mareels, "Feedback-based iterative learning design and synthesis with output constraints for robotic manipulators," *IEEE Control Systems Letters*, vol. 2, no. 3, pp. 513–518, 2018.

[15] Q. Guo, Q. Wang, Z. Zuo, Y. Zhang, D. Jiang, and Y. Shi, "Parametric adaptive control of electro-hydraulic system driving two-dof robotic arm," in *IEEE Conference on Decision and Control*, 2017, pp. 3283–3288.

[16] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *IEEE International Conference on Robotics and Automation*, 2017, pp. 285–292.

[17] K. Lobos-Tsunekawa, F. Leiva, and J. Ruiz-del Solar, "Visual navigation for biped humanoid robots using deep reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3247–3254, 2018.

[18] N. M. Nasrabadi, "Pattern recognition and machine learning," *Journal of Electronic Imaging*, vol. 16, no. 4, p. 049901, 2007.

[19] N. Amann, D. H. Owens, and E. Rogers, "Iterative learning control using optimal feedback and feedforward actions," *International Journal of Control*, vol. 65, no. 2, pp. 277–293, 1996.