Model-Based Meta-Learning for Flight with Suspended Payloads

Suneel Belkhale

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-96
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-96.html

May 29, 2020

Model-Based Meta-Learning for Flight with Suspended Payloads

by

Suneel Belkhale

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Sergey Levine, Chair
Professor Pieter Abbeel

Spring 2020

Model-Based Meta-Learning for Flight with Suspended Payloads

by Suneel Belkhale

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 28, 2020

(Date)

Professor Pieter Abbeel
Second Reader

28 - MAY - 2020

(Date)

Model-Based Meta-Learning for Flight with Suspended Payloads

Abstract

Model-Based Meta-Learning for Flight with Suspended Payloads

by

Suneel Belkhale

Master of Science in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Sergey Levine, Chair

Transporting suspended payloads is challenging for autonomous aerial vehicles because the payload can cause significant and unpredictable changes to the robot's dynamics. These changes can lead to suboptimal flight performance or even catastrophic failure. Although adaptive control and learning-based methods can in principle adapt to changes in these hybrid robot-payload systems, rapid mid-flight adaptation to payloads that have a priori unknown physical properties remains an open problem. We propose a meta-learning approach that "learns how to learn" models of altered dynamics within seconds of post-connection flight data. Our experiments demonstrate that our online adaptation approach outperforms non-adaptive methods on a series of challenging suspended payload transportation tasks. Videos and other supplemental material are available on our website https://sites.google.com/view/meta-rl-for-flight

Contents

Acknowledgments

I would like to thank Professor Sergey Levine for advising me as an undergraduate and graduate student, as well as for providing the resources to pursue this project. I also want to thank my graduate student advisor Gregory Kahn for his mentorship and guidance. This project would also not have been possible without my collaborator Rachel Li, who prototyped the meta-learning algorithm and created the multimedia and figures for this project, and Dr. Rowan McAllister, who developed the meta-learning algorithm. Thanks to Dr. Roberto Calandra for his advice and guidance in developing this project. Section 1.2, portions of Chapter 3, and the figures & tables are shared from our conference paper, which can be found here: https://arxiv.org/abs/2004.11345

Our team also thanks Simon Ramstedt for recommending the Tello drone and Somil Bansal for providing a quadcopter simulator. This research was supported by the National Science Foundation under IIS-1700697 and IIS-1651843, ARL DCIST CRA W911NF-17-2-0181, NASA ESI, the DARPA Assured Autonomy Program, and the Office of Naval Research, as well as support from Google, NVIDIA, and Amazon.

Chapter 1

Introduction

1.1 Motivation

In traditional planning and trajectory optimization, there is an assumption that the system dynamics can be fully characterized prior to the optimization. While this assumption holds when considering a single robot in isolation and for very simple joint dynamics, performing system identification becomes more challenging when the physical properties of the robotic system yield complex dynamics or interactions. For example, a pendulum system may be simple to model in isolation, but modeling dynamics becomes much more complicated if this pendulum is suspended from a helicopter in the presence of variable wind. In most real world cases, unfortunately, the robotic system is never isolated: physical properties of the "controllable" entity cannot be decoupled from the rest of the environment, and so performing one system identification before trajectory optimization becomes challenging in a large number of cases. Additionally, as the robot moves between different contexts, previous assumptions on the dynamics can fall apart. Consider a robotic quadruped tasked with walking through smooth city landscapes versus possibly muddy and bumpy dirt trails in nature. In the best case, a robot should be able to adapt to a wide range of contexts quickly and reliably.

In this thesis, we view designing dynamics models as a continual task. We rely on learning methods to train dynamics models directly from observed data rather than depending on domain experts to perform system identification. Learning methods suffer from a dependence on large quantities of high quality data. When considering online adaptation to new contexts with previous learning methods, it is infeasible to both collect enough representative data to learn about the new context and simultaneously perform well in that context. Therefore sample efficiency in online adaptation is a key challenge.

We specifically consider a quadcopter adapting to the presence of different suspended payloads. The quadcopter can lose control easily if adaptation takes too long, so fast online

adaptation can be studied in this domain. Additionally, the dynamics of suspended payload flight are stochastic and nonlinear, and thus are difficult to model without learning techniques. Most importantly, varying the payload in this domain greatly varies the observed dynamics. For example, decreasing the length of the string produces tighter and faster oscillations.

Our method learns to distinguish the payload context at training and test time within the dynamics model, thereby optimizing for fast adaptation.

1.2 Related Work

Prior work on control for aerial vehicles has demonstrated impressive performance and agility, such as enabling aerial vehicles to navigate between small openings [13], perform aerobatics [11], and avoid obstacles [19]. These approaches have also enabled aerial vehicles to aggressively control suspended payloads [23, 24]. These methods typically rely on manual system identification, in which the equations of motion are derived and the physical parameters are estimated for both the aerial vehicle [12, 27] and the suspended payload [23, 24]. Although these approaches have successfully enabled controlled flight of the hybrid system, they require *a priori* knowledge of the system, such as the payload mass and tether length [4]. When such parameters cannot be identified in advance, alternative techniques are required.

Many approaches overcome the limitations of manual system identification by performing automated system identification, in which certain parameters are automatically adapted online according to a specified error metric [22, 8]. However, the principal drawback of manual system identification—the reliance on domain knowledge for the equations of motion—still remains. While certain rigid-body robotic systems are easily identified, more complex phenomena, such as friction, contacts, deformations, and turbulence, may have no known analytic equations (or known solutions). In such cases, data-driven approaches that automatically model a system's dynamics from data can be advantageous.

Prior work has also proposed end-to-end learning-based approaches that learn from raw data, such as value-based methods which estimate cumulative rewards [25] or policy gradient methods that directly learn a control policy [26]. Although these model-free approaches have been used to learn policies for various tasks [14, 21], including for robots [9], the learning process generally takes hours or even days, making it poorly suited for safety-critical and resource-constrained quadcopters.

Model-based reinforcement learning (MBRL) can provide better sample efficiency, while retaining the benefits of end-to-end learning [3, 6, 15, 2]. With these methods, a dynamics model is learned from data and then used by either a model-based controller or to train a control policy. Although MBRL has successfully learned to control complex systems such as quadcopters [1, 10], most MBRL methods are designed to model a single task with un-

changing dynamics, and therefore do not adapt to rapid online changes in the dynamics of a system.

One approach to enable rapid adaptation to time-varying dynamical systems is *meta-learning*, which is a framework for *learning how to learn* that typically involves fine-tuning of a model's parameters [5, 7, 16] or input variables [18, 20]. There has been prior work on model-based meta-learning for quadcopters. O'Connell et al. [17] used the MAML [5] algorithm for adapting a drone's internal dynamics model in the presence of wind. Although they demonstrated the meta-learning algorithm improved the model's accuracy, the resulting adapted model did not improve the performance of the closed-loop controller. In contrast, we demonstrate that our meta-learning approach does improve performance of the model-based controller. Nagabandi et al. [16] also explored meta-learning for online adaptation in MBRL for a legged robot, demonstrating improved closed-loop controller performance with the adapted model. Our work focuses on suspended payload manipulation with quadcopters, which presents an especially prominent challenge due to the need for rapid adaptation in order to cope with sudden dynamics changes when picking up payloads.

1.3 Contribution Summary

In this thesis, we present a meta-learning algorithm that enables a quadcopter to adapt to various payloads in an online fashion. The algorithm can be viewed as a model-based meta-reinforcement learning method: we learn a predictive dynamics model, represented by a deep neural network, which is augmented with stochastic latent variables that represent the unknown factors of variation in the environment and task. The model is trained with data from different payload masses and tether lengths, using variational inference to estimate the corresponding posterior distribution over these latent variables. This training procedure enables the model to adapt to new payloads at test-time by inferring the posterior distribution over the latent variables. While continuously adapting the model online, a controller uses the model to control the suspended payload along a specified path. We additionally present the hardware and software design that was required to test our algorithm in the suspended payload setup on a commercial quadcopter.

Chapter 2

Background

2.1 Model-Based Reinforcement Learning

In the model-based formulation of Reinforcement Learning, the robot-environment system is represented as a Markov decision process with state $s_t \in \mathbb{R}^{d_s}$ and action $a_t \in \mathbb{R}^{d_a}$, where t is the discrete time step. At each time step, the state updates based on some unknown continuous distribution on next states $p(s_{t+1}|s_t, a_t)$. A trajectory refers to a single sampling pass through of this dynamics distribution using an action sequence $a_{0:H-1}$ and starting at state s_0 to recursively produce states $s_{1:H}$, where H is the trajectory horizon.

Varied contexts are represented as tasks for the robot. The K given tasks are denoted as $\{\mathcal{T}_1, ..., \mathcal{T}_K\}$, and have corresponding finite time horizons $\{H_1, ..., H_K\}$. The goal in each task is to maximize the expected sum of future rewards $r(s_t, a_t) \in \mathbb{R}$ over the time horizon.

In MBRL, the task is to estimate $p(s_{t+1}|s_t, a_t)$ using a neural network with parameters θ using observed state transitions. Specifically, we are given a dataset $\mathcal{D}^{train} = \{E_1, ..., E_N\}$ where $E_i = \{(s_0^i, a_0^i, s_1^i), (s_1^i, a_1^i, s_2^i), ...\}$ represents an episode of data collection over some time horizon. The neural network outputs parameterize the next state distribution, in our case using gaussian mean and variance. Then, the parameters θ of the dynamics model are trained via maximum likelihood:

$$\begin{aligned} \theta^* &= \arg\max_{\theta} \; p(\mathcal{D}^{train}|\theta) \\ &= \arg\max_{\theta} \sum_{(s,a,s') \in \mathcal{D}^{train}} \log p_\theta(s'|s, a)\,. \end{aligned} \tag{2.1}$$

We extend the PETS algorithm [2], which has previously been shown to learn expressive neural network dynamics models and attain good sample efficiency and final performance.

In PETS, multiple dynamics models are trained in parallel. Using Model Predictive Control, one can use the resulting dynamics function to perform trajectory optimization. Through CEM, MPC iteratively improves the expected future rewards for an action sequence $a_{t:t+H-1}$ using the objective:

$$a^*_{t:t+H-1} = \arg\max_{a_{t:t+H-1}} \mathbb{E}_{s_{t:t+H} \sim p_\theta} \left[r(s_{t:t+H}, a_{t:t+H-1}) \right], \tag{2.2}$$

Here, $s_{t:t+H}$ is a trajectory as defined by rolling out the dynamics function as described previously starting at s_t. In MPC, only the first action in the optimized sequence is used at each time step, thereby creating a closed loop control scheme. In Algorithm 1, the full training procedure for online MBRL is given.

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 s_t
4: Solve for action a^*_t given p_{θ^*} and s_t using MPC ▷ see (2.2)
5: Execute action a^*_t
6: Record outcome: $\mathcal{D}^{\text{train}} \leftarrow \mathcal{D}^{\text{train}} \cup \{s_t, a^*_t, s_{t+1}\}$
7: Train dynamics model p_θ using $\mathcal{D}^{\text{train}}$ ▷ see (2.1)
8: **end while**

Chapter 3

Methodology

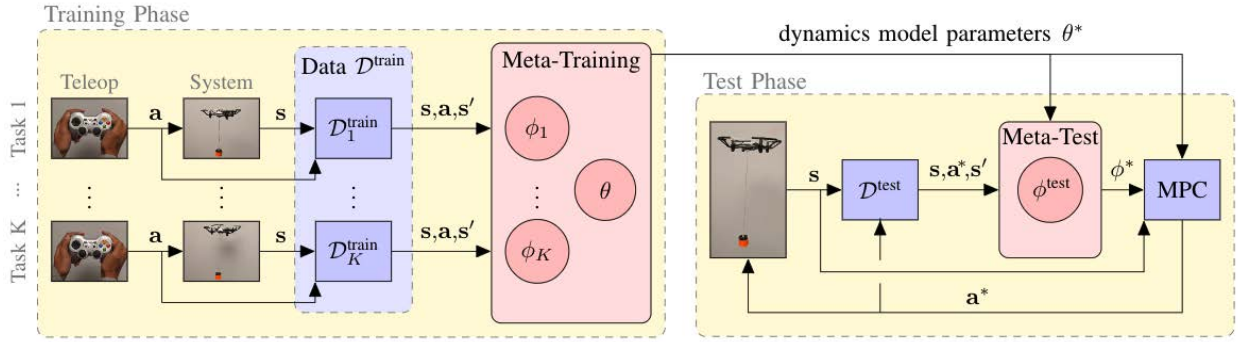

Figure 3.1: System diagram of our meta-learning for model-based reinforcement learning algorithm. In the training phase, we first gather data by manually piloting the quadcopter along random trajectories with K different payloads, and saving the data into a single dataset $\mathcal{D}^{\text{train}}$ consisting of K separate training task-specific datasets $\mathcal{D}^{\text{train}} \doteq \mathcal{D}^{\text{train}}_{1:K}$. We then run meta-training to learn the shared dynamics model parameters θ and the adaptation parameters $\phi_{1:K}$ for each payload task. At test time, using the learned dynamics model parameters θ^*, the robot infers the optimal latent variable ϕ^* online using all of the data $\mathcal{D}^{\text{test}}$ from the current task. The dynamics model, parameterized by θ^* and ϕ^*, is used by a model-predictive controller (MPC) to plan and execute actions that follow the specified path. As the robot flies, it continues to store data, infer the optimal latent variable parameters, and perform planning in a continuous loop until the task is complete.

3.1 Latent Variable Model

We extend PETS by considering an additional context dependent input $z_k \in \mathbb{R}^{d_z}$ to our dynamics model, where k is the current task index. z_k is called the latent variable. The new dynamics function takes the form $p(s_{t+1}|s_t, a_t, z_k)$. While s_t, s_{t+1}, and a_t are known in the data, z_k is unknown since it represents the factors that vary in the dynamics function. In our case, the latent variable for a quadcopter with a suspended payload might represent string length or mass. At test time, our goal is to infer this latent variable from recent state

transitions in order to quickly adapt to new contexts. At training time, the latent variables could be known or unknown, and we evaluate both methods.

3.2 Training with Known Latent Variables

If the factors of variation are known and quantifiable at training time, we use the training dataset $\mathcal{D}^{\text{train}}$ with the dynamics variable z_k for each state transition. In the suspended payload with varying string length example, we segmented the overall training dataset by the string length $(\mathcal{D}_1^{\text{train}}, ..., \mathcal{D}_K^{\text{train}})$ and assigned fixed values to the latent variable input by the corresponding string length. We assume that each episode of collected training data corresponds to a single task. Thus for episode i with task $\mathcal{T}_k$, the state transition $(s_t^i, a_t^i, s_{t+1}^i)$ includes $(s_t^i, a_t^i, s_{t+1}^i, z_k)$. Training here proceeds as normal for PETS on the slightly modified objective:

$$\begin{aligned}\theta^* &\doteq \arg\max_\theta \log p(\mathcal{D}^{\text{train}}|\mathbf{z}_{1:K}, \theta) \\ &= \arg\max_\theta \sum_{k=1}^{\text{K}} \sum_{(s_t,a_t,s_{t+1})\in\mathcal{D}_k^{\text{train}}} \log p_\theta(s_{t+1}|s_t, a_t, z_k)\,. \end{aligned} \tag{3.1}$$

3.3 Training with Unknown Latent Variables

Training with known latent variables assumes we have access to the factors of variation at training time. In many cases, the axes of variation might be hard to identify, segment, and even measure. For a quadruped moving on jagged rocks versus smooth teflon, for example, it is unclear how to represent the context numerically; furthermore, it may be advantageous to let the network handle representation learning. We present an additional formulation that requires only that the user can estimate the dimensionality of the latent variable.

In the training procedure for unknown latent variables, we simultaneously infer the context variables for each task as well as update the dynamics parameters θ. Instead of having fixed context variables for each task, we represent each task with $\phi_k \doteq \{\mu_k, \Sigma_k\}$, where $z_k \sim \mathcal{N}(\mu_k, \Sigma_k)$. At the beginning of training and testing, we set $\phi_k = \mathcal{N}(0, I)$. Our objective takes the form:

$$\theta^* = \arg\max_\theta \max_{\phi_{1:K}} \mathbb{E}[\log p(\mathcal{D}^{\text{train}}|z_{1:K}, \theta)] \tag{3.2}$$

$$= \arg\max_\theta \max_{\phi_{1:K}} \mathbb{E}[\sum_{k=1}^{K} \log p(\mathcal{D}_k^{\text{train}}|z_k, \theta)]. \tag{3.3}$$

$$z_k \sim \mathcal{N}(\mu_k, \Sigma_k)$$
$$\phi_k = (\mu_k, \Sigma_k)$$

In words, this objective aims to find distributions for each task latent variable that allow for a learned dynamics model to have the strongest predictive ability. Our meta-training algorithm optimizes the evidence lower bound (ELBO) as a proxy for the above objective, with gradient steps on $\phi_{1:K}$ and θ performed in an alternating fashion.

3.4 Testing

At test time in both cases, the parameters θ^* are fixed. With meta-training complete, θ^* has been designed to perform well on many tasks, so the challenge now becomes discerning the current context. With only a few transitions in the current episode, our goal is to extract the latent context ϕ^* that maximizes the probability of these transitions occurring.

$$\phi^* = \arg\max_{\phi^{test}} \mathbb{E}[\log p(\mathcal{D}^{\text{train}} | z^{test}, \theta^*)]. \tag{3.4}$$

$$z^{test} \sim \mathcal{N}(\mu^{test}, \Sigma^{test})$$
$$\phi^{test} = (\mu^{test}, \Sigma^{test})$$

The evidence lower bound is used again here. Equation (3.4) is used to calculate gradient steps to improve the predictions of our learned dynamics model as more transitions are collected. Note that with this formulation, it is not necessary to have the task at test time be sampled from the training tasks. Since z is a continuous random variable, our aim was that the meta-training designs a latent space in which interpolation between task latent variables z_k yields a continuous spectrum of tasks. For example, in the suspended payload with string length variation case with $d_z = 1$, we hope that by collecting a short string and a long string dataset, performance generalizes to any string length in between.

Algorithm 2 Model-Based Meta-Reinforcement Learning for Quadcopter Payload Transport

// Training Phase
for Task $k = 1$ to K **do**
  for Time $t = 0$ to T **do**
    Get action a_t from human pilot
    Execute action a_t
    if $\mathbf{z}_k$ is known **then** ▷ case sect. 3.2
      Record outcome: $\mathcal{D}^{\text{train}} \leftarrow \mathcal{D}^{\text{train}} \cup \{s_t, a_t, s_{t+1}, z_k\}$
    else ▷ case sect. 3.3
      Record outcome: $\mathcal{D}^{\text{train}} \leftarrow \mathcal{D}^{\text{train}} \cup \{s_t, a_t, s_{t+1}\}$
    end if
  end for
end for
Train dynamics model p_{θ^*} given $\mathcal{D}^{\text{train}}$ ▷ see (3.3)

// Test Phase
Initialize variational parameters: $\phi^* \leftarrow \{\mu^{\text{test}} = 0, \Sigma^{\text{test}} = I\}$
for Time $t = 0$ to T **do**
  Solve optimal action a_t^* given p_{θ^*}, q_{ϕ^*}, and MPC ▷ see (2.2)
  Execute action a_t^*
  Record outcome: $\mathcal{D}^{\text{test}} \leftarrow \mathcal{D}^{\text{test}} \cup \{s_t, a_t^*, s_{t+1}\}$
  Infer variational parameters ϕ^* given $\mathcal{D}^{\text{test}}$ ▷ see (3.4)
end for

Chapter 4

Experimental Design

4.1 Environment

In this chapter, the design process for experimentation will be outlined. In addition, the iterations and improvements I made will be described in detail. Our goals at the onset of environment design were as follows:

E1: Environment has nonlinear stochastic dynamics.

E2: PETS performs well on each task in the environment with sufficient data from that task.

E3: PETS performance suffers on individual tasks when exposed to all task datasets.

E4: Environment supports a fast reaction time.

E5: Control loop budgets sufficient time for variational inference (latent variable optimization) at test time.

Robot Selection

Quadcopters present highly nonlinear dynamics and are inherently stochastic due to wind and other external noise (**E1**). In addition, having 3D space available during experimentation allows for more complex and interesting demonstrations of adaptation. To minimize risk of damage and maximize safety during data collection and experimentation, we were primarily interested in mini and nano-quadcopters with indoor flying capabilities. I initially ran experiments with the Crazyflie 2.0 from Bitcraze, a nano-quadcopter weighing $\sim$ 27 grams with a 4 minute flight time. While these nano-quadcopters offered flexibility in control and software, I found that many parts were prone to breaking, the flying time was too short, and the quadcopter carrying capacity was minimal. We switched to the Tello from DJI (Fig. 4.1), a mini-quadcopter weighing $\sim$ 80g and supporting 13 minutes of flight. The

Figure 4.1: DJI Tello with suspended payload.

additional thrust capability of the Tello enables a wide range of carrying and transporting tasks. We estimate that the Tello can carry up to $\sim$ 20g of additional mass. Another advantage is that it connects to an external computer via wifi, whereas the Crazyflie only connected via radio. The Tello is limited in software support, but the SDK supports basic velocity control operations. We limited our action space to (v_x, v_y, v_z) since controlling (roll, pitch, yaw) or individual motor thrust yields less flight stability during data collection and experimentation.

Task Selection

In order to create significant diversity in the tasks for the quadcopters (**E2**, **E3**), we explored aerial payload transport. I experimented with a wide range of payloads and connection mechanisms. Due to its onboard downward-facing sensors for optical flow and altitude estimation, I found that the simplest and most reliable tethering approach was to use a string connected at the front of the Tello. The string placement introduces an additional torque and reduces the overall carrying capability, but the payload is less likely to obstruct the Tello sensors for large periods of time.

The payload target itself is a 30cm tall cylinder I designed and 3D-printed using a carbon fiber composite to ensure it is both durable and light-weight. The cylinder has a string attachment to easily attach and detach it from the Tello. By putting neon orange tape on this cylinder, we can easily identify it from an external camera (Fig. 4.1).

The Tello localizes itself through optical flow and IMU data, which leads to moderate drift

over time. This is exacerbated by the increased volatility of the IMU measurements due to the suspended payload. Additionally, when transporting a payload, payload stability and control is a more important goal than just quadcopter stability. Therefore, we keep an external camera facing towards the quadcopter, through which we measure the pixel coordinates and pixel area of the payload target. Specifically, I implemented a simple OpenCV filtering and tracking script to identify the payload by its neon orange color. The observation space of our environment includes only these measurements of the payload target (no quadcopter state measurements).

We considered three main axes of variation for the suspended payload environment: payload mass, payload rigidity, and string length. I noticed that varying the payload mass between 0g and 20g produces a relatively small effect on the overall flight dynamics; this could be because the velocity control of the Tello is robust enough to make the dynamics moderately agnostic to mass and inertia. Payload rigidity is a less intuitive dimension to represent numerically and is also hard to implement in such a restrictive mass range. The variation in string length produces meaningful changes in the oscillation of the suspended payload, so our quantitative experiments primarily consider adaptation to different string length.

Control Setup

Due to its size and weight requirements, the Tello does not support onboard computing. Therefore, we use a System 76 Oryx Pro laptop with an NVIDIA GTX 1070 graphics card to run PETS online. The laptop simultaneously interfaces with the Tello via wifi, receiving and monitoring the Tello state and sending actions through the Tello SDK. Additionally, to receive observations for PETS, a Logitech C290 web camera sends 640x480 images to the laptop via USB.

ROS Network

I used Robot Operating System (ROS) Kinetic to manage internal communication on the laptop. ROS enables modularity in hardware and software components. Figure 4.2 shows the design of the ROS-based communication network. The low latency system design allows for high modularity, full recordings, and flexible recording playback. This allows for arbitrary controllers in the loop, and the ROS network abstraction allows plug and play of any quadcopter, desktop, or control strategy. This choice was validated when we decided to switch to the Tello from the Crazyflie, which required changing only a few lines of code. The key ROS nodes are described below.

MPC: The MPC node receives observations from the External Camera node and the Copter node, runs MPC on the PETS dynamics model based on some user specified cost function,

and then sends actions back to the Copter node. While the MPC node is not running action selection, it performs the optimization described in Section 3.4.

Copter: The Copter node synchronously sends velocity actions to the Tello and asynchronously monitors the Tello state. This includes checking for crashes, lost connections, and low battery. It also performs takeoff and landing when requested by the Data Capture or Joystick node.

Ext Cam: The External Camera node receives images from the web camera, runs simple OpenCV filtering to extract the pixel coordinates and area of the target, and then sends these to the MPC node.

Visualizer: The visualizer node aggregates the information into a visual representation. An example of this is shown in Figure 4.3, with action sequences, overlayed predicted trajectories, goal positions, etc.

Data Capture: the Data Capture node is used as a high level manager of the experimentation flow. This node signals the start and ends of episodes when all participating nodes are "ready" and/or upon a user input through the joystick. It also coordinates takeoff and landing, crash handling, and joystick overrides. Additionally, the data capture node records all the ROS messages into the standard rosbag storage format. Rosbags can be replayed easily for offline testing. This central experiment controller proved invaluable in preventing crashes, streamlining data collection, debugging timing and algorithmic issues, and minimizing latency.

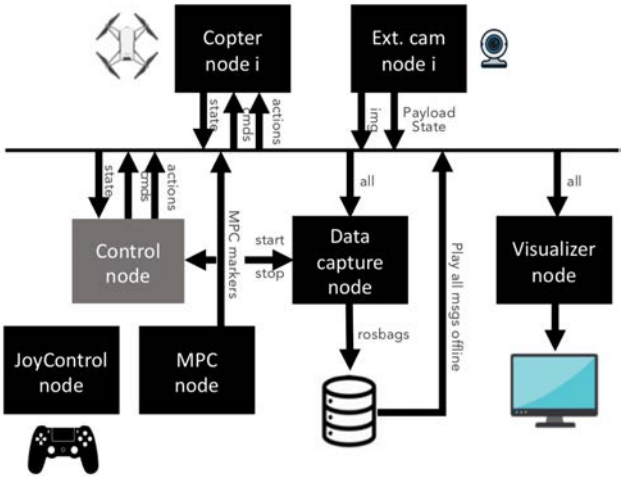

Figure 4.2: ROS Network diagram

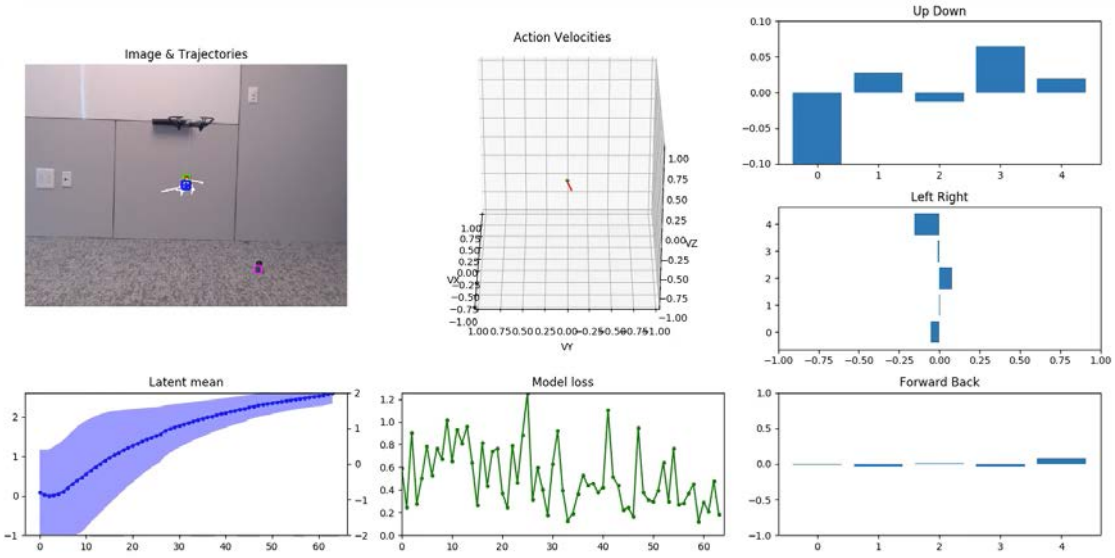

Figure 4.3: ROS visualization node example. Upper left: current external camera image with the goal position (blue), payload target position (green), platform position (magenta), future trajectories (white). Upper middle: 3D cartesian velocity plot of the current action. Lower left: latent variable history (1D) for this episode, showing μ and σ converging. Lower middle: predictive loss history of the model, note that it decreases as the latent value converges. Right column: MPC action selections from $t = \tau$ to $t = \tau + H - 1$ for up-down, left-right, and forward-back axes.

Control Flow

The control flow is as follows: first, the controller computes an action for the latest received observation; then this action gets sent across the network to the quadcopter; the quadcopter then performs the action, all while we continually poll the quadcopter for state information. Due communication and inertial lag, it is often the case that actions do not manifest for several time steps. I mitigated the effects of this by including a state and action history as part of our "state" that gets fed into PETS. I chose the length of this history by picking the history length that minimizes the validation loss of the PETS model.

Our experiments illustrate that actions indeed do not have immediate effects; this can be attributed to communication lag, inertial lag, and observation lag (camera processing time). We estimate that on average, it takes 150ms for actions to be reflected in the state. Therefore, with the design of our system, it became particularly important to ensure that latency is similar at training versus test time, which implies that the latency of action selection must be as small as possible.

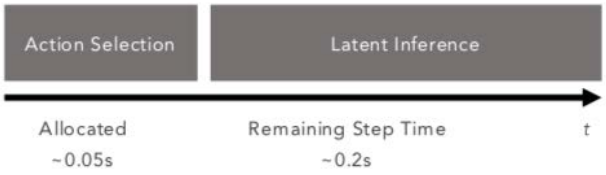

Figure 4.4: Visual diagram for the time split of one control loop. We minimize action selection time (reactivity) and thereby maximize latent inference time (adaptability)

Controller Timing

In additional to overall performance, control frequency is important for both **E4** and **E5**. I experimented with different model sizes and horizon lengths to minimize this action selection time while also allowing for optimal overall performance. I found that a relatively shallow 4 layer network and an action horizon of 5 can operate action selection at a frequency of 10-20Hz (action selection time of 0.05-0.1 seconds). A constant horizon has diminishing predictive power as the overall control frequency gets faster, however. Therefore I chose a time horizon of 1.25 seconds with a horizon length of 5, yielding a Δt of 0.25. This leaves roughly 0.15-0.2 seconds per iteration for the latent inference. The 4Hz control frequency reflects a balance in trade-off space between reactivity of the controller (react within 0.25 seconds), latent computation time (60-80% of total time), and long predictive horizons (1.25 seconds into future).

Note that it is very important to devote all computational resources to action selection at test time since this minimizes overall control lag as discussed previously (we want to send our choice in action as soon as possible). Therefore, we do not have a separate thread running latent computation in parallel. Instead, we rigidly separate the two computations. This can be seen visually in Figure 4.4.

Data collection

Our previous PETS code base was written in Tensorflow and was not flexible enough to support our algorithmic additions. Therefore, I created a PyTorch repository to support latent variable training and testing in PETS. This repository implements the MPC node described above. Since quadcopter flight is sensitive to small changes in actions, I collected training and validation data offline via manual joystick flight rather than using random actions or training PETS online. Additional noise was injected on top of the expert actions to improve the coverage of the dataset. Roughly 40 minutes of data (roughly 10k data points) were needed to learn dynamics models on each task.

4.2 Evaluation

During experimentation, we hoped to compare prior approaches with our approaches in our selected environment. Our goals for experiment design were as follows:

X1: Quantitatively demonstrate adaptability of our method compared to non-adaptive methods on the string pickup task.

X1.1: Show adaptation on a set of known trajectories for each method.

X2: Compare the known latent variable with the unknown latent variable formulation.

X3: Demonstrate that inferring the latent variable reduces trajectory following error.

X4: Provide an end-to-end task to demonstrate real world efficacy.

Model Architecture

As discussed in Section 4.1, we use a fully connected neural network with 4 hidden layers of width 200 for all experiments. The raw inputs to the model consist of the current pixel coordinates and area, a past history of states and actions, and the latent variable. Through validation, the optimal history length was found to be 8 (2 seconds of history) for most tasks. Instead of passing in the raw past states, I found that passing in deltas between each consecutive time step minimized over-fitting to the high dimensional input. Additionally, the pixel area is not normally distributed across data collection, which can create problems for network training. By taking the square root of the area, I found that the "side length" has a normal distribution, and is therefore a stronger feature input. The output of the network represents a delta state, such that the predicted next observation is $s_{t+1} = s_t + \Delta s_t$ where $\Delta s_t \sim p_\theta(\cdot|s_t, a_t, z_k)$. The inputs and outputs are normalized across the training dataset so that each dimension is $\mathcal{N}(0, 1)$.

Latent Variable Evaluation

For the Tello experiments, our latent variable represented string lengths of 18CM (latent value -1) and 30CM (latent value 1). In order to properly show **X1.1** and **X3**, I designed three representative evaluation trajectories to follow during adaptation:

(1) **square**: Travels clockwise in a square pattern in the image plane. This is the simplest task and tests straight line following.

(2) **circle**: Travels clockwise in a circular pattern in the image plane. This task is medium difficulty and evaluates curve following.

(3) **figure8**: Travels in a figure-8 pattern in the ground plane. This is the hardest task since it involves forward back control and simultaneous left-right control.

These are fixed length trajectories that can be compared quantitatively with ease. The main metric (cost function) we use for quality is mean squared error between the current position and the goal position along the trajectory for this time step. Note that the squared error terms are weighted by the inverse standard deviation of each dimension in the observation space for consistency. We evaluate each trajectory on baseline policies, our method with known latent variables, and our method with unknown latent variables (**X1**, **X2**).

End-to-End Task

Since our quantitative tests are controlled in both space and time, our qualitative test aims to demonstrate a real world use case of our method that is not constrained to a specific time frame. To us, the most compelling end-to-end task was a full pick-up and drop-off sequence for a payload. Since we require the payload target to be present at all times for recording observations, I designed and 3D-printed an additional ground payload that can be picked up at close range via a light-weight magnet, as shown in Figure 4.5. The magnetic pick-up radius is very sensitive to the weight of the payload, so we decouple the magnet and the ground payload through another 3D printed element that contains the magnet and connects to the ground payload with a string. The payload target has an opposite polarity magnet to pick up the ground payload. 5-10k data points were collected for the payload target only and the payload target + ground payload configurations.

Initially, the Tello follows some trajectory, adapting to just carrying the payload target. The pick up sequence starts with a color-coded platform on the ground specifying the 3D location of the ground payload, which sits on top of the platform. MPC moves the payload target within the magnetic pick-up radius, and then lifts the ground payload into the air. It then proceeds to follow some trajectory while the algorithm adapts to the new context, and then moves to the drop-off location, where a human is waiting to manually detach the magnet and ground payload with a grabber claw. Once the magnet is detached, the Tello once again

Figure 4.5: Our meta-reinforcement learning method controlling a quadcopter transporting a suspended payload in an end-to-end task. This task is challenging since each payload induces different system dynamics, which requires the quadcopter controller to adapt online. The controller learned via our meta-learning approach is able to (a) fly towards the payload, (b) attach the cable tether (payload target) to the ground payload using a magnet, (c) take off, (d) fly towards the goal location while adapting to the newly attached payload, and (e) deposit the payload using an external detaching mechanism.

adapts. This sequence targets autonomous aerial delivery applications and demonstrates the efficacy of our method (**X4**). By enabling continuous adaptation, this sequence also allows us to test the effect of the latent variable on tracking error (**X3**).

Additional Use-Cases

In addition to the end-to-end test, we employed our approach to demonstrate simple obstacle avoidance. For obstacle avoidance, we placed a recycling bin in the flight path of the Tello, and created a way-point sequence for flight to bypass it in the ground plane. This task aims to illustrate the trajectory following capabilities of our method in unknown dynamics contexts, since our method adapts as the Tello completes its trajectory.

In order to further challenge the algorithm, we fixed the Tello way-point in the camera frame but let the camera be moved by a human observer. First, we moved the external camera in a hallway environment with multiple turns to demonstrate this camera following behavior. For the next experiment, using a simple "wand" I designed to hold the camera, we manually provide way-points by moving the camera in 3D space. Since the observation frame is moving, actions are no longer in the same frame of reference as the observations, so this task illustrates the robustness of the learned dynamics models. We perform a pick-up similar to the end-to-end task using this camera following approach to demonstrate our method's ability to adapt even in a moving reference frame.

Chapter 5

Experimental Results

5.1 Baselines

Using the environment described in Section 4.1, I evaluated our meta-learning approach with both known and unknown latents against the following baselines:

1. **MBRL without history**: The state consists only of the current payload pixel location and pixel area.
2. **MBRL with history**: The state consists of the concatenation of past states and actions, and therefore represents a simple meta-learning approach.
3. **PID Controller**: The PID constants were manually tuned based on a new trajectory not used in evaluation. There is one PID controller per cartesian velocity axis, and the state consists of the pixel location and area.

5.2 Results

For each of the methods, I ran five experiments each on the square, circle, and figure8 trajectories to evaluate the trajectory following capabilities of each method. Table 5.1 shows the results for each approach in terms of average pixel tracking error, with visualizations of a subset of the trajectories shown in Fig. 5.1. Both the online adaptation methods (our approach and MBRL) better track the specified goal trajectories compared to the non-adaptation methods (MBRL without history and PID controller) which shows that online adaptation leads to better performance. Our approach meta-trained with unknown latent variables also outperforms our approach trained with known latent variables, which highlights that our approach does not require *a priori* knowledge of the suspended payloads during training to successfully adapt at test time.

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 5.1: Comparative evaluation of our method for the tasks of following a circle, square or figure-8 trajectory with either an 18cm or 30cm payload cable length. The table entries specify the average pixel tracking error over 5 trials, with ∞ denoting when all trials failed the task by deviating outside of the camera field of view. Note that the cable length was not given to any method *a priori*, and therefore online adaptation was required in order to successfully track the specified path. Our method was able to most closely track all specified paths for all payloads.

In addition to our method showing better closed-loop performance, the latent variable of our dynamics model is consistent in its interpretation. Fig. 5.2 and Fig. 5.3 show the inferred latent variable and tracking error while our model-based policy is executing at test time. We observe that the dynamics variable converges to different values depending on the cable length, which shows that our test-time inference procedure is able to differentiate between the dynamics of the two different payloads. More importantly, as the inferred value converges, our learned model-based controller becomes more accurate and is therefore better able to track the desired path.

5.3 End-to-End Pick-up Task

Fig. 5.4 shows sample images of the full end-to-end pick-up and drop-off task at each of the stages. Our approach is able to complete the full task by adapting to each new context it encounters. Note that due to the small magnetic pick-up radius, picking up the ground payload requires a high degree of precision. The best baseline method, for example, was too unstable to reliably pick up the ground payload with the same planned trajectory. By inferring the latent variable value, the planner is better able to follow specified trajectories after each change in dynamics. Importantly, this demonstrates the continuous nature of our approach, with a single iteration of this task taking almost 2 minutes and changing dynamics contexts three times.

5.4 Additional Use-Cases

Fig. 5.5 illustrates our method applied to the obstacle avoidance task. Note that the algorithm simultaneously avoids the obstacle and adapts to the current dynamics context. Fig. 5.6 demonstrates the successful camera following ability in a hallway with multiple

turns and varying lighting, which can affect the OpenCV payload target detection and thus affect observations to our method. In Fig. 5.7, the pick-up portion of the end-to-end task is recreated with the camera wand following technique, and our method successfully adapts to the new context even under the strain of a moving frame of reference.

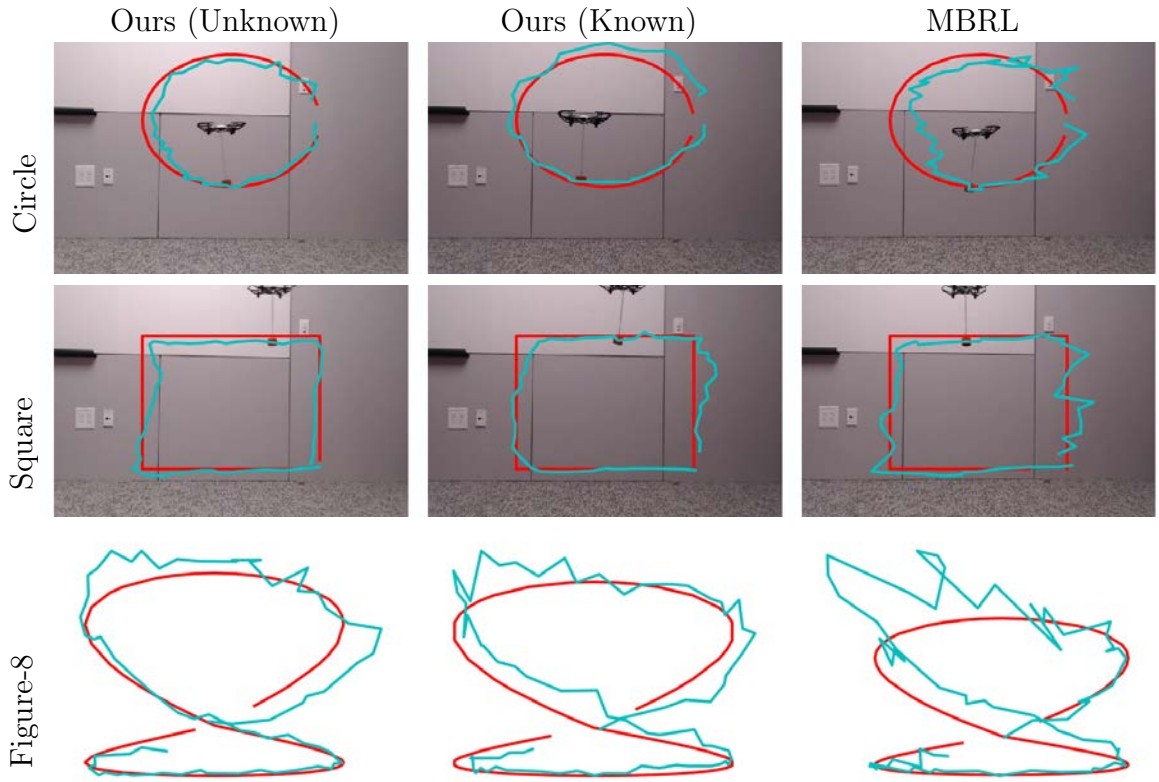

Figure 5.1: Comparison of our meta-learning approach with unknown and known factors of variation versus model-based reinforcement learning (MBRL) with past states and actions concatenated. The tasks are to either follow a circle or square in the image plane, or a figure-8 parallel to the ground. The specified goal paths are colored in red and the path taken by each approach is shown in cyan. Our approaches are better able to adapt online and follow the specified trajectories.

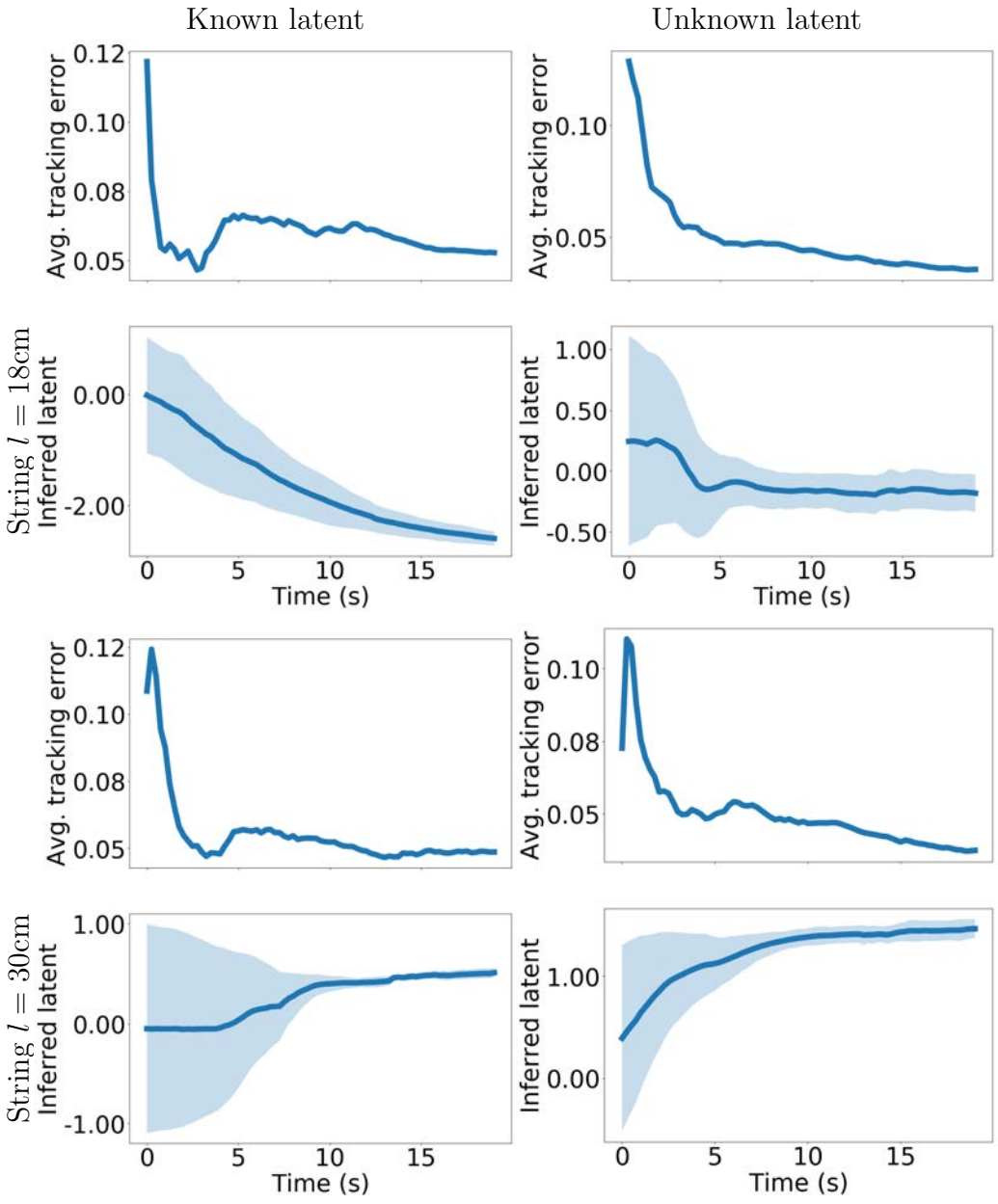

Figure 5.2: Visualization of the inferred latent variable and tracking error over time for the task of following a figure-8 trajectory. We show our approach trained with known variables (left column) and unknown variables (right column) with either a payload cable length of 18cm (top row) or 30cm (bottom row). For all approaches, the inferred latent variable converges as the quadcopter flies and adapts online. The converged final latent values are different depending on the cable length, which shows the online adaptation mechanism is able to automatically differentiate between the different payloads. Furthermore, as the latent value converges, the tracking error also reduces, which demonstrates that there is a correlation between inferring the correct latent variable and the achieved task performance.

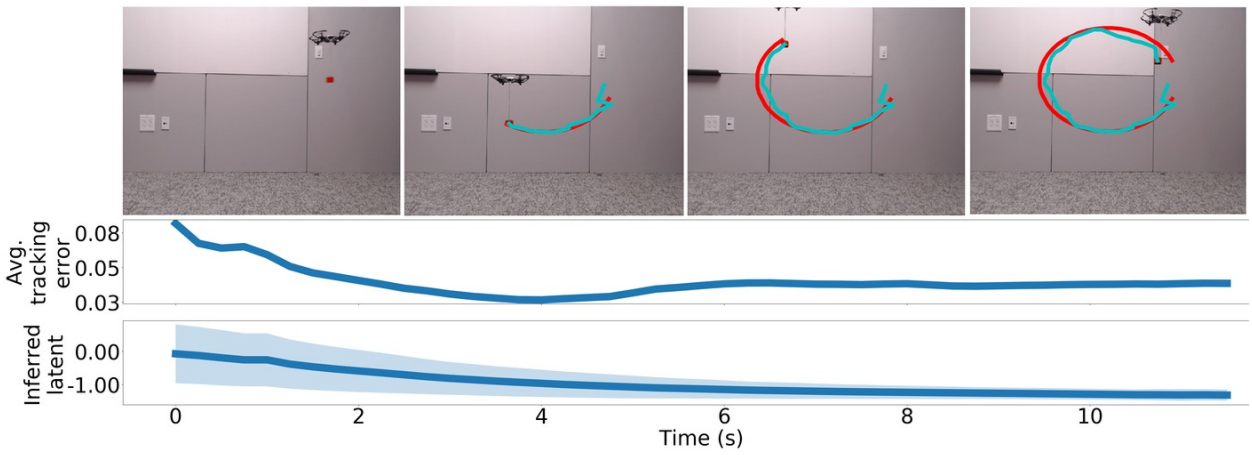

Figure 5.3: As the quadcopter follows the circle trajectory using our model-based controller, our approach adapts online to the *a priori* unknown payload by inferring the latent value which maximizes the dynamics models accuracy. This online adaptation reduces the tracking error as the quadcopter flies, enabling the quadcopter to successfully complete the task.

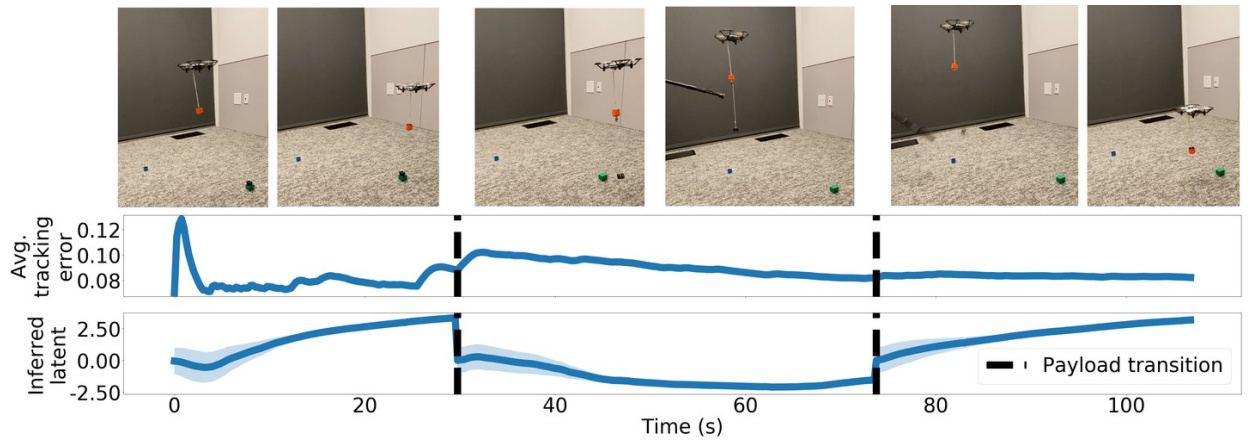

Figure 5.4: Visualization of our approach successfully completing the full quadcopter payload transportation task. The task consists of three distinct phases: before the quadcopter picks up the payload, while the payload is in transit to the goal, and after the payload is dropped off. Our approach continuously adapts the latent dynamics variable online using the current test-time dataset, and flushes the test-time dataset each time the quadcopter transitions between phases, which are delineated by the vertical black lines. The inferred latent variable is the same for when no payload is attached, but different when the payload is attached, which demonstrates that our inference procedure successfully infers the latent variable depending on the payload. Within each phase, the tracking error also reduces over time, which shows that our online adaptation mechanism improves closed-loop performance.

Figure 5.5: Our approach enables a quadcopter to transport a suspended payload around an obstacle. The user first defines a path that goes around the obstacle in the pixel space of the external camera. Our approach then encourages the suspended payload to follow this path while simultaneously adapting to the properties of the suspended payload.

Figure 5.6: Our approach enables a quadcopter to control a suspended payload to follow a target. The target is the external camera that is used to track the suspended payload. Our approach encourages the suspended payload to stay in the center of the camera image and at a specific pixel size, and therefore as the external camera moves, the quadcopter moves in order to keep the suspended payload centered.

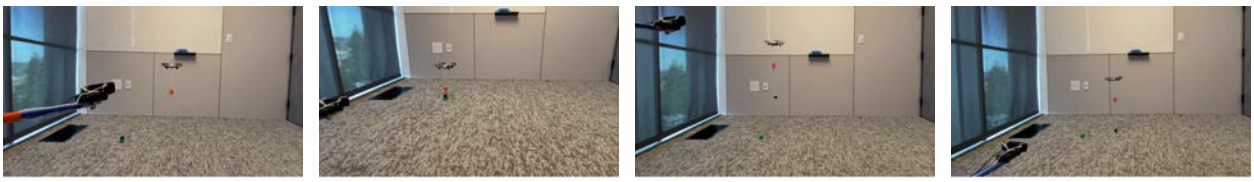

Figure 5.7: Our approach enables a quadcopter to follow trajectories dictated using a "wand"-like interface. The wand consists of mounting the external camera that is used to track the suspended payload on the end of a stick. By defining the cost function to encourage the suspended payload to stay centered, as the user moves the wand, our approach enables the quadcopter to adapt online to the specific payload while keeping the payload centered in the external camera's field-of-view.

Chapter 6

Discussion

6.1 Conclusion

In this project, I presented a meta-learning approach for model-based reinforcement learning and demonstrated online adaptation for a quadcopter with varying suspended payloads. Using PETS, we use a deep neural network to learn the underlying dynamics of each environment. We augment this network by adding a context-specific latent variable as an input to the PETS model, considering both supervised and unsupervised values. The latent variable is optimized to improve the accuracy of the dynamics model both at training and test time. Our experimental setup maximizes the training time for this latent variable while prioritizing fast reaction times for the planning algorithm. Through our quantitative experiments, we observe a meaningful reduction in trajectory following error with our adaptive method compared to adaptive and non-adaptive baselines. Our qualitative experiments demonstrate that the inference of the latent variable does improve closed-loop performance.

6.2 Future Work

While our approach allowed for successful control of a quadcopter with a suspended payload, our control inputs were limited to Cartesian velocities. In order to enable more complex maneuvers and fine-tuned adaptation, one would need lower level control (e.g. pitch, roll, yaw or individual motor thrust). As touched upon in Section 4.1, lower level control brings new challenges that are worth tackling in the future. Additionally, estimating the position of the payload target may be difficult in different conditions with the limited OpenCV approach used in this project. In the future, it will be worthwhile to explore learning directly from images since this could enable even better adaptation. For our end-to-end task, we require manually specifying when the suspended payload was successfully picked up and dropped off in order to reset our latent variable memory; another area of future study is removing this requirement for human oversight so the algorithm becomes fully autonomous.

In addition, future work might extend this algorithm to arbitrary payloads on UAVs with larger thrust capacity. Furthermore, testing in the presence of constraints like wind and low visibility could present additional "contexts" for the meta-learning algorithm. Solving these challenges will enable a wide variety of real-world aerial payload manipulation abilities, and this project represents a meaningful step towards this goal.

Bibliography

[1] Somil Bansal, Anayo K Akametalu, Frank J Jiang, Forrest Laine, and Claire J Tomlin. Learning quadrotor dynamics using neural network for flight control. *IEEE Conference on Decision and Control (CDC)*, pages 4653–4660, 2016.

[2] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *Neural Information Processing Systems (NeurIPS)*, pages 4754–4765, 2018.

[3] Marc Deisenroth and Carl E Rasmussen. PILCO: A model-based and data-efficient approach to policy search. In *International Conference on Machine Learning (ICML)*, pages 465–472, 2011.

[4] Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Automated aerial suspended cargo delivery through reinforcement learning. *Artificial Intelligence*, 247:381–398, 2017. ISSN 00043702. doi: 10.1016/j.artint.2014.11.009.

[5] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, volume 70, pages 1126–1135, 2017.

[6] Yarin Gal, Rowan McAllister, and Carl Edward Rasmussen. Improving PILCO with Bayesian neural network dynamics models. In *ICML Workshop on Data-Efficient Machine Learning*, volume 4, page 34, 2016.

[7] James Harrison, Apoorva Sharma, Roberto Calandra, and Marco Pavone. Control adaptation via meta-learning dynamics. In *NeurIPS Workshop on Meta-Learning*, 2018.

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

[9] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[10] Nathan O. Lambert, Daniel S. Drew, Joseph Yaconelli, Sergey Levine, Roberto Calandra, and Kristofer S. J. Pister. Low level control of a quadrotor with deep model-based reinforcement learning. *IEEE Robotics and Automation Letters (RA-L)*, 4(4):4224–4230, 2019. ISSN 2377-3766. doi: 10.1109/LRA.2019.2930489.

[11] Sergei Lupashin, Angela Schöllig, Michael Sherback, and Raffaello D'Andrea. A simple learning strategy for high-speed quadrocopter multi-flips. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1642–1648, 2010.

[12] Robert Mahony, Vijay Kumar, and Peter Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics & Automation Magazine*, 19(3): 20–32, 2012.

[13] Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *International Journal of Robotics Research*, 31(5):664–674, 2012.

[14] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing Atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[15] Anusha Nagabandi, Gregory Kahn, Ronald S Fearing, and Sergey Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566, 2018.

[16] Anusha Nagabandi, Ignasi Clavera, Simin Liu, and Ronald S Fearing. Learning to adapt in dynamic, real-world environments via meta-reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2019.

[17] Michael O'Connell, Guanya Shi, Xichen Shi, and Soon-Jo Chung. Meta-learning-based robust adaptive flight control under uncertain wind conditions. *Caltech Preprint*, 2019.

[18] Christian F Perez, Felipe Petroski Such, and Theofanis Karaletsos. Efficient transfer learning and online adaptation with latent variable models for continuous control. *arXiv preprint arXiv:1812.03399*, 2018.

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

[20] Steindór Sæmundsson, Katja Hofmann, and Marc Peter Deisenroth. Meta reinforcement learning with latent variable Gaussian processes. *arXiv preprint arXiv:1803.07551*, 2018.

[21] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897, 2015.

[22] Jean-Jacques E Slotine and Weiping Li. On the adaptive control of robot manipulators. *International Journal of Robotics Research (IJRR)*, 6(3):49–59, 1987.

[23] Sarah Tang and Vijay Kumar. Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2216–2222, 2015.

[24] Sarah Tang, Valentin Wüest, and Vijay Kumar. Aggressive flight with suspended payloads using vision-based control. *IEEE Robotics and Automation Letters (RA-L)*, 3(2): 1152–1159, 2018.

[25] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4): 279–292, 1992.

[26] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[27] Xiaodong Zhang, Xiaoli Li, Kang Wang, and Yanjun Lu. A survey of modelling and identification of quadrotor robot. In *Abstract and Applied Analysis*, volume 2014. Hindawi, 2014.