Adapting with Latent Variables in Model-Based Reinforcement Learning for Quadcopter Flight

Rachel Li

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-77
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-77.html

May 28, 2020

Acknowledgement

To everyone who has been a part of my journey, thank you.

Adapting with Latent Variables in Model-Based Reinforcement Learning for Quadcopter Flight

by Rachel Li

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

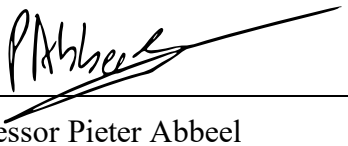
Committee:

Professor Sergey Levine
Research Advisor

May 26, 2020

(Date)

* * * * * * *



Professor Pieter Abbeel
Second Reader

26-MAY-2020

(Date)

ADAPTING WITH LATENT VARIABLES IN MODEL-BASED REINFORCEMENT LEARNING FOR QUADCOPTER FLIGHT

Rachel Li
Electrical Engineering and Computer Science
University of California, Berkeley
rachel_li@berkeley.edu

May 26, 2020

ABSTRACT

Real-world robots will require adaptation to a wide variety of underlying dynamics functions. For example, an autonomous delivery drone would need to fly with different payloads or environmental conditions that modify the physics of flight, and a land robot might encounter varying terrains during its runtime. This paper focuses on developing a single sample-efficient policy that adapts to time-varying dynamics, applied to a quadcopter in simulation that carries a payload of varying weight and a real mini-quadcopter carrying a variable string length hanging payload. From the sample-efficient PETS policy, our approach learns a dynamics model from data and learns a context variable to represent a range of dynamics. At test time, we infer the context that best explains recent data. We evaluate this method both on a simulated quadcopter and a real quadcopter, the Ryze Tello. For both scenarios, we illustrate the performance improvements of our method in adapting to different dynamics compared to traditional model-based techniques. Supplemental materials and videos can be found at our website: https://sites.google.com/view/meta-rl-for-flight.

1 Introduction

When operating robots in the real world, there are often many factors that can affect their dynamics, including changes in weather, onboard equipment failure, and others. These can be unpredictable and highly detrimental to performance. Accounting for these when developing a model typically requires domain-specific knowledge about how these factors will affect the dynamics. However, we propose applying meta-learning to model-based reinforcement learning to create an efficient system for adapting to unknown conditions.

Recently, model-based reinforcement learning (MBRL) algorithms have greatly increased performance to rival model-free algorithms while achieving greater sample efficiency [4, 13]. Our goals in this paper are two-fold. First, we apply PETS to real-world conditions and applications. Second, we explore variational inference techniques [7] to improve the generalization capability of these algorithms to real-world conditions.

We specifically consider a quadcopter in simulation, as it offers enough simplicity in modeling dynamics while still allowing for meaningful changes in dynamics. In order to better support adaptation to the changing quadcopter physics at test time, such as weight or string length, one contribution that we make to these algorithms is the introduction of latent variable models, which will be optimized online. This will additionally allow the dynamics model to be trained to represent a continuous spectrum of dynamics contexts.

We then test the ability of PETS to learn to operate on a real world quadcopter and apply the latent variable model to validate our method of adaptation. In particular, we aim to control 1) a quadcopter in simulation carrying a payloads of various weights and 2) a real quadcopter with a hanging payload with various length strings as their dynamics change. Latent variables can be used in this setting to learn to adapt to hardware failures (such as propeller failure), changing environment conditions (like wind), or task-related changes (like changes in weight from pick-up or drop-off). In our work, we will be using the latent variable for task-related changes.

2 Related Work

The following section is based on the background research conducted in Belkhale et al. [2].

In the realm of flight control, prior work has been done to achieve various tasks including: flight through tight spaces [11], aerobatics [10], and obstacle avoidance [16]. However, these methods require prior knowledge about the con-

figuration of the quadcopter, which must then be manually configured.

Certain work has attempted to solve this problem by creating a system with unknown variables to represent the unknown configuration [17, 6]. Then, using automatic system identification, the varying parameters are automatically learned. However, these methods still require domain knowledge about how the variations affect the dynamics in order to program their effect into the system.

In order to solve this problem, meta-learning, a framework for essentially learning how to learn, can be applied. In this formulation, the effect of different configurations is learned alongside the dynamics. Prior work in using meta-learning for quadcopter flight has been shown to improve model accuracy but with no significant benefit in control for actual task performance [14].

Our work improves upon the existing work by eliminating the need for domain knowledge when preparing for quadcopter flight. We demonstrate an improvement in task performance when using our approach compared to standard model-based reinforcement learning methods in a challenging setting that requires fast adaptation. Parts of this report are based on collaborative work [2].

Our latent variable formulation was based off the research done by Perez, Such, and Karaletsos [15]. Their work implemented a variational inference approach to learn an explicit representation of unknown environment properties through latent variable models. This accelerates learning in such environments during training and, more importantly, facilitates generalization on novel environments at test time. For example, in our quadcopter environment, we modeled the latent variable for task-related changes, like the mass of the payload and the varying-length string from which the payload is hanging.

In addition, our latent variable formulation was built on the foundation of the PETS algorithm by Chua et al., which we discuss in the next section [4]. This algorithm, probabilistic ensembles with trajectory sampling (PETS), works to achieve performance similar to model-free algorithms while retaining the sample efficiency capable of model-based algorithms.

3 Model-Based Reinforcement Learning

3.1 General Formulation

In the model-based reinforcement learning framework, we have a conditional distribution for the next state given the current state and action, and parameterized by the specific parameters of our model. Thus, MBRL strives to fit an approximation to the actual state transition function for the data given in terms of N $\{\{s_t, a_t\} \rightarrow s_{t+1}\}$ pairs, which we denote as $\mathcal{D}^{\text{train}}$.

$$s_{t+1} = s_t + f_\theta(s_t, a_t)$$

Using the dataset, we are able to train the parameters θ of the neural network using maximum likelihood:

$$\begin{aligned}\theta^* &= \arg\max_\theta \; p(\mathcal{D}^{\text{train}}|\theta) \\ &= \arg\max_\theta \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}^{\text{train}}} \log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\,. \quad (1)\end{aligned}$$

Once the dynamics model, f_θ, has been estimated, this model is then utilized to evaluate multiple candidate action sequences with this estimated dynamics model after getting the resulting state trajectories. Then, using the expected rewards for all these candidate sequences, we can then find the best candidate sequence to maximize the reward obtained as follows

$$A^* = \arg \min_{a_t, a_{t+1}, \ldots, a_{t+H-1}} \sum_{t'=t}^{t+H-1} c(\hat{s}_{t'}, a_{t'})$$

where we have $\hat{s}_{t'+1} = \hat{s}_{t'} + f_\theta(\hat{s}_{t'}, a_{t'})$. While model-based learning algorithms have been shown to be much more sample-efficient as compared to model-free algorithms, the asymptotic performance achieved on some standard complex learning tasks has been significantly lower. For this reason, Chua et al. propose an algorithm, PETS, which we discuss in the next section, and which will be our baseline algorithm on which we test the value of learning latent or context variables [4].

3.2 PETS Algorithm

Probabilistic ensembles with trajectory sampling (PETS) is an algorithm proposed by Chua et al., that combines uncertainty-aware deep network dynamics models with sampling-based uncertainty propagation. PETS consists of three main components, as seen in Figure 1, which we will be relying on as a baseline method to support this study.

First, the algorithm determines a dynamics model by using multiple randomly initialized neural networks, in what effectively works as an ensemble of bootstrapped probabilistic models. The ensemble of probabilistic models aims to capture two types of model uncertainty, aleatoric and epistemic, where the use of probabilistic networks explains the inherent stochasticities of the system, and the ensembles allow us to account for the subjective uncertainty about the dynamics function. Then, the algorithm uses a particle-based propagation algorithm to propagate different state trajectories through the different models. Specifically, P particles are created from the current state and each particle is propagated by one of the bootstrap models in the ensemble (in the TS(∞) variation). Finally, the algorithm plans and optimizes for a sequence of actions, using a model predictive controller (MPC) [5, 8, 13]. While a random sampling shooting method is used traditionally, PETS improves on this by using the cross-entropy method for optimization (CEM), which samples actions from a distribution closer to previous action samples that yielded high reward, selecting the action sequence with the highest predicted reward:

$$\mathbf{a}_t^* = \arg\max_{\mathbf{a}_t} \left[\max_{\mathbf{a}_{t+1:t+H}} \sum_{\tau=t}^{t+H} \mathbb{E}_{\mathbf{s}_\tau \sim p_\theta} \left[r(\mathbf{s}_\tau, \mathbf{a}_\tau) \right] \right]. \quad (2)$$

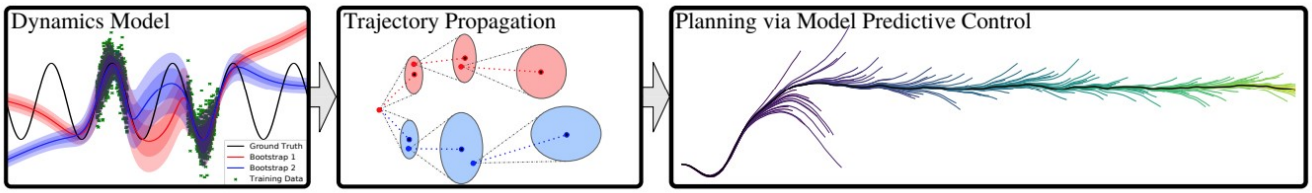


Figure 1: PETS algorithm model. Figure reproduced from Chua et al. [4]

A summarized version of this algorithm is shown in Algorithm 1, reproduced from Belkhale et al. [2].

Algorithm 1 Model-Based Reinforcement Learning

1: Initialize dynamics model p_θ with random parameters θ
2: **while** not done **do**
3: Get current state $\mathbf{s}_t$
4: Solve for action $\mathbf{a}_t^*$ given p_{θ^*} and $\mathbf{s}_t$ using MPC ▷ see (2)
5: Execute action $\mathbf{a}_t^*$
6: Record outcome: $\mathcal{D}^{\text{train}} \leftarrow \mathcal{D}^{\text{train}} \cup \{\mathbf{s}_t, \mathbf{a}_t^*, \mathbf{s}_{t+1}\}$
7: Train dynamics model p_θ using $\mathcal{D}^{\text{train}}$ ▷ see (1)
8: **end while**

4 Environments

4.1 Quadcopter Simulation

We adapted a quadcopter simulator created by Somil Bansal in order to validate the performance of PETS in this domain. The simulator environment was based off the Crazyflie, a small and versatile quadcopter. This quadcopter was chosen because of its size and ability to control at fine detail. In order to simulate the dynamics of the system, we are using a simple nonlinear dynamics model.

The state space of our quadcopter includes the Cartesian coordinate position (x, y, z), the velocities in each direction, the angles (roll α, pitch β, yaw γ), and the angular velocities. The action space includes the roll, pitch, and yaw rate, which allows us to control the quadcopter for finer maneuvers. In our simulation, we fixed the thrust in the z-direction so that the quadcopter stays at a constant height above the ground. This means that our controls effectively only take effect in a 2D plane.

To control the quadcopter, we created a proportional–integral–derivative (PID) controller on top of the quadcopter simulator. Given a desired action, the PID controller performs a control-loop mechanism to reach the setpoint using proportional, integral, and derivative correction terms. We opted for a PID controller because it provides accurate control. The coefficients for the proportional, integral, and derivative terms were hand-tuned to achieve the desired performance.

5 Latent Variable Framework

In adding the latent variable, we adapted our optimization for our neural network parameters from Equation 1 to the following:

$$\begin{aligned}\theta^* &\doteq \arg\max_\theta \log p(\mathcal{D}^{\text{train}}|\mathbf{z}_{1:K}, \theta) \\ &= \arg\max_\theta \sum_{k=1}^{\text{K}} \sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}_k^{\text{train}}} \log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}_k)\,.\end{aligned} \tag{3}$$

In order to find the appropriate value for our latent variable, we would like to maximize the posterior: $p(\theta, \mathbf{z}_{1:K}|\mathcal{D}_{1:K}^{\text{train}})$. However, this is computationally intractable, so we instead based our framework off the formulation specified by Perez, Such, and Karaletsos [15] where we maximize the probability of our seen dataset, the evidence lower bound (ELBO), which is an approximate estimate of the posterior. Essentially, we are performing regularized regression for a maximum likelihood cost function.

$$\begin{aligned}&\log p(\mathcal{D}^{\text{train}}|\theta) \\ &= \log \int_{\mathbf{z}_{1:K}} p(\mathcal{D}^{\text{train}}|\mathbf{z}_{1:K}, \theta) p(\mathbf{z}_{1:K}) \mathrm{d}\mathbf{z}_{1:K} \\ &= \sum_{k=1}^{K} \log \mathbb{E}_{\mathbf{z}_k \sim q_{\phi_k}} p(\mathcal{D}^{\text{train}}|\mathbf{z}_k, \theta) \cdot \frac{p(\mathbf{z}_k)}{q_{\phi_k}(\mathbf{z}_k)} \\ &\geq \sum_{k=1}^{K} \mathbb{E}_{\mathbf{z}_k \sim q_{\phi_k}} \left[\sum_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{D}_k^{\text{train}}} \log p_\theta(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{z}_k) \right] \\ &\qquad - \mathrm{KL}(q_{\phi_k}(\mathbf{z}_k)||p(\mathbf{z}_k)) \\ &\doteq \mathrm{ELBO}(\mathcal{D}^{\text{train}}|\theta, \phi_{1:K}),\end{aligned} \tag{4}$$

From this formulation, we find that the cost that the network uses is composed of two parts: a Kullback-Leibler regularization term and a log likelihood probability. The KL divergence term prevents our latent variable from straying too far from our prior, where the prior is a standard normal Gaussian distribution. For the log likelihood, we use the log-Gaussian probability of each observation given the predictions of the dynamics model at that time step.

$q_\phi(\cdot)$ is the distribution of the latent variable $\mathbf{z}_k$, and we represent this with $\mathcal{N}(\mu^{\mathbf{z}}, \Sigma^{\mathbf{z}})$, where $\mu^{\mathbf{z}}$ and $\Sigma^{\mathbf{z}}$ are the variables we learn directly. p_θ is the model predicted probability

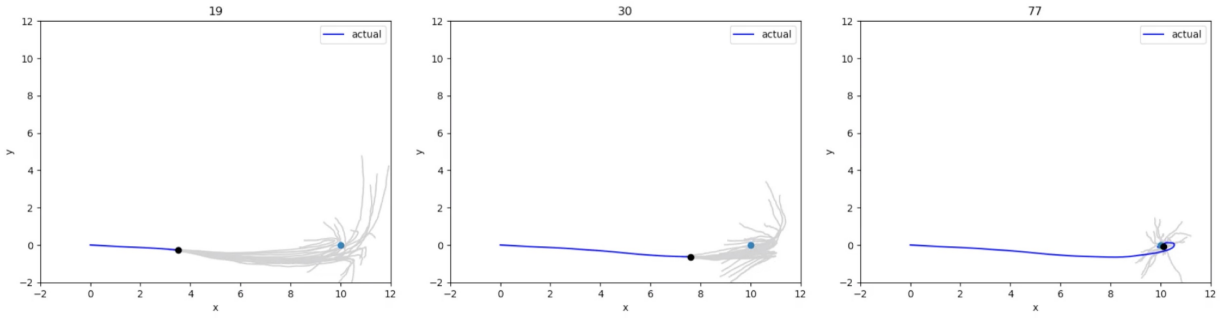


Figure 2: Quadcopter single goal flight at various time steps. The actual trajectory is shown in dark blue with the goal in light blue. All evaluated trajectories are shown in gray. When the quadcopter is furthest from the goal (left), the trajectories are much longer than when the quadcopter is closer to the goal (middle). This difference is because the planning recognizes that the goal is further away. When the quadcopter is at the goal and needs to hover to maintain its position (right), the trajectories are much shorter and try to keep the quadcopter centered in its position.

of the given transition. $p(\cdot)$ is our prior which is $\mathcal{N}(0, 1)$. We use a structured latent variable, meaning the latent variable represents the weight of the payload in the case of simulation or the length of the string in the case of the real quadcopter.

In order to normalize the values of the payload weight, we are actually predicting the standard deviation of the weight from the previously seen distribution of weights. To estimate the value of the latent variable, we are learning two values: the mean and standard deviation. This will essentially allow us to estimate a distribution from which we believe the latent variable is sampled.

Learning the latent variable is composed of two parts. During training time, we learn the dynamics model with the value of the known latent variable passed in as part of the state. During test time, we can either use a known or unknown latent variable model. In the situation where there are known dynamics variables, we are able to directly use the ground-truth as input. However, when the dynamics variables are unknown, we start with the unknown latent variable initialized to the standard normal Gaussian distribution. Then, we will take a sample from the latent variable distribution and use it along with the state during predictions. Using either an online or an offline dataset, we can pick the best $\mu^{\mathbf{z}}$ and $\Sigma^{\mathbf{z}}$ according to the cost function defined previously.

5.1 Data Collection

During data collection, the quadcopter was flown in simulation with randomized actions, ensuring that the action-space was well covered. We modified the states in the simulation environment to include a concatenated history of past states and actions. This MBRL controller with history along with the vanilla version of PETS provided two baseline model-based reinforcement learning controllers that were used for comparisons during testing.

5.2 Offline Learning

During offline learning, we ran the dynamics training on data sets for different latent variables with 50 episodes of data, where each episode consists of 150 timesteps. Then, during test time, we froze the parameters of the dynamics model and only optimized for the latent variable. We preprocessed another set of data of the same size so that we receive tuples of (s, a, s'). With our latent variable distribution $q_{\phi^{\text{test}}}(\mathbf{z}^{\text{test}})$ initialized to be the prior $\mathcal{N}(0, I)$, we randomly sampled batches of the data points across episodes to update the latent variable.

5.3 Online Learning

During online training, we ran the dynamics training similar to offline training. However, during online test-time, we optimized the latent variable based on recent batches of observations. This allowed to learn changes in the latent variable in real time.

6 Experimentation

6.1 Prediction Accuracy

We evaluated the performance of the model by looking at its one-step prediction accuracy across all dimensions, which is shown in Figure 3. We visualized each of the predicted state components individually due to the high dimensionality of the state space. The graphs depict the difference in actual and predicted values for the dimension, where we have sorted by ground truth in each dimension. Predictions on the x, y positions and their velocities are the most accurate. We note that the height of the quadcopter, which corresponds to the z coordinate, was held fixed in the dataset, which may explain its lower prediction accuracy.

6.1.1 Toy Tasks

In order to test the application of PETS to the quadcopter environment, we created a few toy tasks: hovering at a nearby goal position, following a square trajectory, and following a figure-8 trajectory. For each of these tasks, we have provided intermediate planning visualizations of the quadcopter's position in a 2D plane as well as the planning trajectories being

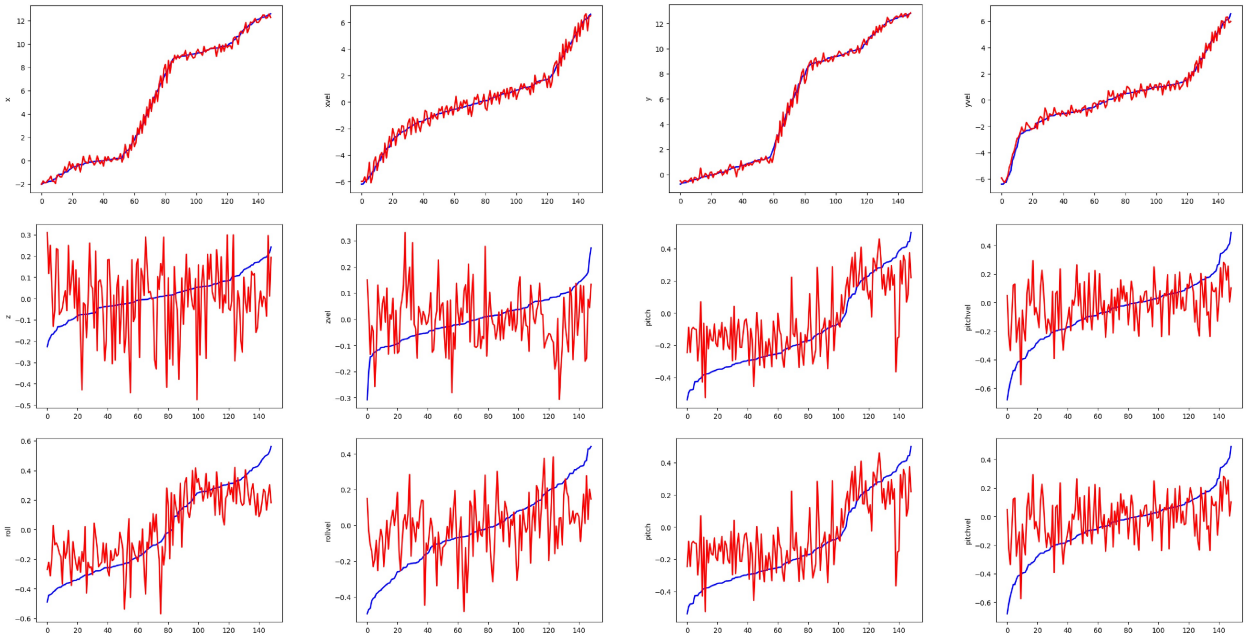


Figure 3: Visualized one-step predictions where the blue line represents the sorted ground truth for each of the 12 dimensions: x, y, z and their velocities, roll, pitch, yaw, and their velocities. The red line denotes the predictions for each respective value. As shown in the figures, the prediction of x, y and their respective velocities is the most accurate.

considered at that time step. The blue dot in each subfigure in Figure 2 represents the waypoint that the quadcopter is trying to reach. Full videos for these toy tasks can be viewed here.

The first task was to reach a single waypoint and hover at the destination. This would test the system's ability to control the quadcopter at a basic level, requiring it to fly in a single direction and hover with stability. We can see in the figures the difference in planning trajectories when we are flying towards the goal and when we are trying to hover at the goal.

An interesting note about this trajectory planning is that we are able to see the algorithm planning ahead to account for momentum, as seen in the evaluated trajectories in Figure 2. When the target waypoint is close, the trajectories are shorter because of the lower speed necessary to produce hovering. When the target waypoint is further, it plans for higher speed flight. As it approaches the waypoint, the trajectory becomes more granular when hovering near the waypoint. This shows that the algorithm plans to reach the waypoint quickly but slows down so that it can hover, taking its momentum into account.

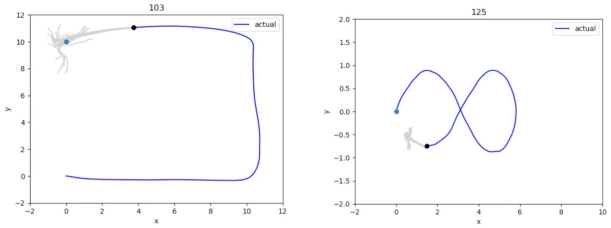


Figure 4: Visualization of simulated quadcopter in mid-flight for square and figure-8 trajectories. The actual trajectory is shown in dark blue with the target waypoint in light blue. The evaluated trajectories are shown in gray.

Our second task consisted of following waypoints in a square formation. This tested the general ability of PETS to control the quadcopter in a simple scenario consisting of mainly flying straight and making sharp 90 degree turns. In the first image of Figure 4, we can see the planning trajectories that PETS is considering as well.

Our final task consisted of following more frequently changing waypoints in a figure-8 formation. This tests the ability of the system to update its trajectory more frequently and to fly in a continuous range of directions. The intermediate planning trajectories can be seen in the second image of Figure 4.

Based on the planning accuracy and ability to follow the trajectories specified by our toy tasks, we were confident in the ability of PETS to operate under our target tasks.

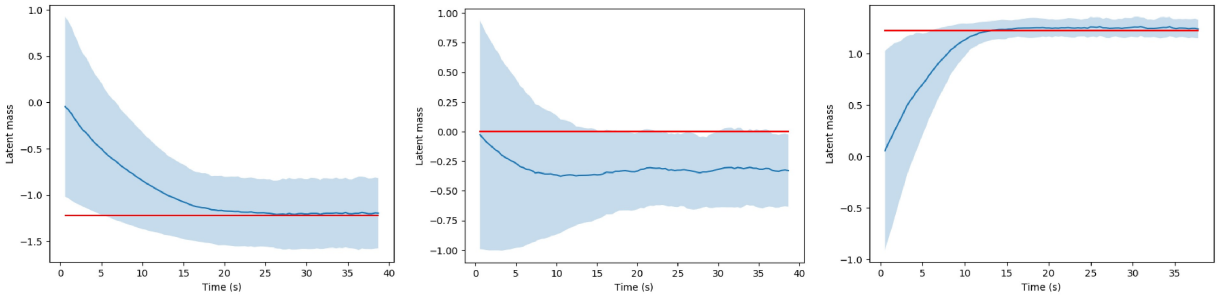


Figure 5: From left to right: No additional weight configuration, 0.05kg additional weight configuration, 0.1kg additional weight configuration. The target latent variable value is shown in red, and the value learned by our algorithm is shown in blue with its standard deviation. Learned latent values for all three weight configurations approach their target, and, more importantly, are distinct, which means the algorithm can distinguish the weight configuration.

6.1.2 Latent Variable Verification

Before adding a latent variable, we tested the limitations of the dynamics model learned by vanilla PETS. We trained a model based on a quadcopter with no additional weight and tested its ability to control a quadcopter with varying amounts of weight to follow the square waypoints toy tasks. We evaluated its performance visually based on its ability to reach all four waypoints, completing the square trajectory. The dynamics model trained on the 0kg additional weight was able to successfully complete the toy task with up to 0.07kg of additional weight, which led us to believe that the dynamics are roughly the same up until that weight. We took this into account when selecting our test weights.

To test the latent variable framework, we collected data on three weight configurations: no additional weight, 0.05kg additional weight, and 0.1kg additional weight. We selected 0.05kg because it was in the range of weights that a model learned on no additional weight could control successfully, and we selected 0.1kg weight as an out-of-range value. Normalized, the corresponding latent values for these configurations were -1.22, 0, and 1.22 for the 0kg, 0.05kg, and 0.1kg weight settings respectively. We tested the latent variable inference using batch sizes of 10, 30, 100, 150, 300, and 1000.

Figure 5 show the results of our best experiment with test-time training batch size of 300 and the speed of latent value convergence in clock time.

We can see from Figure 5 that the latent variable can achieve different values based on the actual parameter values being estimated. The most important takeaway from this test was that the latent framework was able to learn three distinct values for the three weights, which means that algorithm can distinguish the weights.

We concluded that the reason the latent variable did not converge as well to the target values for the two lower masses (0kg and 0.05kg) is because the dynamics with those weights are less distinguishable. This is expected because the quadcopter could fly with up to 0.07kg additional mass when trained on data collected with 0kg additional mass.

7 Joint Work

7.1 Real-world Quadcopter

The following section describes joint work done with Suneel Belkhale in order to apply the meta-learning algorithm to a real-world quadcopter.

After initial experimentation with the Crazyflie, we decided to switch to the Tello Ryze quadcopter for real-world flight due to the larger carrying capacity, sturdier build, and longer duration of flight.

Using Robot Operating System (ROS), we created a cross-network system to minimize lag and allow for quadcopter control using interchangeable learning agents. In this system, we planned using a horizon length of five and sent a new action to the quadcopter every 0.25 seconds, giving us a control frequency of 4HZ. This timestep was chosen to account for two calculations: 1) action selection, which takes roughly 0.05-0.10 seconds and 2) latent optimization. This choice balances reactivity, a long horizon of planning (1.25 seconds), and plenty of time for latent optimization per iteration.

Rather than varying the weight, as we did in simulation, we varied the length of the string carrying the payload because we found that varying the string length changed the dynamics more significantly than mass due to the mitigating effects of the Tello's onboard stabilization controllers. From the Tello, we hung a payload with a varying length string to represent the multiplicity of dynamics functions.

We kept an external camera facing towards the quadcopter, through which we measured the pixel coordinates and area of hanging payload using OpenCV [3]. The state space of the real quadcopter included only these measurements of the payload target. The action space is (v_x, v_y, v_z) of the quadcopter, and we aimed to arbitrarily control the position of the payload in 3D space.

Our training and testing pipeline was modified to include real-world data collection and interfacing with the Tello. The modified pipeline is shown in Figure 6.

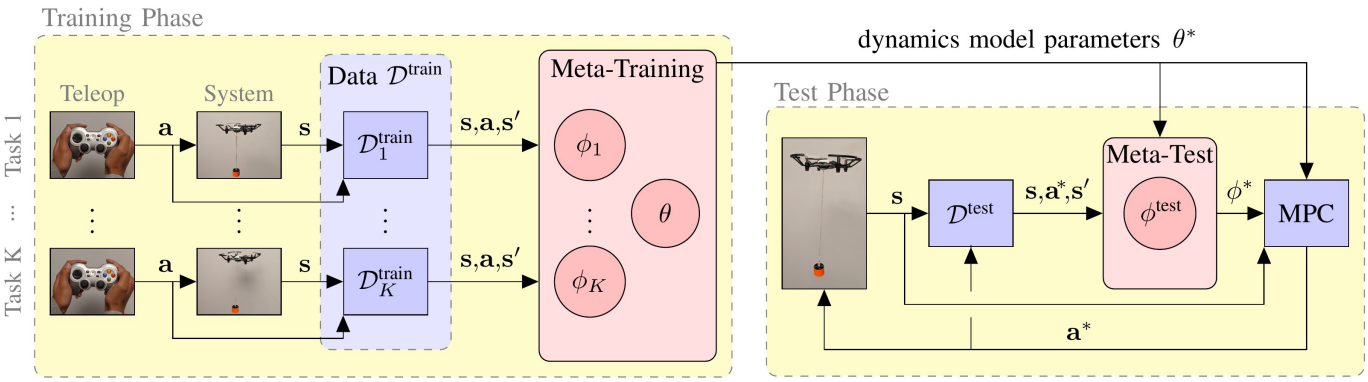


Figure 6: System diagram for the real-world quadcopter created by Rowan McAllister [2]. In the training phase, we manually piloted the quadcopter with various payload configurations, which were concatenated into a single dataset. Then, meta-training is run to learn the neural network weight parameters as well as the adaptation parameters ϕ for each payload. At test time, using the learned dynamics model parameters θ^*, the robot infers the optimal latent variable ϕ^* online using all of the data from the current task. The dynamics model is then used by a model-based controller to plan and execute actions that follow the desired path. As the robot flies, it continues to store data, infer the optimal latent variable parameters, and perform planning until the task is complete.

Algorithm	Avg. Tracking Error (pixels) for each Task Path and Payload String Length (cm)					
	Circle		Square		Figure-8	
	18	30	18	30	18	30
Ours (unknown variable)	**23.62±2.67**	**24.41±3.90**	**23.88±2.81**	**26.57±3.84**	**24.67±1.33**	29.08±6.00
Ours (known variable)	31.81±6.49	30.49±2.65	26.37±3.63	31.68±4.68	29.84±2.84	**28.28±3.76**
MBRL without history	∞	∞	∞	∞	∞	∞
MBRL	39.96±4.40	42.36±2.84	32.37±2.40	39.26±5.16	34.17±1.90	41.01±7.26
PID controller	70.58±4.01	67.98±2.50	65.79±9.99	69.53±6.85	90.15±10.40	86.37±9.27

Table 1: Performance comparison for following the three trajectories (circle, square, figure-8) while carrying payloads with unknown-length strings across variants of our method and existing methods. In all setups, our method achieves the lowest tracking error. Note: A result of infinity denotes that the method caused the quadcopter to exit the camera frame of view and was therefore unable to achieve the task.

7.2 Real-world Quadcopter Testing

7.2.1 Latent Variable Verification

For the Tello experiments, our latent variable represented string lengths of 18CM (latent value -1) and 30CM (latent value 1). Before running latent experiments, we used vanilla PETS as a baseline. First, we collected roughly 10,000 data points for each of our two starting configurations. Then, we trained separate dynamics models with vanilla PETS on these datasets, which we will refer to as the 18CM policy and the 30CM policy. We then tested each policy in each string length configuration and verified that each policy worked well on the environment for which it was tested but performed worse on average for the other environment. This both demonstrates the efficacy of PETS in the real world and validates the need for our latent variable to have one policy for all environments.

After verifying that the string length had a significant effect on dynamics, we trained a latent variable model on both 18CM and 30CM data. Using this model during test-time, we either provided the known value for the latent variable or have the model infer the value from the data.

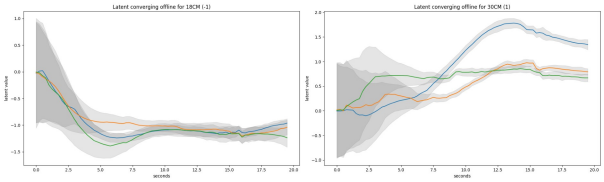


Figure 7: Visualization of offline latent value convergence for Tello quadcopter with 3 trials each created by Suneel Belkhale. On the left, the evaluated string length was 18CM, and the right visualizes the convergence for the 30CM setup. For both, the latents converge between 5-10 seconds on average.

We tested offline latent value convergence by running our latent variable optimization on a playback of recorded data. In this setup, we converged quickly to the desired latent value, usually within 5-10 seconds. Some examples of this can be seen in Figure 7.

Finally, we demonstrate a fully online version of the latent model. This final model was trained on 16,000 data points,

corresponding to 1.1 hours of flight. We used three suspended payload control tasks as benchmarks: (1) square in the plane of image, (2) circle in plane of image, (3) one dimensional horizontal sinusoid motion, a figure-8, parallel to the ground. Convergence to the correct latent mean and variance is the same for the online case as in the offline case, and the performance coverage of the latent online policy is better than both the 18CM and 30CM policies.

Table 1 compares our method to various non-adaptation estimates by calculating the pixel tracking error between the target location on each path versus the actual location of the quadcopter. For all setups, the length of the string was not provided, so meta-learning was necessary in order to adapt to the dynamics of the payload. As seen in the table, our method is able to complete the goal with the lowest tracking error. The standard MBRL algoirthm without history exhibited failure for each of the tests, where failure is defined as either failing to reach the first coordinate (square, circle, figure-8), last coordinate (square task), or running out of frame (square, circle, figure-8). As shown here, the online latent policy shows much better coverage over all dynamics contexts.

Figure 8 shows the qualitative difference between the paths flown by variations of our method and the MBRL controller with past states and actions concatenated. Our method exhibits greater stability and better path-following across all tasks.

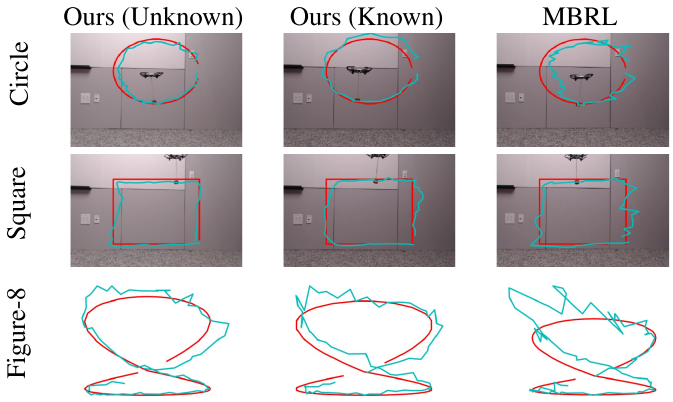


Figure 8: Path-following comparison of our meta-learning approach versus MBRL with past states and actions concatenated. The tasks are to follow a circle or square in the image plane, or a figure-8 parallel to the ground. The target trajectory is shown in red while the path flown is shown in cyan. In all three cases, our methods are better able to follow the trajectory.

7.3 Applications

Outside of the path-following tasks, we also tested our method on a series of tasks that modeled realistic scenarios that a quadcopter may be tasked with when carrying a payload. This section is based on the applications described in Belkhale et al. [2].

7.3.1 Full Pickup and Dropoff Sequence

In the pickup-dropoff sequence, the quadcopter must fly without a payload in a circle trajectory, then pick up a payload, successfully travel in a circle trajectory, and carry it to a specified dropoff location where a person removes the payload. Finally, it must fly steadily in another circle trajecotry without the payload. In this task, since the quadcopter alternates between not carrying and carrying a payload, it must adapt to these changes online in order to succeed. The series of images in Figure 9 shows the quadcopter flying in each stage as well as its ability to learn the appropriate latent value after each transition.

7.3.2 Other Applications

Other tasks we tested included avoiding obstacles while flying along a pre-specified trajectory (Figure 10), flying along directed trajectories (Figure 11), and following a target (Figure 12).

8 Conclusions

In this paper, we have shown the application of model based techniques to a real world application, specifically using PETS with quadcopter simulations and real-world quadcopters. We examined adding a latent variable model to the dynamics function to be able to adapt online to changes in dynamics. We showed that in the real world, this latent model is able to quickly adapt to new dynamics functions. Additionally, the latent model improves performance over traditional MBRL methods.

9 Future Work

One next step will be to test adaptation when the dynamics change multiple times. For example, testing an end-to-end autonomous pickup and drop off. Another avenue of future study will be adaptation to latent values outside of the trained distribution. For example, we might try controlling a 24CM string and seeing if the latent variable value converges appropriately, and then seeing if this performs as well as expected on the real quadcopter.

To augment the algorithm itself, future work can also be done to make the latent trainable as well, which would remove semantic meaning from the value of the latent and allow the algorithm to learn its own values without having to know the precise weight or string length during training time. Additionally, we can add multiple latent dimensions to capture more changes that can affect quadcopter dynamics.

10 Contributions

Rachel worked on creating the quadcopter environment and testing PETS performance in operating the quadcopter with the PID controller. Rachel also worked on implementing the latent variable framework for PETS and testing it with the quadcopter environment.

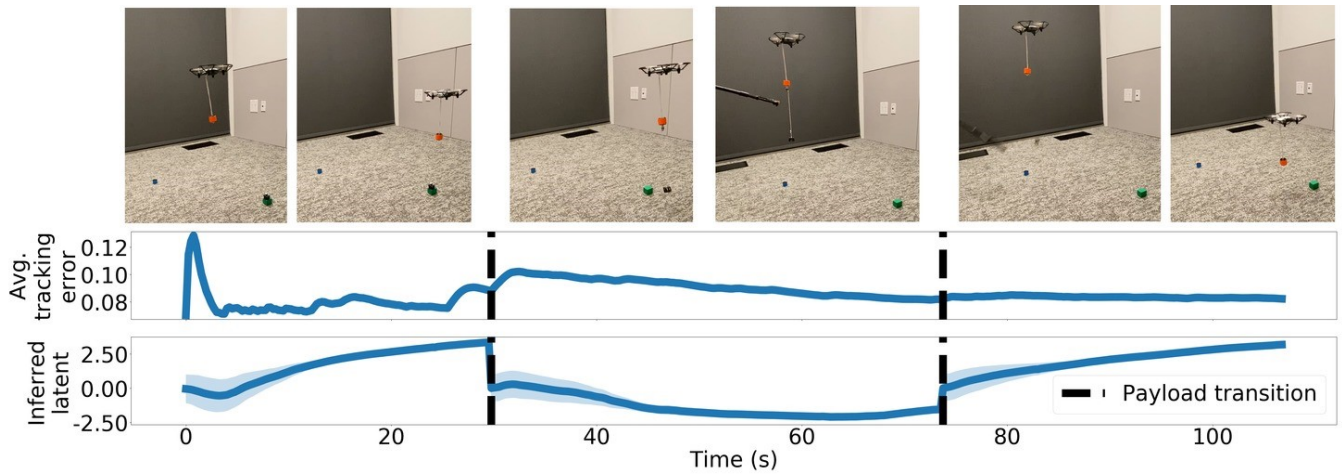


Figure 9: Visualization of our approach successfully completing the full quadcopter payload transportation task. The task consists of three distinct phases: before the quadcopter picks up the payload, while the payload is in transit to the dropoff location, and after the payload is dropped off. During each phase, the our approach uses the collected data to infer the latent value. When the state changes, all data is flushed, and the latent inference continues with new data. As shown in the graphs, the latent values are different when carrying a payload and when flying with no payload, and the latent values converge to the same value when flying without a payload, which demonstrates the success of the method.

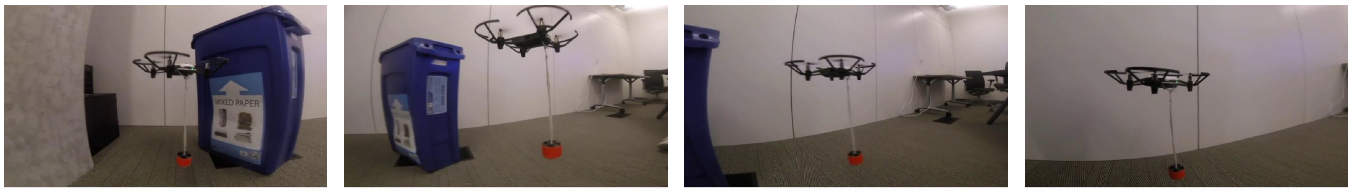


Figure 10: In this scenario, the quadcopter is given a trajectory to follow that avoids the obstacle. By using our method, the quadcopter is able adapt to the payload and maneuver closely around the obstacle.

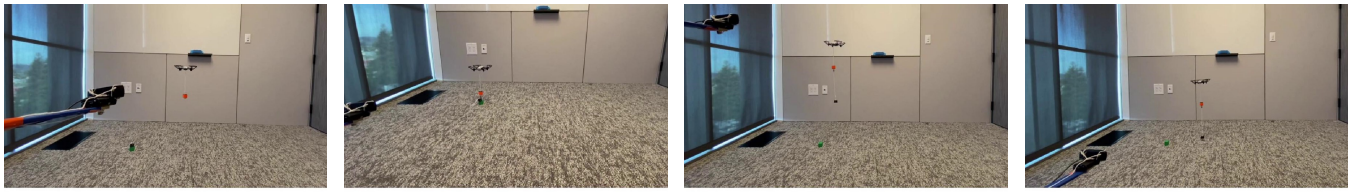


Figure 11: In certain scenarios, like pickup and dropoff, the user may prefer to directly indicate where the payload should go, letting the quadcopter decide how to fly to move the payload towards the desired location. In this experiment, we mounted the external camera onto a stick to create a "wand." The robot's goal is to keep the suspended payload in the center of the camera's field of view at a specific pixel size as the camera moves around.

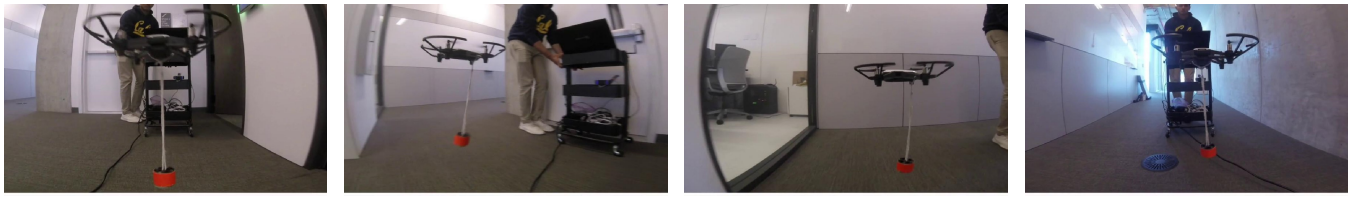


Figure 12: With the goal of keeping the payload in the center of an external camera's field of view, we created a following robot that could follow a user's trajectory around obstacles.

Suneel Belkhale adapted the latent variable framework for the Tello quadcopter and modified PETS to interface with ROS. Suneel also worked on the Tello testing analysis with PETS and created the quadcopter rig.

Rachel and Suneel worked together on testing the final system on the application tasks.

Gregory Kahn contributed to the base quadcopter setup and provided guidance throughout the project.

Rowan McAllister developed the meta-learning algorithm and provided guidance throughout the project.

Roberto Calandra provided guidance throughout the project.

Sergey Levine provided primary guidance throughout the project.

11 Acknowledgements

I would like to thank my research advisor Sergey Levine for the opportunity to work in the RAIL lab and the support during my experience. A special thank you to Rowan McAllister for the guidance throughout.

References

[1] Christopher G Atkeson and Juan Carlos Santamaria. "A comparison of direct and model-based reinforcement learning". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 4. IEEE. 1997, pp. 3557–3564.

[2] Suneel Belkhale et al. *Model-Based Meta-Reinforcement Learning for Flight with Suspended Payloads*. 2020. arXiv: 2004.11345 [cs.RO].

[3] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008.

[4] Kurtland Chua et al. "Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models". In: *CoRR* abs/1805.12114 (2018). arXiv: 1805.12114. URL: http://arxiv.org/abs/1805.12114.

[5] Carlos E Garcia, David M Prett, and Manfred Morari. "Model predictive control: theory and practice—a survey". In: *Automatica* 25.3 (1989), pp. 335–348.

[6] Petros Ioannou and Bariş Fidan. *Adaptive control tutorial*. SIAM, 2006.

[7] Michael I Jordan et al. "An introduction to variational methods for graphical models". In: *Machine learning* 37.2 (1999), pp. 183–233.

[8] Sanket Kamthe and Marc Peter Deisenroth. "Data-efficient reinforcement learning with probabilistic model predictive control". In: *arXiv preprint arXiv:1706.06491* (2017).

[9] Katie Kang et al. "Generalization through Simulation: Integrating Simulated and Real Data into Deep Reinforcement Learning for Vision-Based Autonomous Flight". In: *CoRR* abs/1902.03701 (2019). arXiv: 1902.03701. URL: http://arxiv.org/abs/1902.03701.

[10] Sergei Lupashin et al. "A simple learning strategy for high-speed quadrocopter multi-flips". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010, pp. 1642–1648.

[11] Daniel Mellinger, Nathan Michael, and Vijay Kumar. "Trajectory generation and control for precise aggressive maneuvers with quadrotors". In: *International Journal of Robotics Research* 31.5 (2012), pp. 664–674.

[12] Anusha Nagabandi et al. "Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning". In: *CoRR* abs/1708.02596 (2017). arXiv: 1708.02596. URL: http://arxiv.org/abs/1708.02596.

[13] Anusha Nagabandi et al. "Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 7559–7566.

[14] Michael O'Connell et al. "Meta-Learning-Based Robust Adaptive Flight Control under Uncertain Wind Conditions". In: *Caltech Preprint* (2019).

[15] Christian F. Perez, Felipe Petroski Such, and Theofanis Karaletsos. "Efficient transfer learning and online adaptation with latent variable models for continuous control". In: *CoRR* abs/1812.03399 (2018). arXiv: 1812.03399. URL: http://arxiv.org/abs/1812.03399.

[16] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments". In: *Robotics Research*. Springer, 2016, pp. 649–666.

[17] Jean-Jacques E Slotine and Weiping Li. "On the adaptive control of robot manipulators". In: *International Journal of Robotics Research (IJRR)* 6.3 (1987), pp. 49–59.

[18] Tingwu Wang and Jimmy Ba. "Exploring Model-based Planning with Policy Networks". In: *CoRR* abs/1906.08649 (2019). arXiv: 1906.08649. URL: http://arxiv.org/abs/1906.08649.